Constructions d'ordres, analyse de la complexité

Guillaume Bonfante

Institut National
Polytechnique de Lorraine

Département de formation doctorale en informatique

École doctorale IAE + M

Constructions d'ordres, analyse de la complexité

THÈSE

présentée et soutenue publiquement le 27 octobre 2000

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine
(spécialité informatique)

par

Guillaume Bonfante

Composition du jury

Président : Jean-Pierre Jouannaud, Professeur

…férences
…esseur

François Lamarche, Directeur de recherches
Jean-Yves Marion, Maître de conférences

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

Remerciements

Pour cette thèse, j'ai été accompagné par de nombreuses personnes. Je tiens à les remercier. Commençons par les rapporteurs :

Jean-Pierre Jouannaud, professeur à l'Université de Paris Sud, Orsay, pour avoir accepté d'être président de jury, et pour les indications qu'il m'a données en vue de la poursuite de mes travaux.

Peter Dybjer, Professeur au « Chalmers University of Technology », ses préoccupations scientifiques étant assez proches des miennes, ses commentaires ont été extrêmement constructifs.

Jacques Jaray, Maître de conférences à l'École des Mines, avec son point de vue pragmatique mais également à la recherche d'une vision plus abstraite de l'informatique.

Je tiens ensuite à remercier ceux qui m'ont aidé tout au long de ces quatre années : Adam Cichon, mon directeur de thèse, pour son accueil, François Lamarche pour tout le travail, le suivi, et l'encadrement qu'il m'a apporté, Jean-Yves Marion avec qui les rencontres fructueuses se sont révélées déterminantes.

Les membres de l'équipe Calligramme ont été pour moi un appui sûr, au début de mon installation, puis par la suite lorsque je les ai rejoints. Marie-Claude Portmann, Jacques Jaray, de l'École des Mines m'ont également soutenu lorsqu'il le fallait.

Enfin, je n'oublie pas les collègues de bureau pour leur gentillesse, leur écoute ; par ordre d'apparition, Zino, Barbara, Daniel, Hélène, Laurent, Catherine, Jérôme et Jean-Yves. Pour la typographie Denis. Anne pour la relecture.

Table des matières

Introduction

Les travaux que nous présentons dans cette thèse se sont articulés autour d'un article de Cichon et Tahhan-Bittar [CTB98], d'un article de Cichon et Lescanne [CL92] et de l'ouvrage (en préparation) de Simmons [Sim00]. Dans [CTB98], Cichon et Tahhan-Bittar présentent une version constructive du théorème de Higman (sur l'ordre de plongement des mots). La technique employée à cette fin est d'utiliser une notion de hiérarchie de fonctions sous-récursives. Ceci nous amène à deux concepts qui jouent un rôle central dans notre étude : les *hiérarchies de fonctions sous-récursives* (et la notion de *termes ordinaux* afférente) et la notion de *preuve de terminaison par interprétation sur un bon ordre.*

Dans son livre [Sim00], Simmons présente un λ-calcul, une extension du système T de Gödel, dans lequel sont définis un type pour les entiers, un type pour les termes ordinaux et les deux itérateurs qui traduisent les principes d'induction correspondant à ces deux types. Il montre en particulier comment utiliser le système pour construire des ordinaux (les plus grands possibles) et les hiérarchies de fonctions sous-récursives.

Enfin, nous mentionnons les travaux de Cichon et Lescanne [CL92]. Dans cet article, le point de vue sur la notion de preuve de terminaison à l'aide d'une interprétation sur un bon ordre est différent. Il s'agit essentiellement d'analyser la preuve de terminaison pour prévoir la complexité ; plus précisément, leur étude est dans le cadre de la réécriture, et Cichon et Lescanne donnent des critères syntaxiques sur les interprétations polynomiales sur les entiers pour borner la com-

plexité des fonctions calculées par de tels systèmes de réécriture.

Le cadre général est donc l'étude de la notion d'ordre, en particulier (mais pas uniquement) dans le cadre de la réécriture. Nous avons donc abordé deux sous-thèmes. Le premier sous-thème concerne l'analyse des preuves de terminaison en réécriture, et plus spécialement leurs conséquences du point de vue de la complexité. Le deuxième sous-thème vise plus spécifiquement à construire des ordres, et des systèmes, des λ-calculs, qui permettent de les manipuler aisément.

Réécriture et complexité

Parmi tous les *modèles de calculs*, nous nous concentrons donc sur la *réécriture*, systèmes d'équations orientées sur les termes d'une algèbre. Elle a retenu notre attention pour deux raisons. Premièrement, cette méthode est de « haut niveau », elle est de ce fait une forme de programmation très intuitive, simple ; la réécriture peut être vue comme une implantation des spécifications algébriques : un type algébrique est la donnée d'une structure, d'opérations sur cette structure et d'équations entre ces opérations ; la réécriture, en orientant les équations, donne un mécanisme pour calculer. Ainsi, la réécriture est le « moteur » sous-jacent de systèmes comme ELAN [1].

Deuxièmement, et à notre avis ce point n'est pas indépendant de ce que nous venons de voir, la formalisation de la réécriture en termes mathématiques est suffisamment simple pour nous permettre de prouver des propriétés intéressantes concernant les systèmes de réécriture. Nous nous sommes intéressés plus particulièrement au problème de la terminaison : « Aurai-je une réponse à ma requête ? » et encore plus précisément au problème de la complexité : « Dans combien de temps aurai-je une réponse à ma requête ? ». Et pour cela, nous nous appuyons sur le fait que les techniques employées pour montrer la terminaison des systèmes de réécriture peuvent, en général, être réutilisées pour analyser la complexité des algorithmes.

Un des outils pour montrer la terminaison des systèmes de réécriture est de plonger la relation de réécriture dans un ordre bien fondé connu au préalable. Mais, en fait, la donnée d'une telle preuve

1. Cf. http://www.loria.fr/equipes/protheo/SOFTWARES/ELAN/index.html.

de terminaison nous donne d'autres renseignements sur le système de réécriture. En particulier, de nombreux exemples montrent qu'une étude plus poussée des preuves de terminaison fournit des bornes sur la longueur de dérivation des systèmes de réécriture, en d'autres termes, le nombre d'étapes nécessaires pour aboutir un calcul. À titre d'exemple, Hofbauer [Hof92b] (et de manière indépendante, Cichon [Cic90]) montre que la longueur de dérivation d'un terme de tout système de réécriture qui admet une preuve de terminaison par l'ordre MPO est au plus récursive primitive.

Mais connaître la longueur de dérivation d'un système ne nous renseigne pas précisément sur la complexité des fonctions calculées. Nous verrons un exemple de méthode de preuve de terminaison (l'ordre de Knuth-Bendix restreint à des symboles de poids strictement positif) dont la longueur de dérivation est au plus exponentielle, mais pour laquelle les fonctions *effectivement* calculées sont calculées (par des machines de Turing) dans l'*espace linéaire* et un autre exemple (les interprétations polynomiales réduites à des polynômes de degré au plus un) dont la longueur de dérivation est également au plus exponentielle mais qui permet de calculer *toutes* les fonctions calculables en *temps exponentiel* (également par des machines de Turing).

En plus de cela, la seule donnée de la longueur de preuve de terminaison ne nous permet pas de résoudre le problème suivant : étant donné une fonction dont la complexité est *a priori* compatible avec la longueur de dérivation, existe-t-il un système de réécriture qui la calcule et qui admet une preuve de terminaison selon la méthode choisie ? Il s'agit en quelque sorte d'un problème de complétude vis-à-vis de la question de la longueur de dérivation qui donne des contraintes sur la complexité des fonctions calculées. Une réponse à cette question témoigne de la richesse (petite ou grande) de la méthode de preuve de terminaison.

Nous nous sommes intéressés à deux méthodes de preuve de terminaison : la méthode par interprétation polynomiale sur les entiers, et la méthode qui s'appuie sur l'ordre de Knuth-Bendix.

La première est sémantique : elle s'appuie sur l'ordre usuel des entiers naturels qui est bien fondé. À chaque étape de réécriture, on se « débrouille » pour que l'interprétation du terme, un entier naturel, diminue. Dès lors, nous sommes assurés que le système de réécriture

termine. En fait, c'est un résultat de Hofbauer et Lautemann [HL89], la longueur de dérivation est au plus doublement exponentielle en la taille des entrées (cf également Geupel [Geu88]). C'est ici qu'intervient le travail de Cichon et Lescanne ; ils se sont arrêtés sur la conjecture de Huet et Oppen [HO80] « *Of course, polynomial interpretations do not suffice in general since they give a polynomial upper bound on the complexity of the computations by* $\mathcal{R}$[un système de réécriture], *interpreted as a program computing over integers,* [...] » qui pourrait paraître grossièrement fausse eu égard au travail de Hofbauer, Lautemann et Geupel. Mais en fait, les résultats de Hofbauer, Lautemann et Geupel ne répondent pas à la question (la bonne?) : « Quelles sont les fonctions véritablement calculées par de tels programmes? ». Cichon et Lescanne montrent que de telles fonctions sont bornées polynomialement (et même linéairement dans certains cas), ce qui donne une toute autre perspective à la conjecture de Huet et Oppen.

Le deuxième ordre que nous considérons est l'ordre de Knuth-Bendix introduit à la fin des années 1960 [KB70]. D'après Hofbauer [Hof92a], la longueur de dérivation de tels systèmes est multiplement récursive (de rang 4 dans la hiérarchie de Péter [Pét67]), en d'autres termes, d'une complexité qui est trop élevée si l'on veut calculer avec (ou assurer de tels calculs) de manière efficace et/ou réaliste. Mais des restrictions naturelles de cet ordre donnent lieu à des bornes simplement exponentielles [HL89] ; c'est à ces restrictions que nous nous sommes intéressés.

Constructions d'ordres, vers un calcul monotone

Pour montrer la terminaison des algorithmes, nous nous servons donc d'ordres bien fondés. Ces ordres peuvent être structurés suivant une hiérarchie (leur type d'ordre) qui permet de les comparer. De manière parallèle, la théorie des hiérarchies de fonctions permet de comparer les fonctions du point de vue de leur complexité. L'idée est d'utiliser une hiérarchie de fonctions qui sert d'échelle de référence. Chaque fonction de cette hiérarchie est alors repérée par un indice ordinal qui la classe dans la hiérarchie. Cette dernière est déterminée par un cas de base, une fonctionnelle qui correspond au successeur ordinal, l'opération limite étant prise en compte par la diagonalisation.

Cette théorie trouve sa source dans les travaux de Grzegorczyk [Grz53] qui définissent une hiérarchie de fonctions avec des indices entiers. Cette hiérarchie recouvre l'ensemble des fonctions récursives primitives. L'extension aux ordinaux est due a Robbin [Rob] et Wainer. Les hiérarchies ont ensuite été développées par Löb et Wainer [LW70a, LW70b], Girard [Gir81], Cichon [CW83]. Ces travaux sont fortement liés à la notion de preuve de terminaison en réécriture ainsi que l'a montré Touzet dans [Tou97]. Touzet construit notamment des ordinaux pour montrer la terminaison de systèmes qui terminent pour l'ordre de Knuth-Bendix, l'ordre MPO et l'ordre LPO.

La question que nous nous sommes posée, et qui est au cœur des travaux que nous présentons, concerne l'étude de la *monotonie* dans les structures d'ordres comme les termes ordinaux. Cette motivation est venue de l'observation suivante : l'idée de Löb et Wainer d'étendre la hiérarchie de Grzegorczyk aux indices ordinaux repose sur l'opération de diagonalisation d'une famille de fonctions $(f_n)_{n\in\mathbb{N}}$. Sous certaines hypothèses de monotonie (en particulier lorsque la famille est une ω-chaîne pour l'ordre « pointwise »), la fonction diagonalisée $(\Delta f)(n) = f_n(n)$ a une complexité plus grande que chacune des fonctions de la famille. L'intérêt est que cette construction est similaire à la construction de suites fondamentales (suites croissantes d'ordinaux) pour les ordinaux.

En ce sens, ce ne sont pas les ordinaux qui sont le bon objet d'étude mais les ordinaux équipés de suites fondamentales. Les travaux de Cichon et Touzet [CT96] dans ce domaine ont conduit à la proposition d'un système de termes ordinaux. Vis-à-vis des ordinaux, de tels systèmes offrent davantage de perspectives de par leur plus grande souplesse d'utilisation et de par des distinctions plus fines qu'ils permettent de faire entre deux processus. Plus précisément, le terme ordinal $1 + \omega$ correspond à une opération de diagonalisation suivie de l'utilisation d'une fonctionnelle, il est différent du terme ordinal ω qui ne correspond qu'à l'opération de diagonalisation ; dans le cadre de l'arithmétique ordinale, cette distinction ne tient plus du fait de l'égalité : $1 + \omega = \omega$.

Nous avons repris l'étude du parallèle entre les ordinaux et les hiérarchies de fonctions sous-récursives, mais du point de vue de la monotonie. Un des premiers objectifs que nous nous sommes fixé a été

de dégager une structure qui chapeaute les deux concepts (ordinaux et hiérarchies de fonctions sous-récursives) et de décrire la structure universelle correspondante.

En ce qui concerne la syntaxe, l'idée naïve de construire un calcul à la Simmons dans le cas où l'on tient compte de la monotonie échoue rapidement. De tels systèmes s'avèrent très faibles, et de nouveaux principes sont nécessaires. Cela nous a amené à une deuxième partie de notre étude. Nous nous posons maintenant le problème de la construction d'une syntaxe dans ce cadre. Nous nous plaçons pour cela au confluent de trois théories, la théorie des types, la théorie des catégories et la théorie de l'induction.

Le lien entre les algèbres universelles et les principes d'induction est connu depuis longtemps, au moins depuis Lawvere qui a proposé une axiomatisation des entiers naturels dans la théorie des catégories. En fait, de nombreuses structures mathématiques qui présentent un principe d'induction ressemblent fortement à une théorie algébrique absolument libres[2], théorie donnée par un certain nombre d'opérateurs sans équations supplémentaires. Ainsi, dans le cadre de la théorie des types de Martin-Löf, des notions très générales d'algèbres universelles absolument libres ont été proposées, citons pour l'exemple les articles de Dybjer [Dyb91, Dyb97, DS99, PS89]. Les travaux de Coquand et Paulin dans ce domaine ont donné lieu à des schémas similaires bien que dans un cadre différent, celui du Calcul des Constructions [CP89, PM93].

Le calcul de Simmons (cf [Sim00]) appelé **λH** nous donne un exemple de construction de schémas d'induction dans un calcul typé plus élémentaire en un certain sens que l'approche à la manière de Dybjer (qui met en jeu une notion d'univers par exemple) ou celle de Coquand-Paulin (qui met en jeu les aspects imprédicatifs du Calcul des Constructions). Il introduit deux types de base, un type pour les entiers naturels et un type pour les termes ordinaux. À la manière du système T de Gödel, les entiers naturels sont représentés à l'aide des deux constantes $\overline{0} : \mathbb{N}$ et $\overline{s} : \mathbb{N} \to \mathbb{N}$, et pour tout type X d'une constante $I_X : \mathbb{N} \to X \to (X \to X) \to X$, le principe d'induction des entiers naturels dont « le sens » est : étant donné un « élément »

2. De « absolutely free algebric theories ».

$a \in X$, une « fonction » $t : X \to X$, et un « entier » n, le terme $I_X nat$ représente la composée n fois de la fonction t sur a ; informellement, $I_X nat = \underbrace{f \circ f \circ \cdots f}_{n \text{ fois } f}(a)$[3]. Le principe d'induction est donné via des règles de réduction :

$$\begin{aligned} I_X \overline{0} at &\rhd a \\ I_X (\overline{s} n) at &\rhd t(I_X nat) \end{aligned}$$

Pour les termes ordinaux, Simmons utilise, en plus de la constante $\underline{0} : \mathcal{O}$ pour le terme ordinal de base et de la constante $\underline{s} : \mathcal{O} \to \mathcal{O}$ pour le successeur, une constante $\underline{\mathtt{Lim}} : (\mathbb{N} \to \mathcal{O}) \to \mathcal{O}$ qui sert pour le cas limite. Cette fois-ci, le schéma d'induction est légèrement compliqué du fait de l'opérateur limite : pour tout type X, Simmons introduit une constante $J_X : \mathcal{O} \to X \to (X \to X) \to ((\mathbb{N} \to X) \to X) \to X$ avec les trois règles de réduction correspondantes :

$$\begin{aligned} J_X \underline{0} atu &\rhd a \\ J_X (\underline{s} n) atu &\rhd t(J_X natu) \\ J_X (\underline{\mathtt{Lim}} v) atu &\rhd u(\lambda n : \mathbb{N}. J_X (vn) atu) \end{aligned}$$

Mais dans ce calcul, on ne tient pas compte de la notion d'ordre sur les termes ordinaux *à l'intérieur du calcul.* Simmons a besoin de l'ordre sur les termes ordinaux lorsqu'il veut comparer deux termes, mais il utilise alors la sémantique associée aux termes ordinaux pour les comparer, l'opération a lieu en dehors du calcul. Notre travail a consisté à développer la notion d'ordre à l'intérieur du système.

L'autre calcul que nous connaissons qui traite des termes ordinaux est celui de Danner [Dan99], mais son point de vue est différent du nôtre. Son objectif est essentiellement d'analyser la « complexité » des notations ordinales, la complexité correspond à la complexité du processus qui construit la notation ordinale (de notre point de vue,

3. Simmons utilise un itérateur (d'où la lettre I !) plutôt que le traditionnel récurseur ; cela, explique-t-il, simplifie l'approche catégorique.

c'est surtout la complexité des fonctions sous-récursives qui nous intéresse). En particulier, Danner étudie la complexité des opérations arithmétiques sur les termes ordinaux.

De manière plus générale, la notion d'ordre est d'une grande importance en informatique, elle sert notamment pour *comparer* des objets, nous pensons par exemple à la notion de ω-chaîne qui est extrêmement répandue. Il nous semble par conséquent très intéressant de développer des outils qui permettent de manipuler les structures ordonnées.

Plan et contributions de la thèse

Nous reprenons ici brièvement le plan de la thèse. Nous présentons les résultats les plus importants qui seront exposés.

Chapitre 1. Ce chapitre d'introduction est consacré à la présentation de quelques éléments dont nous avons besoin pour le reste du texte, en particulier, les ordres, les termes et quatre modèles de calcul : la réécriture, les machines de Turing, le λ-calcul (même si nous ne le considérons pas comme un modèle de calcul en tant que tel), et la notion de fonction récursive à manière de Herbrand-Gödel-Kleene. Pour éviter un exposé trop long, nous n'avons pas cherché à faire un exposé très détaillé de chacune de ces notions. Nous convions le lecteur qui ne connaîtrait pas les rudiments que nous venons de mentionner à se reporter aux ouvrages classiques (nous donnons quelques références).

Chapitre 2. Dans ce chapitre, nous abordons plus précisément les notions de théorie des catégories que nous avons utilisées. Nous avons cherché à présenter uniquement les notions indispensables pour la lecture du texte, sans pour autant oublier de donner des exemples qui, nous l'espérons, éclairerons le lecteur. L'objectif principal de ce chapitre est de donner les éléments nécessaires à deux fins : premièrement, dans le cadre du λ-calcul, la théorie des catégories chapeaute d'un point de vue sémantique de nombreuses constructions syntaxiques de manière élégante et simple. Deuxièmement, nous nous servons de la théorie dans le cadre de l'algèbre universelle. Ce deuxième point sera également développé dans l'introduction du chapitre 6.

Chapitre 3. Nous exposons dans ce chapitre les travaux que nous

avons accomplis concernant l'analyse en termes de complexité des preuves de terminaison par interprétation polynomiale dans le cadre de la réécriture. Cela nous permet de donner des caractérisations des fonctions calculées par les systèmes de réécriture qui admettent de telles preuves de terminaison. Dans cette optique, nous avons démontré l'équivalence entre les classes suivantes (les deux premiers tableaux) :

Classes de complexité	Systèmes de réécriture
PTIME	$\Pi(0)$
ETIME	$\Pi(1)$
E$_2$TIME	$\Pi(2)$
TIME$(\exp_\infty(n+k))$	$\Pi(3)$

Classes de complexité	Systèmes de réécriture
NPTIME	$\Delta(0)$
NETIME	$\Delta(1)$
NE$_2$TIME	$\Delta(2)$
NTIME$(\exp_\infty(n+k))$	$\Delta(3)$

Classes de complexité	Systèmes de réécriture
PSPACE	$\Theta(0)$, $\Gamma(0)$
ESPACE	$\Theta(1)$, $\Gamma(1)$
E$_2$SPACE	$\Theta(2)$, $\Gamma(2)$
SPACE$(\exp_\infty(n+k))$	$\Theta(3)$, $\Gamma(3)$

où les critères $\Pi(i)$ et $\Delta(i)$ correspondent à la forme des interprétations des constructeurs de l'algèbre sur laquelle on travaille. Nous

avons également obtenu une caractérisation parallèle à celles qui précèdent mais pour des classes de complexité en espace. Pour cela, nous avons dégagé la notion de pseudo-preuve de terminaison par interprétation polynomiale. Ce sont les critères $\Theta(i)$ et $\Gamma(i)$ qui correspondent aux critères précédents, mais dans le cadre des pseudo-preuves de terminaison. En outre, nous obtenons une caractérisation de LINSPACE en termes de pseudo-preuve de terminaison par interprétation polynomiale.

Chapitre 4. Nous faisons une étude analogue à celle du chapitre 3, mais pour les preuves de terminaison par l'ordre de Knuth-Bendix. Nous donnons deux caractérisations en termes de complexité : les systèmes de réécriture qui admettent une preuve de terminaison par l'ordre de Knuth-Bendix avec un poids strictement positif pour chaque symbole correspondent exactement aux fonctions de LINSPACE. D'autre part, nous définissons la notion de systèmes de mots enrichis et nous montrons que cela donne une caractérisation de ESPACE.

Chapitre 5. Dans ce chapitre, nous faisons une première approche de la notion de monotonie dans le cadre des termes ordinaux. Nous analysons la notion de progression, c'est-à-dire des structures qui admettent une constante, une opération monotone et inflationnaire et une opération limite. La notion de progressions chapeaute en particulier trois structures qui nous intéressent, les ordinaux, les termes ordinaux et les hiérarchies de fonctions sous-récursives.

Nous classifions les progressions selon deux critères : l'inflationnarité ($x \leq sx$) et la stationnarité ($\mathtt{Lim}(0)_{n\in\mathbb{N}} = 0$ où 0 est la constante de base). Nous donnons pour chacune des sous-structures ainsi définies un théorème de représentation de la structure libre à l'aide d'un unique concept, les *arborescences*.

Chapitre 6. Après les travaux présentés au chapitre 5, nous avons cherché à construire une syntaxe des termes ordinaux ordonnés. Pour cela, nous avons dû résoudre dans un premier temps quelques problèmes de sémantique catégorique sur les graphes. Dans ce chapitre, nous dégageons une nouvelle[4] structure monoïdale sur les graphes,

4. Nous avons interrogé les catégoriciens à ce sujet via la mailing-list, et nous

le bon univers de travail, comme nous le justifions dans l'introduction du chapitre. D'autre part, nous définissons une notion d'algèbre universelle sur les graphes, les Ψ-algèbres, et nous montrons que sous certaines conditions, de telles algèbres sont analogues à des ordres. Nous donnons alors quelques exemples d'utilisation des Ψ-théories ; en particulier, nous montrons qu'elles recouvrent des ordres comme l'ordre sur les termes ordinaux, ou bien des ordres (d'une grande importance en réécriture) comme l'ordre de Kruskal.

Chapitre 7. Ce chapitre est d'une certaine manière l'aboutissement des deux chapitres précédents. Nous y définissons une syntaxe pour les termes ordinaux ordonnés, mais plus généralement, la syntaxe proposée vise à manipuler/définir des graphes, leurs sommets, leurs arêtes, les morphismes de graphes. Nous mettons en jeu les notions sémantiques vues au chapitre précédent pour définir un λ-calcul typé (avec des types dépendants) qui permet de construire des ordres inductifs et des fonctions monotones (en fait, des morphismes de graphes). Nous donnons comme exemple « un système T » dans lequel on peut montrer la monotonie des fonctions. Nous construisons également une extension du système $\boldsymbol{\lambda H}$ de Simmons aux termes ordinaux ordonnés.

n'avons pas eu de réponse qui prouverait le contraire.

Chapitre 1

Notions préliminaires

Les travaux que nous présentons dans cette thèse requièrent quelques notions préliminaires. Nous les décrivons dans ce chapitre. Nous commençons par quelques rappels touchant les ordres et les termes ; ceci nous donnera l'occasion de fixer les notations et la terminologie. Nous présentons ensuite les modèles de calcul que nous utilisons.

Nous n'avons pas cherché à formaliser les fondements de notre cadre mathématique. Nous utilisons de manière naïve les termes d'« ensembles », d'« égalité », etc. Notons toutefois que nous considérons des collections d'objets telle la collection de tous les ensembles, de ce fait, nous avons besoin d'une théorie un peu plus forte que celle de Zermelo-Fraenkel ; de ce point de vue, la théorie des ensembles de Von Neumann-Bernays-Gödel convient (pour une présentation de cette théorie, nous renvoyons le lecteur à Mendelson [Men64]).

1.1 Ordres, ordinaux

Étant donné un ensemble E, une relation binaire $\mathcal{R}$ sur E est une partie de $E \times E$, où $\times$ est le produit cartésien. Lorsque nous parlons d'une relation binaire en général, nous entendons la paire $(E, \mathcal{R})$. On note habituellement les éléments d'une relation binaire : $\mathcal{R}(x,y)$ ou encore $x \mathcal{R} y$.

Définition 1.1.1. Étant donné une relation binaire $\mathcal{R}$ sur E, on dit de la relation qu'elle est :

- *réflexive* si $\forall x \in E$, $\mathcal{R}(x,x)$,
- *anti-réflexive* si $\forall x \in E$, $\neg\mathcal{R}(x,x)$,
- *symétrique* si $\forall (x,y) \in E \times E$, $\mathcal{R}(x,y) \iff \mathcal{R}(y,x)$,
- *anti-symétrique* si $\forall (x,y) \in E \times E$, $\mathcal{R}(x,y) \wedge \mathcal{R}(y,x) \Rightarrow x = y$,
- *transitive* si $\forall (x,y,z) \in E \times E \times E$, $(\mathcal{R}(x,y) \wedge \mathcal{R}(y,z)) \Rightarrow \mathcal{R}(x,z)$,

Définition 1.1.2 (Relation d'équivalence, Ordres). Nous distinguons parmi les relations binaires $\mathcal{R}$ sur E, les *relations d'équivalence*, *les pré-ordres*, les *ordres partiels*, les *ordres stricts* selon qu'elles vérifient où non les propriétés précédentes :

	Réfl.	Anti-réfl.	Sym.	Anti-sym.	Trans.
rel. d'équivalence	×		×		×
pré-ordre	×				×
ordre partiel	×			×	×
ordre strict		×			×

Le symbole × indique que la relation vérifie la propriété en question.

Définition 1.1.3. Un *ordre total* est un ordre strict $(E, \mathcal{R})$ tel que pour tout $x,y \in E$, l'un des trois cas suivants est réalisé :

- $x \mathcal{R} y$,
- $y \mathcal{R} x$,
- $x = y$.

Par la suite, nous notons les relations d'équivalence à l'aide des symboles $=$, $\simeq$, $\equiv$, ..., les pré-ordres et les ordres partiels à l'aide de $\leq$, $\geq$, $\preceq$, $\succeq$, $\subseteq$, ..., pour les ordres stricts et les ordres totaux, nous utilisons les symboles $<$, $>$, $\prec$, $\succ$, ... Lorsque nous parlons d'ordre sans plus de précisions, nous désignons une des relations d'ordre qui précèdent ; le contexte (et la notation) lève alors toute ambiguïté.

Définition 1.1.4 (Relation inverse). Étant donné une relation binaire $\mathcal{R}$ sur un ensemble E, on considère la relation inverse donnée par : $x \mathcal{R}^{-1} y$ ssi $y \mathcal{R} x$.

Définition 1.1.5 (Relations induites par un pré-ordre). Étant donné un pré-ordre $\preceq$ sur un ensemble E, on considère sur E les relations suivantes :

- la relation d'équivalence donnée par : $x \simeq y$ ssi $(x \preceq y) \wedge (y \preceq x)$,
- l'ordre strict donné par $x \prec y$ ssi $(x \preceq y) \wedge \neg(y \preceq x)$,

On notera que la relation inverse du pré-ordre est également un pré-ordre.

L'ordre $\prec$ induit un ordre partiel sur l'ensemble $E/\simeq$: $\overline{a} \prec \overline{b}$ ssi $a \prec b$.

Définition 1.1.6. Étant donné un ordre $(X, <)$, un élément a de X est dit

- *maximal* (respectivement *minimal*) s'il n'y a aucun $x \in X$ tel que $a < x$ (resp. $x < a$),
- le *plus grand élément, le maximum* (respectivement *plus petit élément, le minimum*) de X si pour tout $x \in X$, $x \leq a$ (resp. $a \leq x$),
- un *majorant* (resp. un *minorant*) d'une partie $U \subset X$ si pour tout $x \in U$, $x \leq a$ (resp. $a \leq x$). Le majorant (resp. le minorant) est dit strict si $\forall x, x < a$ (resp. $a < x$).

Définition 1.1.7 (Fonction monotone). Étant donné une relation binaire $\mathcal{R}_A$ sur un ensemble A et une relation binaire $\mathcal{R}_B$ sur B, une fonction $f : A \to B$ est monotone si pour tout x $\mathcal{R}_A$ y, on a $f(x)$ $\mathcal{R}_B$ $f(y)$. On dit aussi que la fonction respecte les relations binaires.

Un *isomorphisme* entre deux relations binaires est une fonction monotone bijective dont l'inverse est également monotone.

Définition 1.1.8 (Dérivation). Étant donné une relation $\mathcal{R}$ sur un ensemble E, une dérivation est une suite $(x_i)_{i \geq 0}$ d'éléments de E telle que :

$$\begin{array}{lll} x_0 \mathcal{R} & x_1 & \\ & x_1 \mathcal{R} & x_2 \\ & & x_2 \mathcal{R} \cdots \\ & & \cdots \end{array}$$

On note habituellement une telle dérivation : $x_0 \mathcal{R} x_1 \mathcal{R} x_2 \cdots$.

Définition 1.1.9 (Fermetures d'une relation). Étant donné une relation $\mathcal{R}$ sur un ensemble E,

- la fermeture réflexive de la relation $\mathcal{R}$ est la relation $\underline{\mathcal{R}}$ définie par $x \underline{\mathcal{R}} y$ ssi $x = y$ ou $x \mathcal{R} y$;
- la fermeture symétrique de la relation $\mathcal{R}$ est la relation $\mathfrak{S}\mathcal{R}$ définie par $x \mathfrak{S}\mathcal{R} y$ ssi $x \mathcal{R} y$ ou $y \mathcal{R} x$;
- la fermeture transitive de la relation $\mathcal{R}$ est la plus petite relation (notée $\mathcal{R}^*$) qui vérifie :

$$\frac{x \mathcal{R} y}{x \mathcal{R}^* y} \qquad \frac{x \mathcal{R} y \qquad y \mathcal{R}^* z}{x \mathcal{R}^* z}$$

En d'autres termes, $x \mathcal{R}^* y$ ssi il existe une dérivation $x \mathcal{R} x_1 \mathcal{R} x_2 \cdots \mathcal{R} y$.

Définition 1.1.10 (Relation nœtherienne, bien fondée). Une relation $\mathcal{R}$ sur un ensemble E est nœtherienne si toute dérivation est finie. Une relation est *bien fondée* si sa relation inverse est nœtherienne.

Si une relation est réflexive, elle ne peut pas être à strictement parler nœtherienne, toutefois, on la considère comme telle si sa restriction stricte l'est.

Les ordinaux

Définition 1.1.11 (Bon ordre). Un ordre total $\prec$ sur un ensemble X est appelé un bon ordre si toute partie non vide de X admet un plus petit élément.

Propriété 1.1.12. Nous avons une autre caractérisation des bons ordres : une relation est un bon ordre ssi elle est un ordre total bien fondé.

Définition 1.1.13. Un *segment initial* d'un ordre $(X, \prec_x)$ est une partie U de X telle que $\forall x \in U$, $\forall y \prec_x x, y \in U$. En particulier, X est un segment initial de X. Un segment initial propre de X est un segment initial différent de X.

Propriété 1.1.14. Nous énumérons ici quelques propriétés des bons ordres. Étant donné un bon ordre $(X, \prec)$, les propriétés suivantes s'obtiennent en cascade :

1. si $f : X \to X$ est monotone, alors pour tout $x \in X$, $x \preceq f(x)$,
2. le seul isomorphisme d'un bon ordre sur lui-même est l'identité,
3. si deux bons ordres sont isomorphes, l'isomorphisme entre eux est unique,
4. un bon ordre n'a pas de segment initial propre isomorphe à lui-même.

Théorème 1.1.14.1.
Étant donné deux bons ordres $(X, \prec_x)$ et $(Y, \prec_y)$, l'un des trois cas (mutuellement exclusifs) suivants est valide :

(a) *X est isomorphe à Y,*

(b) *X est isomorphe à un segment initial propre de Y,*

(c) *Y est isomorphe à un segment initial propre de X.*

On en déduit une relation de pré-ordre sur la classe des ordres : $(X, \prec_x) \leq (Y, \prec_y)$ si et seulement si $(X, \prec_x)$ est un segment initial de $(Y, \prec_y)$. Une classe d'équivalence pour la relation d'équivalence induite (cf définition 1.1.5) s'appelle un type d'ordre.

Le pré-ordre induit également un ordre noté $<$ sur les types d'ordre (cf définition 1.1.5).

Théorème 1.1.14.2.
L'ordre $<$ sur la classe des types d'ordre est un bon ordre.

Nous donnons maintenant une représentation canonique en termes ensemblistes des types d'ordre, ce sont les *ordinaux*. L'idée est de définir un ordinal en sorte que :

$$\alpha < \beta \text{ ssi } \alpha \in \beta, \qquad \alpha = \{\beta \mid \beta < \alpha\}$$

Définition 1.1.15. Un ensemble X est transitif si $\forall x \in X, x \subseteq X$.

Définition 1.1.16 (Ordinal). Un ordinal est un ensemble transitif bien ordonné par la relation $\in$. On note les ordinaux à l'aide des lettres grecques α, β, ...

`Ord` désigne la classe de tous les ordinaux. Enfin, sur `Ord`, on définit $\alpha < \beta$ ssi $\alpha \in \beta$.

Propriété 1.1.17.

(i) L'ordre $<$ sur `Ord` est total,

(ii) Pour tout α, $\alpha = \{\beta \mid \beta < \alpha\}$,

(iii) $0 = \emptyset$ est un ordinal,

(iv) Pour tout ordinal α, $\{\alpha \cup \{\alpha\}\}$ est un ordinal. On note ce dernier $\alpha + 1$ et on l'appelle le successeur de α.

(v) Si un ordinal n'est pas un successeur, on le qualifie d'ordinal limite. 0 est un ordinal limite. ω désigne le plus petit ordinal limite différent de 0. Les ordinaux plus petits que ω sont les ordinaux finis (les entiers naturels) :

$$0 = \emptyset, \quad 1 = 0 + 1, \quad 2 = 1 + 1$$

Théorème 1.1.17.1.
Un bon ordre est isomorphe à un et un seul ordinal.

Définition 1.1.18. On note Ω le plus petit ordinal non dénombrable[5]. Cet ordinal correspond au bon ordre suivant. Soit $(O, \leq)$ le plus petit ensemble ordonné vérifiant les règles :

$$\frac{}{0 \in O} \qquad \frac{\alpha \in O}{s\alpha \in O} \qquad \frac{\forall x \in \omega, \alpha_x \in O}{(\alpha_x)_{x \in \omega} \in O}$$

$$\frac{}{0 \leq 0} \qquad \frac{\alpha \in O}{\alpha < s\alpha} \qquad \frac{\forall x \in \omega, \alpha_x \in O}{\alpha_k < (\alpha_x)_{x \in \omega} \in O} k \in \omega$$

Ordre sur les multi-ensembles

5. Celui-ci existe, d'après le théorème du bon ordre de Zermelo.

Définition 1.1.19. Étant donné un ensemble E, on appelle une fonction $E \to \mathbb{N}$ un *multi-ensemble* sur E. On note $\texttt{MultiEns}(E)$ l'ensemble des multi-ensembles sur E. Soient $e \in E$ et m un multi-ensemble sur E. $m(e)$ s'appelle la *multiplicité* de e dans m. L'ensemble des éléments de multiplicité strictement positive d'un multi-ensemble m s'appelle le *support* du multi-ensemble.

Soit $e \in E$, $\{e\}$ désigne le multi-ensemble suivant :

$$\begin{cases} e \mapsto 1 \\ x \mapsto 0 \quad \text{si } x \neq e \end{cases}$$

D'autre part, si m et n sont des multi-ensembles sur E, $m+n$ est défini par :

$$\forall e \in E, \quad (m+n)(e) = m(e) + n(e)$$

Il en est de même pour les unions finies,

$$\forall e \in E, \quad (\sum_{i=1}^{n} m_i)(e) = \sum_{i=1}^{n} m_i(e)$$

Définition 1.1.20. Un ordre $>$ sur E induit un ordre sur $\texttt{MultiEns}(E)$ défini comme suit :

$$\begin{array}{rl} X > & \emptyset \\ \{a\} + X > & \{b\} + Y \quad \text{si } \{a\} + X > Y \text{ et } a > b \end{array}$$

1.2 Algèbres de termes et de mots

Définition 1.2.1. Une *signature* Σ est une famille finie d'ensembles finis $(\Sigma_i)_{i \in I}$ $(I \subset \mathbb{N})$ de *symboles/lettres*. À moins que nous ne fassions mention du contraire, nous supposerons que les ensembles Σ_i sont disjoints deux à deux. L'indice de l'ensemble auquel appartient le symbole s'appelle l'*arité* du symbole, un symbole dans Σ_n est dit *n-aire* ; on dit plus particulièrement pour les symboles de Σ_1 qu'ils sont unaires et pour les symboles de Σ_2 qu'ils sont binaires. Nous utiliserons le terme de constante pour désigner les symboles de Σ_0. Enfin, nous noterons $\mathbf{ar}(f)$ (ou encore n_f) l'arité du symbole f.

Définition 1.2.2. Supposons que l'on dispose d'un ensemble $\mathcal{X}$ (éventuellement vide) d'éléments appelés *variables*, disjoint de la signature Σ. Nous définissons l'*algèbre de termes* $\mathcal{T}(\Sigma,\mathcal{X})$ sur Σ et $\mathcal{X}$ comme étant le plus petit ensemble vérifiant les règles :

$$\frac{x \in \mathcal{X}}{x \in \mathcal{T}(\Sigma,\mathcal{X})}(\text{i}) \qquad \frac{\forall i \leq n, \quad t_i \in \mathcal{T}(\Sigma,\mathcal{X}) \qquad f \in \Sigma_n}{f(t_1,t_2,\cdots,t_n) \in \mathcal{T}(\Sigma,\mathcal{X})}(\text{ii})$$

où dans la règle (ii), n est éventuellement nul. Un terme est dit *clos* s'il ne contient pas de variables, c'est-à-dire, si sa définition n'utilise pas la règle (i). L'ensemble des termes clos est noté $\mathcal{T}(\Sigma)$.

Remarque 1.2.3. Observons que $\mathcal{T}(\Sigma)$ est vide si et seulement si Σ_0 est vide. En conséquence de quoi, nous ferons toujours l'hypothèse que $\Sigma_0 \neq \emptyset$.

Définition 1.2.4. Si la signature Σ ne contient que des constantes et des symboles unaires, l'algèbre de terme $\mathcal{T}(\Sigma,\mathcal{X})$ est dite une *algèbre de mots*. En ce cas, on désigne par $w[k]$ le symbole a_{i_k} du mot $w = a_{i_1}(a_{i_2}(\ldots a_{i_k}(\cdots)\cdots))$.

Exemple 1.2.5. Nous utiliserons souvent les entiers codés en notation unaire (*les entiers bâtons*), $\mathtt{Nat}$. $\mathtt{Nat}_0 = \{0\}$, $\mathtt{Nat}_1 = \{s\}$. Tout entier naturel n se représente dans $\mathcal{T}(\mathtt{Nat})$ par :

$$\underline{n} = \underbrace{s(s(\cdots(s}_{n \text{ fois } s}(0))\cdots))$$

Par la suite, afin d'alléger la notation, nous supprimerons les parenthèses pour les symboles unaires.

Exemple 1.2.6. Étant donné un ensemble E, l'algèbre E^* des mots sur E est l'algèbre de mots provenant de la signature Σ définie ainsi : $\Sigma_0 = \{\epsilon\}$ et $\Sigma_1 = E$. Nous écrirons généralement $a.w$ (voire aw) le terme $a(w)$ où $a \in E$ et $w \in E^*$. Le terme ϵ s'appelle le mot vide.

Par extension, si W désigne une partie de E^*, pour tout symbole $a \in E$, nous notons $a.W$ l'ensemble $\{aw \mid w \in W\}$.

Définition 1.2.7 (Taille et hauteur d'un terme). On définit la *taille* $|t|$ et la *hauteur* $h(t)$ d'un terme $t \in \mathcal{T}(\Sigma,\mathcal{X})$ inductivement :

$$\begin{cases} |x| &= 1 \\ |f(t_1,\ldots,t_n)| &= 1+\sum_{i=1}^{n}|t_i| \end{cases} \qquad \begin{cases} h(x) &= 1 \\ h(f(t_1,\ldots,t_n)) &= 1+\max_{i=1}^{n} h(t_i) \end{cases}$$

Note 1.2.1. Nous utilisons une notation pour désigner la taille (la hauteur) d'un n-uplet de termes : $|t_1,\ldots,t_n| = \sum_{i=1}^{n}|t_i|$ et $h(t_1,\ldots,t_n) = \max_{i=1}^{n} h(t_i)$.

1.3 Modèles de calculs

De nombreuses études ont été consacrées à la recherche de caractérisations mathématiques de la calculabilité, c'est-à-dire des *procédés mécaniques/algorithmes* de calcul. Parmi les modèles que nous utilisons, trois d'entre-eux revètent une grande importance épistémologique : les fonctions récursives générales (les équations de Herbrand-Gödel), le λ-calcul et les machines de Turing. Ils recouvrent en effet la même notion de calculabilité. Nous faisons une parenthèse historique retraçant le développement de ces modèles de calcul (inspirée de [Kle52, Dav58, Dav82, Ros84]).

En 1931, Gödel introduit de manière explicite la classe des fonctions récursives primitives[6] ; puis, sur une suggestion de Herbrand, définit une notion de fonctions récursives générales[7] (nos fonctions récursives). Dans ce contexte, Kleene étend la récursion primitive au moyen d'un opérateur de minimisation non bornée en 1936 [Kle36], et il montre que la classe des fonctions ainsi définies est identique à la classe des fonctions récursives générales de Gödel.

De manière parallèle, Church, aidé par ses étudiants (dont Kleene et Rosser) a développé un système dont l'objectif initial était de construire les fondements des mathématiques à partir de la notion

6. D'autres auteurs ont étudié les définitions récursives, parmi lesquels, on peut citer Dedekind (1888), Peano (1888) ou Skolem (1923).

7. Selon Davis, quoiqu'il soit conscient de la question, Gödel n'est pas convaincu que cette notion capture l'ensemble des fonctions calculables.

de fonction. Cette approche est montrée inconsistante par Kleene et Rosser. Toutefois, il en est resté un sous-système, le λ-calcul qui permet de définir une notion de calculabilité montrée équivalente à celle de fonction récursive générale de Gödel [CR36].

Turing s'intéresse à la nature du calcul d'une manière différente, il cherche à décrire ce que fait une personne quand elle calcule, il supprime de cette analyse les détails inutiles et en tire le modèle bien connu avec un ruban infini et une tête de lecture/écriture[8] [Tur36]. Ce modèle de calcul s'avère équivalent à celui de Church.

Le fait que des approches si différentes aboutissent à la même notion de calculabilité donne de sérieux atouts au postulat, « la thèse de Church », selon lequel ces modèles recouvrent correctement la notion de calculabilité.

Nous utilisons ces modèles pour l'aspect particulier qu'ils donnent à la notion de calcul. Ainsi, nous considérons les machines de Turing dans le cadre des classes de complexité, le λ-calcul comme une syntaxe pour la sémantique des théories algébriques que nous introduisons au chapitre 6. Enfin, nous utilisons la notion de fonction récursive au travers des hiérarchies de fonctions sous-récursives.

Dans ce chapitre, nous présentons plus en détail la théorie des fonctions récursives, le λ-calcul, les machines de Turing ainsi que la réécriture, modèle de calcul dont l'histoire est plus récente, mais dont l'avenir nous semble prometteur.

1.4 Systèmes de réécriture

Les systèmes de réécriture sont des ensembles d'équations orientées. Nous avons utilisé plusieurs références, principalement l'article de Dershowitz et Jouannaud [DJ90], mais aussi celui de Jouannaud et Lescanne [JL86] et celui de Huet et Oppen [HO80].

Nous supposons dans cette section que nous disposons d'une signature Σ et d'un ensemble de variables $\mathcal{X}$ à l'aide desquels nous définissons l'algèbre de termes $\mathcal{T}(\Sigma,\mathcal{X})$.

8. Indépendamment du travail de Turing (mais pas des travaux autour du λ-calcul), Post a proposé une notion de machine très proche de celle de Turing [Pos36].

Définition 1.4.1 (Contexte). Un contexte relativement à une variable x est un terme dans lequel x a une occurence unique.

Définition 1.4.2 (Substitution). Une *substitution* est une assignation de termes pour certaines variables. Plus formellement, étant donné une signature Σ et un ensemble de variables $\mathcal{X}$, une substitution est une fonction $\sigma : S_\sigma \rightarrow \mathcal{T}(\Sigma,\mathcal{X})$ où $S_\sigma \subset \mathcal{X}$ est un ensemble fini qui s'appelle le *support* de la substitution. Une telle substitution s'étend canoniquement sur les termes par :

$$\begin{array}{rcll} \sigma(y) = & y & & \text{si } y \notin S_\sigma \\ \sigma(f(t_1,t_2,\cdots,t_n)) = & f(\sigma(t_1),\sigma(t_2),\cdots,\sigma(t_n)) & & \text{si } f \in \Sigma_n \end{array}$$

En fait, il est d'usage de noter la substitution après le terme, c'est-à-dire $t\sigma$ plutôt que $\sigma(t)$. Par ailleurs, nous notons $\{x \mapsto t\}$ la substitution qui a pour support le singleton $\{x\}$ et associe à x le terme t.

Une substitution σ est dite *close* si pour toute variable $x \in S_\sigma$, nous avons $x\sigma \in \mathcal{T}(\Sigma)$.

Définition 1.4.3 (Règle et système de réécriture). Étant donné une algèbre de termes $\mathcal{T}(\Sigma,\mathcal{X})$, une *règle de réécriture* est une paire (l,r) de termes telle que $V(r) \subset V(l)$ où $V(t)$ est l'ensemble des variables du terme t. Nous écrivons habituellement de telles paires sous la forme : $l \rightarrow r$. Le terme l s'appelle le *membre gauche* de la règle, et le terme r son *membre droit*. Un *système de réécriture* est une paire $(\mathcal{T}(\Sigma,\mathcal{X}),\mathcal{R})$ composée d'une algèbre de termes $\mathcal{T}(\Sigma,\mathcal{X})$ et d'un ensemble $\mathcal{R}$ de règles de réécriture.

Par la suite, nous supposons que $\mathcal{X}$ est donné une fois pour toutes, et nous abrégeons $(\mathcal{T}(\Sigma,\mathcal{X}),\mathcal{R})$ en $(\mathcal{R},\Sigma)$. Nous discutons maintenant de quelques propriétés de la réécriture et de son aspect calculatoire. Pour simplifier l'exposé, nous supposons pour le reste de la section que nous nous sommes donné un système de réécriture $(\mathcal{R},\Sigma)$ vis-à-vis duquel les définitions que nous faisons sont dépendantes.

Définition 1.4.4 (Relation de réécriture). La *relation de réécriture en un pas* est la relation binaire sur $\mathcal{T}(\Sigma,\mathcal{X})$ définie par : $u \rightarrow_\mathcal{R} v$

ssi il existe un contexte $t \in \mathcal{T}(\Sigma,\mathcal{X})$ vis-à-vis de x, une substitution σ et une règle $l \to r$ tels que $u = t\{x \mapsto (l\sigma)\}$ et $v = t\{x \mapsto (r\sigma)\}$. Le sous-terme $l\sigma$ de u s'appelle un *redex*, le terme $r\sigma$ s'appelle le réduit.

Définition 1.4.5. La fermeture réflexive transitive de $\to_{\mathcal{R}}$ se note $\overset{*}{\to}_{\mathcal{R}}$. Autrement dit, $u \overset{*}{\to}_{\mathcal{R}} v$ s'il existe une dérivation finie $t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} \cdots \to_{\mathcal{R}} t_n$ telle que $t_0 = u$ et $t_n = v$. En ce cas, on dit que u se *réécrit* en v. Pour souligner le fait qu'il y a n étapes dans la dérivation, nous utilisons la notation $u \to^n_{\mathcal{R}} v$.

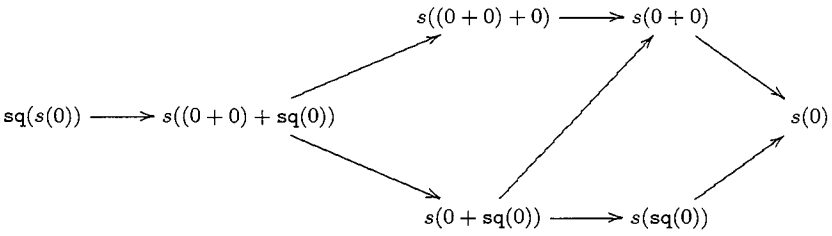
1.4.1 Terminaison et confluence

Définition 1.4.6. Un terme u est dit en *forme normale* s'il n'existe pas de terme $v \in \mathcal{T}(\Sigma,\mathcal{X})$ tel que $u \to_{\mathcal{R}} v$. Nous notons $t \overset{!}{\to}_{\mathcal{R}} u$ le fait que $t \overset{*}{\to}_{\mathcal{R}} u$ et que u soit en forme normale.

Exemple 1.4.7. La réécriture peut être utilisée pour calculer. En général, le calcul se fait par normalisation. Prenons par exemple la signature `Carré` composée de `Nat`, d'un symbole unaire `sq`, et d'un symbole binaire $+$. Nous pouvons calculer le « carré » des entiers en notation unaire à l'aide du système de réécriture :

$$\begin{array}{llll} 0+y & \to y & \mathtt{sq}(0) & \to 0 \\ sx+y & \to s(x+y) & \mathtt{sq}(sx) & \to s((x+x)+\mathtt{sq}(x)) \end{array}$$

Nous représentons ici quelques termes de la relation de réécriture en un pas :



Nous donnons ici quelques illustrations des définitions précédentes :

- $s(0+0) \to_{\mathcal{R}} s(0)$;

- $\mathtt{sq}(s(0)) \xrightarrow{*}_{\mathcal{R}} s(0+0)$;
- $\mathtt{sq}(s(0)) \xrightarrow{!}_{\mathcal{R}} s(0)$;

Remarquons maintenant que pour tout entier n, $\mathtt{sq}(\underline{n}) \xrightarrow{!}_{\mathcal{R}} \underline{n^2}$. En ce sens, nous calculons le carré des entiers bâtons. Nous reviendrons par la suite sur la notion de calcul par des systèmes de réécriture.

Définition 1.4.8 (Terminaison). Un système de réécriture *termine* si la relation $\rightarrow_{\mathcal{R}}$ est nœtherienne, c'est-à-dire, s'il n'existe pas de dérivation infinie. En ce cas, nous disposons de deux notions de longueur de dérivation :

1. la longueur de dérivation *d'un terme* t est la plus longue dérivation à partir de celui-ci :

$$\mathtt{dl}(t) = \max\{n \mid t \rightarrow_{\mathcal{R}}^{n} u\},$$

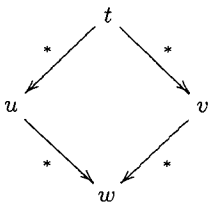
2. la longueur de dérivation *pour le système* est la fonction :

$$\begin{array}{rccl} \mathtt{Dl}: & \mathbb{N} & \rightarrow & \mathbb{N} \\ & n & \mapsto & \max\{\mathtt{dl}(t) \mid |t| \leq n\}. \end{array}$$

Théorème 1.4.8.1 (Huet et Lankford [HL78]).
La propriété de terminaison des systèmes de réécriture est indécidable.

Une des raisons est que l'on peut simuler les calculs d'une machine de Turing avec la réécriture.

Définition 1.4.9 (Confluence). Un système de réécriture est *confluent* si pour tout terme t, u, v tels que $t \xrightarrow{*}_{\mathcal{R}} u$ et $t \xrightarrow{*}_{\mathcal{R}} v$, il existe un terme w tel que $u \xrightarrow{*}_{\mathcal{R}} w$ et $v \xrightarrow{*}_{\mathcal{R}} w$.



Théorème 1.4.9.1 (Huet et Lankford [HL78]).
La propriété de confluence des systèmes de réécriture est indécidable.

Idée de preuve. À tout langage $\mathcal{L}$ reconnu par une machine de Turing, nous pouvons associer un système de réécriture qui répond « vrai » si le mot d'entrée est dans $\mathcal{L}$ et « faux » dans l'autre cas. Nous rajoutons alors une règle qui donne la réponse « vrai » pour chaque entrée. Si le système est confluent, cela signifie que le langage est « plein ». Or cette propriété est indécidable en général. □

Proposition 1.4.10. Si un système de réécriture est confluent et qu'il termine, alors, chaque terme t admet une *unique* forme normale.

L'intérêt de cette proposition est qu'un tel système fournit une procédure de calcul simple : on réécrit le terme donné en entrée jusqu'à ce qu'il ne contienne plus de redex. Dans ce cas, il n'est pas nécessaire de disposer d'une *stratégie* [9] dans le choix des redex à réduire puisque toute dérivation aboutit au même résultat.

1.4.2 Calculs à l'aide de systèmes de réécriture

Comme nous aurons à comparer plusieurs modèles de calculs, nous devons préciser la sémantique des fonctions que l'on calcule. Nous supposons par la suite que les calculs portent sur une algèbre $\mathcal{A}$, dite *algèbre de calcul.* Pour la réécriture, nous passons par deux codages de cette algèbre, le premier vers des symboles d'entrée, le deuxième vers les symboles de résultat. Ce point de vue est issu d'un travail présenté au chapitre 3 et réalisé à partir d'un article de Cichon et Lescanne [CL92].

Dans [CL92], Cichon et Lescanne étudient les fonctions numériques définies par des systèmes de réécriture. Pour cela, ils considèrent les systèmes $(\mathcal{R},\Sigma)$ tels que :

- les entiers sont représentés par l'algèbre $\mathtt{Nat} = \{0,s\}$ contenue dans Σ ;
- le système de réécriture termine, il est confluent et les formes normales sont dans $\mathcal{T}(\mathtt{Nat})$.

9. Nous ne considérons pas ici le problème sous l'angle de la complexité du calcul.

Chaque symbole f d'arité n est alors associé à une fonction ϕ : $\mathcal{T}(\mathtt{Nat})^n \to \mathcal{T}(\mathtt{Nat})$ définie (selon la proposition 1.4.10) par :

$$(t_1, \cdots, t_n) \mapsto \mathrm{nf}(\mathrm{f}(\mathrm{t}_1, \cdots, \mathrm{t}_\mathrm{n}))$$

où nf(t) est la forme normale issue de t.

Les efforts de Cichon et Lescanne portent en particulier sur le problème de savoir quelles sont les restrictions imposées aux fonctions définies par des systèmes de réécriture ayant une preuve par interprétation polynomiale sur les entiers. Leur principal résultat est de montrer que de telles fonctions sont bornées polynomialement quand bien même la longueur de dérivation d'un terme peut être doublement exponentielle (cf. Lautemann et Hofbauer [HL89] et Geupel [Geu88]).

Par rapport au travail de Cichon et Lescanne, nous étendons la notion de fonction « définie par un système de réécriture » selon deux directions. Nous considérons des algèbres de termes quelconques au lieu de `Nat` [BCMT99] et nous faisons une distinction entre l'algèbre des entrées et celle des sorties [BCMT98].

Le fait de considérer des algèbres plus générales que `Nat` a deux avantages. Premièrement, cela nous rapproche de la vraie programmation qui manipule des structures de données comme les entiers binaires, les listes, les arbres, etc. Deuxièmement, du point de vue de la complexité, la notation unaire a des propriétés très particulières. Cela est dû au fait que la taille des termes est exponentielle par rapport à celle de la numération binaire. Ainsi, le problème du voyageur de commerce est polynomial si l'on considère les entrées codées par des entiers bâtons.

A priori, la distinction que nous faisons entre symboles d'entrée et symboles de sortie est inutile ; en effet, on peut sans difficulté rajouter des règles pour transformer les termes d'une algèbre en les termes de l'autre. En revanche, si l'on se donne une méthode de preuve de terminaison, et que l'on considère les systèmes qui terminent pour de telles preuves, on ne peut plus rajouter de règles sans menacer de ruiner la preuve de terminaison. Dans ce cas, la distinction entre entrée et sortie devient primordiale.

Nous considérons donc trois algèbres, a priori distinctes. L'algèbre « sémantique » sur laquelle les fonctions sont définies (et dans laquelle,

elles prennent leur valeur), l'algèbre des entrées qui représente les données et l'algèbre de sortie qui code le résultat. Un problème apparaît à ce niveau là : on ne dispose plus de la composition de manière immédiate. Pour cette raison, nous permettons à un symbole de servir à la fois de symbole d'entrée et de symbole de sortie. La nature entrée/sortie d'un symbole est caractérisée par deux fonctions (inp et out). Ce point de vue permet de retrouver les algorithmes (les systèmes de réécriture) que nous avions dans le cas classique où l'on ne fait pas de distinction (entre entrée et sortie).

Définition 1.4.11. Soient deux signatures $(\Sigma_i)_{i\in I}$ et $(\Sigma'_j)_{j\in J}$ telles que $I \subset J$. Un *morphisme de signature* $(\Sigma_i)_{i\in I} \to (\Sigma'_j)_{j\in J}$ est une famille de fonctions $m = (m_i)_{i\in I} : \Sigma_i \to \Sigma'_i$. En d'autres termes, les fonctions de *traduction* respectent l'arité des symboles. Un tel morphisme m s'étend immédiatement sur les termes $\mathcal{T}(\Sigma,\mathcal{X}) \to \mathcal{T}(\Sigma',\mathcal{X})$:

$$\begin{cases} m(x) = x \\ m(f(t_1,\cdots,t_n)) = m_n(f)(m(t_1),\cdots,m(t_n)) \end{cases}$$

On dit qu'un morphisme de signature est injectif si toutes ses composantes le sont.

Sémantique des calculs en termes de réécriture

Nous supposons que nous disposons d'une algèbre de référence $\mathcal{T}(\mathcal{A})$, l'algèbre de calcul. $\mathcal{T}(\mathcal{A})$ est l'ensemble sur lequel les fonctions que nous étudions dans cette section sont définies et dans lequel elles prennent leurs valeurs.

Définition 1.4.12. Un système de calcul pour l'algèbre de calcul $\mathcal{T}(\mathcal{A})$ est un sextuplet $(\mathcal{R},\mathbb{A},\mathbb{F},f,\mathtt{inp},\mathtt{out})$ où

- $(\mathcal{R},\mathbb{A}\cup\mathbb{F})$ est un système de réécriture qui termine ;
- la signature $\mathbb{A}$ est appelée ensemble des *constructeurs* ;
- $f \in \mathbb{F}$;
- $\mathtt{inp} : \mathcal{A} \to \mathbb{A}$ et $\mathtt{out} : \mathcal{A} \to \mathbb{A}$ sont deux *morphismes de signature injectifs* ;
- pour tout n-uple de termes $(t_1,\cdots,t_n) \in \mathcal{T}(\mathcal{A})$, les formes normales de $f(\mathtt{inp}(t_1),\cdots,\mathtt{inp}(t_n))$ sont dans $\mathcal{T}(\mathtt{out}(\mathcal{A}))$.

Définition 1.4.13 (Calculs déterministes). Une fonction $\phi : \mathcal{T}(\mathcal{A})^n \to \mathcal{T}(\mathcal{A})$ est *calculée* par le symbole f d'un système de calcul $(\mathcal{R},\mathbb{A},\mathbb{F},f,\mathtt{inp},\mathtt{out})$ si :

1. $\mathcal{R}$ est *confluent*,
2. pour tout n-uple de termes $(t_1,\cdots,t_n) \in \mathcal{T}(\mathcal{A})$, le diagramme suivant commute :

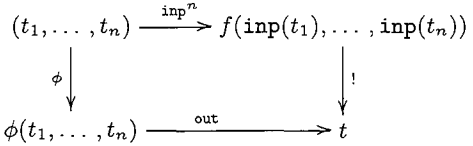


FIG. 1.1 – *Calculs par réécriture*

où t est l'unique forme normale obtenue d'après la proposition 1.4.10.

Remarque 1.4.14. L'hypothèse que les morphismes de codage doivent être injectifs se justifie par les deux observations suivantes :

1. Pour inp, il s'agit d'éviter qu'une identification des symboles d'entrée ne réalise une partie des calculs. Certes, la complexité de cette opération sera négligeable eu égard aux classes de complexité que nous considérons par la suite, mais comme nous pouvons toujours nous ramener au cas où inp est injectif, cela est sans importance.
2. Pour out, l'hypothèse est plus importante ; en effet, sans elle nous considérerions comme étant des algorithmes, des systèmes de réécriture calculant en une seule étape des problèmes de complexité arbitraire (en particulier, le problème de l'arrêt de la machine de Turing !). Prenons un problème de décision quelconque, et supposons que les données soient codées par une algèbre $\mathbb{A}$. Nous pouvons alors considérer le système de réécriture constitué de la signature $\mathbb{A} \cup \{f,1\}$ et de la règle de réécriture :

$$f(x) \to 1$$

Si nous choisissons out comme étant

$$\begin{array}{rll} \texttt{out}: & \text{vrai} & \mapsto 1 \\ & \text{faux} & \mapsto 1, \end{array}$$

ce système termine, il est confluent et le diagramme de la figure 2 commute ; pourtant, cela ne nous donne pas de procédure de décision effective qui résout le problème.

L'hypothèse que out est injective nous permet de décoder la forme normale. Un calcul procède en trois étapes : codage via inp, normalisation et décodage via out.

Exemple 1.4.15. Sur l'algèbre $\mathcal{T}(\texttt{Nat})$, la fonction

$$\begin{array}{rccc} (-)^2: & \mathcal{T}(\texttt{Nat}) & \to & \mathcal{T}(\texttt{Nat}) \\ & \underline{n} & \mapsto & \underline{n^2} \end{array}$$

est calculée par le symbole sq de l'exemple 1.4.7 ; il suffit pour cela de choisir :

- $\mathbb{A} = \texttt{Nat}$;
- $\mathbb{F} = \{\texttt{sq}, +\}$;
- $\texttt{inp} = \texttt{out} = 1_{\texttt{Nat}}$.

Définition 1.4.16 (Calculs non déterministes). À l'instar des machines de Turing non déterministes, nous pouvons considérer les systèmes pour lesquels c'est la « meilleure » des dérivations qui fournit le résultat du calcul. Cette manière de caractériser les fonctions calculées de façon non-déterministe est due à Krentel [Kre88] et Grädel et Gurevich [GG95]. Plus formellement, une fonction $\Phi : \mathcal{T}(\mathcal{A})^n \to \mathcal{T}(\mathcal{A})$ est *maximisée* par le symbole f du système de calcul $(\mathcal{R}, \mathbb{A}, \mathbb{F}, f, \texttt{inp}, \texttt{out})$ si :

1. il existe un ordre $\preceq$ sur l'algèbre $\mathcal{T}(\texttt{out}(\mathcal{A}))$,
2. pour tout n-uple de termes $(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{A})$,

$$\texttt{out}(\Phi(t_1, \cdots, t_n)) = \max_{\preceq} \{t \mid f(\texttt{inp}(t_1), \cdots, \texttt{inp}(t_n)) \xrightarrow{!}_{\mathcal{R}} t\}$$

Remarque 1.4.17. Le point délicat dans cette définition concerne l'ordre $\preceq$ sur l'algèbre $\mathcal{T}(\mathtt{out}(\mathcal{A}))$. Pour que les problèmes ne deviennent pas triviaux, nous devons nous restreindre à des ordres « raisonnables ». Nous entendons par là que l'ordre ne doit pas coder une partie trop conséquente des calculs, c'est-à-dire une partie que l'on peut détecter dans le cadre de la théorie. Si nous ne faisions pas d'hypothèse sur la procédure de calcul de l'ordre, nous considérerions comme étant un algorithme un système qui maximise le problème de l'arrêt de la machine de Turing. Voici comment procéder.

Soit $\mathbb{A}$ une algèbre qui code les machines de Turing. Nous considérons la fonction $\phi : \mathbb{A} \to \mathbb{A} \cup \{0\}$ qui envoie tout x qui correspond au code d'une machine de Turing qui ne s'arrête pas vers 0 et qui envoie tout x qui correspond à une machine qui s'arrête vers x. Cette fonction est maximisée par le système de réécriture suivant. Le système de réécriture est composé des deux règles $f(x) \to 0$ et $f(x) \to x$. Sur $\mathbb{A} \cup \{0\}$, nous définissons l'ordre suivant : $x < 0$ si x est le code d'une machine qui ne s'arrête pas, $0 < x$ sinon.

Pour les cas qui nous intéressent, le terme de « raisonnable » peut être remplacé par : « dont la décision prend un temps de calcul borné polynomialement ».

Exemple 1.4.18. Soit $\{a,b\}^*$, une algèbre de mots. Le problème de trouver le plus long sous-mot commun d'un ensemble (fini!) de mots sur $\{a,b\}$ est NP-complet (cf. Maier [Mai77]). Certes, il ne s'agit pas là du problème NP le plus célèbre ; nous l'avons choisi d'abord parce qu'il est simple à spécifier mais aussi parce qu'il est dans NLINSPACE, ce qui nous donnera l'occasion de l'utiliser souvent.

Nous représentons l'ensemble de mots $S = \{w_1,w_2,\cdots,w_n\}$ par le terme : $*(w_1*(w_2\cdots*(w_n*\epsilon)\cdots))$. La recherche du plus long sous-mot est spécifiée et maximisée par le système de réécriture :

$fax \to fx$	$ad\epsilon \to d\epsilon$	$gax \to fdax$
$fbx \to fx$	$bd\epsilon \to d\epsilon$	$gbx \to fdbx$
$fx \to x$	$d\epsilon \to \epsilon$	$g(*(x,y)) \to *(g(x),g(y))$
$*(fx,fx) \to fx$	$dax \to adx$	
$\mathbf{sous}\ x \to \mathtt{r}gx$	$dbx \to bdx$	$\mathtt{r}fx \to x$

Pour aider le lecteur à se convaincre que le symbole **sous** maximise la recherche du plus long sous-mot, nous donnons quelques indications. Le symbole f sert à effacer le début des mots tandis que la lettre d sert à supprimer la fin des mots. La lettre g répartit les symboles f et d dans toutes les branches ; les symboles **sous** et $\mathtt{r}$ gèrent le début et la fin des calculs.

- $\mathbb{A} = \{a,b,*\}$
- $\mathbb{F} = \{f,g,\mathtt{r},\mathbf{sous}\}$
- $\preceq$ est le plus petit ordre donné par :

  (i) si t contient un symbole $*$, alors $t \preceq t'$ pour tout t' ;
  (ii) si t et t' ne contiennent pas de symbole $*$, $t \preceq t'$ si $|t| < |t'|$;
  (iii) si t et t' ne contiennent pas de symbole $*$ et que $|t| = |t'|$, alors $t < t'$ si t est plus petit pour l'ordre alphabétique que t'.

- $\mathtt{inp} = \mathtt{out} = 1_{\mathbb{A}}$.
  Prenons par exemple l'ensemble $\{abbab, abab, bbabaa, ababab\}$. Le plus long sous-mot est « bab ». Les formes normales du système sont dans l'ordre $\{\epsilon,a,b,ab,ba,bab\}$, et « bab » est bien le plus grand élément de cet ensemble.

Ici, nous faisons grand usage de la précédence ; la recherche de la branche maximale a un contenu calculatoire important. En effet, la longueur de dérivation pour chaque branche est de l'ordre de $O(n)$, il y a donc un nombre exponentiel de branches, et seule l'une d'entre elles fournit le résultat.

1.5 Machines de Turing

Nous faisons ici une présentation des machines de Turing, essentiellement pour fixer les notations, le vocabulaire, et préciser le sens de « calculer une fonction à l'aide d'une machine de Turing » ; en particulier, nous discuterons le cas des machines non déterministes.

Nous présentons ici un modèle standard de machine (non déterministe), avec un ruban infini bilatère et une tête de lecture/écriture unique. Mais, du fait de la robustesse de ce modèle vis-à-vis des classes

de complexité que nous considérons (au delà de LINSPACE ou bien PTIME), ce choix ne pose pas de problème. Nous utiliserons d'ailleurs plusieurs modèles de machines au gré des convenances que cela nous apporte.

Définition 1.5.1. Une machine de Turing $\mathcal{M}$ est un 5-uplet $\mathcal{M} = (\Sigma,\#,Q,q_0,q_f,\delta)$ où :

- Σ est un ensemble appelé *alphabet*,
- $\# \in \Sigma$ est un caractère particulier, le caractère *blanc*,
- Q est un ensemble, dit ensemble des états,
- $q_0 \in Q$ est l'état *initial*,
- $q_f \in Q$ est l'état *final*,
- δ est une relation $\delta \subset Q \times \Sigma \times Q \times \Sigma \times \{-1,0,+1\}$.

Définition 1.5.2. Une *configuration* est la donnée d'un triplet (r,x,q) où :

- r est une fonction $\mathbb{Z} \to \Sigma$, le *ruban de la machine*,
- $x \in \mathbb{Z}$ est la *position courante de la tête de lecture/écriture*,
- $q \in Q$ est l'*état courant de la machine.*

Définition 1.5.3. Étant donné une machine $\mathcal{M} = (\Sigma,\#,Q,q_0,q_f,\delta)$, une *transition en une étape* est la plus petite relation binaire sur les configurations qui contient les paires $((r,x,q),(r',x',q'))$ telles que :

- $(q,r(x),q',a,m) \in \delta$,
- $\begin{cases} r'(y) = r(y) & \text{pour tout } y \neq x, \\ r'(x) = a, \end{cases}$
- $x' = x + m$.

En d'autres termes, la configuration (r',x',q') est la configuration (r,x,q) pour laquelle on a remplacé la lettre $r(x)$ sur le ruban par a, on a déplacé la tête de lecture/écriture de m cases vers la droite et l'état courant est q'.

Nous notons la relation par : $(r,x,q) \vdash_{\mathcal{M}} (r',x',q')$ et par analogie avec la réécriture, nous notons $(r,x,q) \vdash^*_{\mathcal{M}} (r',x',q')$ la fermeture transitive de la relation de transition en une étape. Cette relation est la relation de *transition.*

Définition 1.5.4. Enfin, un calcul procède de la façon suivante. Étant donné un n-uplet $(w_1, \cdots, w_n)$ de mots sur une algèbre $A \subseteq (\Sigma \backslash \{\#\})$, nous construisons le mot $w = w_1 \# w_2 \# \cdots \# w_n$, la configuration initiale est alors $(r_0, 1, q_0)$:

$$\begin{aligned} r_0(y) &= \quad \# \quad \text{si } y \leq 0 \text{ où si } y > |w|, \\ r_0(y) &= \quad w[y] \quad \text{si } 0 < y \leq |w|. \end{aligned}$$

Un *calcul* est la relation binaire sur $A^{*n} \times A^*$ définie par : $(w_1, \cdots, w_n) \vdash_{\mathcal{M}} w$ s'il existe une configuration (r, x, q_f) telle que $(r_0, 1, q_0) \xrightarrow{*}_{\mathcal{R}} (r, x, q_f)$ où $(r_0, 1, q_0)$ est la configuration initiale pour $(w_1, \ldots, w_n)$ et (r, x, q_f) est donné par :

$$\begin{aligned} r(y) &= \quad \# \quad \text{pour tout } y \leq 0 \text{ ou } y > |w|, \\ r(y) &= \quad w[y] \quad \text{pour tout } 0 < y \leq |w|. \end{aligned}$$

1.5.1 Calculs sur une algèbre

Comme pour la réécriture, nous donnons ici la sémantique du modèle de calcul. Le problème vient ici de la nécessaire transformation des termes en mots. Pour cela, nous utilisons une fonction de codage. Nous supposons comme dans la section précédente que nous disposons d'une algèbre $\mathcal{A}$ de calcul.

Définition 1.5.5. Une fonction de codage est une fonction $\mathtt{code} : \mathcal{A} \to A$, fonction qui est étendue en une fonction $\mathcal{T}(\mathcal{A}) \to (A \cup \{(,)\})^*$:

$$\mathtt{code}(f(t_1, \cdots, t_n)) = \mathtt{code}(f)\Big(\mathtt{code}(t_1)\Big)\Big(\cdots\Big)\Big(\mathtt{code}(t_n)\Big)$$

Par la suite, nous ne faisons plus de distinction entre A et $A \cup \{(,)\}$.

Définition 1.5.6 (Calculs non déterministes). Nous disons d'une fonction $\Phi : \mathcal{A}^{*n} \to \mathcal{A}^*$ qu'elle est *maximisée* par une machine de Turing $\mathcal{M} = (A \cup F, \#, Q, q_0, q_f, \delta)$ si :

- F contient la lettre $\#$;
- il existe une fonction de codage injective $\mathtt{code} : \mathcal{A} \to A$;
- il existe une notion de précédence sur les mots de A ;
- pour tout n-uple de termes $w_1, w_2, \cdots, w_n \in \mathcal{T}(\mathcal{A})$, on a :

$$\mathtt{code}(\Phi(w_1, w_2, \cdots, w_n)) = \max\{t \mid \mathtt{code}(w_1)\#\mathtt{code}(w_2)\#\cdots\#\mathtt{code}(w_n) \Rightarrow t\}$$

$$\begin{array}{ccc} (w_1, \cdots, w_n) & \longrightarrow & \mathtt{code}(w_1)\#\mathtt{code}(w_2)\#\cdots\#\mathtt{code}(w_n) \\ \Big\downarrow{\scriptstyle \Phi} & & \Big\downarrow{\scriptstyle \mathcal{M}} \\ \Phi(w_1, \cdots, w_n) & \xrightarrow{\quad\mathtt{code}\quad} & w' \end{array}$$

Remarque 1.5.7. Notons ici que le codage (plus précisément, l'extension du codage) est fondé sur la notation polonaise. D'autres codages sont possibles, nous avons choisi celui-ci pour sa simplicité. La principale propriété que le codage doit vérifier est que ce dernier ne code pas de calculs, tout du moins de calculs non négligeables eu égard au cadre dans lequel on se place, ce qui est le cas de la notation polonaise pour les cas qui nous concernent.

D'autre part, vis-à-vis de la précédence, nous faisons les mêmes hypothèses que pour la réécriture, et pour les mêmes raisons.

Définition 1.5.8 (Calculs déterministes). Dans le cas particulier où δ est un graphe de fonction $Q \times \Sigma \to Q \times \Sigma \times \{-1, 0, +1\}$, la relation de transition en un pas devient une fonction des configurations dans les configurations. Il s'ensuit que la relation de calcul devient également une fonction (partielle) des mots $A^n \to A$. Comme il n'y a plus qu'une seule branche de calcul, la notion de précédence n'est plus nécessaire. La machine est dite alors *déterministe* et on dit que la fonction est *calculée* par la machine de Turing.

Définition 1.5.9 (Classes de complexité). Étant donné une fonction $f : \mathbb{N} \to \mathbb{N}$, on dit que la machine travaille en temps $f(n)$ si pour toute entrée de taille au plus n, le nombre de transitions nécessaires pour atteindre l'état final est borné par $f(n)$.

Étant donné une classe de fonctions $\mathbb{C}$, la classe de fonctions TIME($\mathbb{C}$) est l'ensemble des fonctions qui sont calculées par une machine déterministe qui travaille en temps borné par $f(n)$ pour un certain $f \in \mathbb{C}$ (le paramètre n représente habituellement la taille des entrées). Lorsque l'on autorise des calculs avec une machine non-déterministe, on ajoute un « N » devant le nom de la classe.

PTIME (resp. NPTIME) désigne la classe des fonctions calculées par les machines de Turing déterministes (resp. non déterministe) qui travaillent en temps polynomial ; ETIME (resp. NETIME) désigne les fonctions calculées par les machines déterministes (resp. non déterministes) qui travaillent en temps exponentiel (c'est-à-dire borné par 2^{kn} pour un certain $k \in \mathbb{N}$) et E$_2$TIME (resp. NE$_2$TIME) celles qui travaillent en temps doublement exponentiel (c'est-à-dire borné par $2^{2^{kn}}$ pour un certain $k \in \mathbb{N}$).

De même, si au cours d'un calcul, pour toute donnée de longueur au plus n, une machine utilise au plus $f(n)$ cases pour atteindre la forme normale, on dit de la machine qu'elle travaille en espace $f(n)$. La classe des fonctions calculées par les machines déterministes (respectivement non déterministes) qui travaillent dans l'espace $f(n)$ pour un certain $f \in \mathbb{C}$ est notée SPACE($\mathbb{C}$) (respectivement NSPACE($\mathbb{C}$)).

LINSPACE (resp. NLINSPACE) correspond aux machines déterministes (resp. non déterministes) qui travaillent en espace linéaire, c'est-à-dire borné par $a * n + b$ pour deux constantes a et b ; PSPACE, NPSPACE, ESPACE, NESPACE, E$_2$SPACE, NE$_2$SPACE correspondent à l'espace polynomial, exponentiel et doublement exponentiel (respectivement déterministes et non déterministes).

Nous énonçons ici un théorème dû à Savitch [Sav70] qui donne des égalités pour les classes en espace :

Théorème 1.5.9.1.
Une machine de Turing non déterministe qui travaille en espace $L(n)$ peut être simulée par une machine déterministe qui travaille en espace $L(n)^2$ à la condition que $L(n) \geq \log_2(n)$.

Corollaire 1.5.10. Nous avons par conséquent les égalités : NPSPACE = PSPACE, NESPACE = ESPACE et NE_2SPACE = E_2SPACE.

1.6 Théorie de la récursion

Ainsi que nous l'avons vu dans l'introduction du chapitre, nous présentons ici la théorie de la récursion, tout du moins, la partie qui nous intéresse pour la suite. Les fonctions récursives sont les fonctions de l'arithmétique qui peuvent être *calculées* de manière effective (selon la thèse de Church). Pour cela, l'accent est mis sur le caractère inductif des entiers naturels :

$$0 \text{ est un entier ;}$$
$$\text{si } n \text{ est un entier, } s(n) \text{ est un entier.}$$

Au même titre que de nombreuses démonstrations de propriétés des nombres entiers (cf. Poincaré [Poi03] par exemple), les définitions des fonctions numériques courantes reposent sur le schéma d'induction.

1.6.1 Récursion primitive

Définition 1.6.1. L'ensemble $\mathcal{B}$ des *fonctions de base* est composé de :

(i) la fonction 0 : $\begin{array}{ccc} \mathbb{N} & \to & \mathbb{N} \\ n & \mapsto & 0 \end{array}$

(ii) la fonction successeur s : $\begin{array}{ccc} \mathbb{N} & \to & \mathbb{N} \\ n & \mapsto & n+1 \end{array}$

(iii) pour tout $i \leq k$, la fonction projection :

$$\begin{array}{rccc} \pi_{i,k} : & \mathbb{N}^k & \to & \mathbb{N} \\ & (x_1, \cdots, x_k) & \mapsto & x_i \end{array}$$

On remarquera que $\pi_{1,1}$ est la fonction identité des entiers naturels.

Définition 1.6.2 (Schéma de composition). Soit une fonction f à m arguments et m fonctions $(g_i)_{i\leq m}$ à k arguments. La composée de f et $(g_i)_{i\leq m}$ est la fonction :

$$\begin{array}{rcl} \mathtt{Comp}(f,(g_i)_{i\leq m}) : \mathbb{N}^k & \to & \mathbb{N} \\ (x_1,\cdots,x_k) & \mapsto & f(g_1(x_1,\cdots,x_k),g_2(x_1,\cdots,x_k),\cdots,g_m(x_1,\cdots,x_k)) \end{array}$$

Remarque 1.6.3. Les projections servent à permuter, dupliquer, effacer des arguments des fonctions composées.

Définition 1.6.4 (Fonctions élémentaires). La classe $\mathcal{E}$lémentaire des *fonctions élémentaires* est l'ensemble de fonctions clos par composition contenant les fonctions de base $\mathcal{B}$, l'addition, le produit, la soustraction modifiée $\dot{-}$, les sommes et produits bornés.

- La soustraction modifiée $\dot{-} : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ est définie par :

$$(n,m) \mapsto \begin{cases} 0 & \text{si } n \leq m \\ n-m & \text{sinon} \end{cases}$$

- Soit f une fonction à $k+1$ arguments. Nous définissons la somme bornée $\Sigma f : \mathbb{N}^{k+1} \to \mathbb{N}$:

$$\Sigma f : (x_0,x_1,\cdots,x_k) \mapsto f(0,x_1,\cdots,x_k)+f(1,x_1,\cdots,x_k)+\cdots+f(x_0,x_1,\cdots,x_k)$$

- De manière analogue, le produit borné $\Pi f : \mathbb{N}^{k+1} \to \mathbb{N}$ est défini :

$$\Pi f : (x_0,x_1,\cdots,x_k) \mapsto f(0,x_1,\cdots,x_k)\times f(1,x_1,\cdots,x_k)\times\cdots\times f(x_0,x_1,\cdots,x_k)$$

La classe des fonctions élémentaires a été introduite par Kalmar [Kal43] et Csillag [Csi47] ; elle contient la plupart des fonctions « usuelles ».

Définition 1.6.5 (Schéma de récursion primitive). Étant donné une fonction g à k arguments et une fonction h à $k+2$ arguments, nous définissons la fonction $R(g,h)$:

$$R(g,h) : \begin{array}{rcl} \mathbb{N}^{k+1} & \to & \mathbb{N} \\ (0,x_1,\cdots,x_k) & \mapsto & g(x_1,\cdots,x_k) \\ (s(n),x_1,\cdots,x_k) & \mapsto & h(f(n,x_1,\cdots,x_k),n,x_1,\cdots,x_k) \end{array}$$

Exemple 1.6.6. Sont définis par récursion :

- l'addition :

$$(+) : \begin{cases} 0+y &= y \\ s(x)+y &= s(x+y) \end{cases} \qquad \begin{cases} g = \pi_{1,1} = 1_{\mathbb{N}} \\ h = \mathtt{Comp}(s,\pi_{1,3}) \end{cases}$$

- la multiplication :

$$(\times) : \begin{cases} 0 \times y &= 0 \\ s(x) \times y &= x \times y + y \end{cases} \qquad \begin{cases} g = 0 \\ h = \mathtt{Comp}(+,(\pi_{1,3},\pi_{3,3})) \end{cases}$$

Définition 1.6.7 (Fonctions récursives primitives). La classe des fonctions récursives primitives $\mathcal{PRF}$[10] est l'ensemble de fonctions obtenu inductivement par les règles :

- si $f \in \mathcal{B}$, alors $f \in \mathcal{PRF}$;
- si $f \in \mathcal{PRF}$ et pour tout i, $(g_i)_{i\leq n} \in \mathcal{PRF}$, alors $\mathtt{Comp}(f,(g_i)_{i\leq n}) \in \mathcal{PRF}$;
- si $g,h \in \mathcal{PRF}$, alors $R(g,h) \in \mathcal{PRF}$.

Exemple 1.6.8. L'exemple 1.6.6 montre que la somme est récursive primitive. Par conséquent, il en est de même du produit. À partir de là, nous pourrions montrer que l'exponentielle est aussi récursive primitive, etc.

Certaines fonctions calculables ne sont pas primitives récursives. Le plus célèbre exemple, sans doute, est donné par la fonction d'Ackermann (1928) :

10. pour « Primitive Recursive Functions ».

$$\begin{aligned} A(0,n) &= n+1 \\ A(m+1,0) &= A(m,1) \\ A(m+1,n+1) &= A(m,A(m+1,n)) \end{aligned}$$

Pour montrer que cette fonction n'est pas primitive récursive, on montre que son taux d'accroissement est plus grand que celui de n'importe quelle fonction primitive récursive.

Proposition 1.6.9. Le lecteur vérifiera aisément que la soustraction modifiée est récursive primitive et que pour toute fonction f récursive primitive, Σf et Πf le sont. Par conséquent, nous avons l'inclusion des classes : $\mathcal{E}$lémentaire $\subset \mathcal{PRF}$.

Récursion primitive et boucles FOR

Le lecteur « informaticien » verra que le schéma de récursion correspond à la boucle « FOR » des langages de programmation impérative.

Définition 1.6.10 (Programmes à boucles « FOR »). Cf. par exemple Handley et Wainer [HW97]. Les programmes à boucles FOR sont les programmes construits à partir des instructions :

- les assignations : $x := 0$, $x := y$, $x := y + 1$, $x := y \dot{-} 1$
- les séquencements : $\ldots ; \ldots$
- les conditionnelles : **if** $(x = y)$ **then** $\cdots$ **else** $\cdots$ **fi**
- les boucles FOR : **for** $i = 1$ **to** y **do** $\cdots$ **od**, où i n'est pas assigné dans $\cdots$

Théorème 1.6.10.1 (cf. Handley et Wainer [HW97]).
Les fonctions récursives primitives sont exactement les fonctions calculées à l'aide des programmes à boucle FOR.

Approche logique de la récursion primitive

Définition 1.6.11 (Arithmétique de Peano). L'arithmétique de Peano [Pea88] est le système formel composé des axiomes :

$$\begin{array}{ll} \forall x\ sx \neq 0 & \forall x\ x+0=x \\ \forall x\ \exists y\ x \neq 0 \Rightarrow x=sy & \forall x\ \forall y\ x+s(y)=s(x+y) \\ \forall x\ \forall y\ sx=sy \Rightarrow x=y & \forall x\ x\times 0=0 \\ & \forall x\ \forall y\ x\times s(y)=x\times y+x \end{array}$$

et du schéma d'induction :

$$\forall(x_1,\cdots,x_n)\phi(0,x_1,\cdots,x_n)\wedge(\phi(n,x_1,\cdots,x_n)\Rightarrow\phi(s(n),x_1,\cdots,x_n))\Rightarrow \forall(x_0,\cdots,x_n)\ \phi(x_0,x_1,\cdots,x_n)$$

où $\phi(x_0,\cdots,x_n)$ est une formule dont les variables libres sont $(x_0,\cdots,x_n)$.

Définition 1.6.12. Une formule est dite Σ_0 si elle est de la forme $\exists y\, P$ où P est un prédicat sans quantificateur. Σ_0-IND est la restriction de l'arithmétique de Peano obtenue en restreignant le schéma d'induction aux formules de Σ_0. Une fonction est dite prouvablement totales dans Σ_0-IND s'il existe une relation R telle que

- $f(x)=\min\{y \mid R(x,y)\}$ et,
- $\forall x\exists y.R(x,y)$ est un théorème de Σ_0-IND.

Théorème 1.6.12.1 (Parsons [Par66]).
Les fonctions prouvablement totales dans Σ_0-IND sont récursives primitives.

1.6.2 Fonctions récursives

Définition 1.6.13 (Schéma de minimisation non bornée). Soit une fonction f à $k+1$ arguments. Nous définissons la fonction (partielle) $(\mu t)f$ à k arguments comme suit :

$$\begin{array}{rccl} (\mu t)f: & \mathbb{N}^k & \rightarrow & \mathbb{N} \\ & (x_1,\cdots,x_k) & \mapsto & \min_{t\in\mathbb{N}}\{t \mid f(t,x_1,\cdots,x_k)=0\} \end{array}$$

La classe des fonctions récursives est alors l'ensemble des fonctions récursives primitives étendu par le schéma de minimisation non bornée.

Théorème 1.6.13.1 (Kleene [Kle36]).
La classe des fonctions récursives est complète au sens de Turing[11].

1.6.3 La hiérarchie de Grzegorczyk

Une propriété intéressante de la théorie des fonctions récursives primitives est que l'on peut calculer aisément le nombre d'étapes nécessaires à l'évaluation d'une fonction de cette classe. On peut également borner le taux de croissance des fonctions récursives primitives. Il s'avère que nous pouvons utiliser ce taux de croissance pour construire une classification des fonctions récursives primitives : il s'agit de la hiérarchie de Grzegorczyk. Le critère utilisé correspond en fait au nombre d'utilisations imbriquées du schéma de récursion (cf. Handley et Wainer [HW97]).

Définition 1.6.14. Grzegorczyk construit dans [Grz53] une suite $\mathcal{E}_0 \subset \mathcal{E}_1 \subset \mathcal{E}_2, \ldots$ de classes de fonctions (croissante pour l'ordre par inclusion) telle que chaque classe est close pour la composition et le schéma de récursion bornée. f est obtenue par récursion bornée si

- $f = R(g,h)$,
- il existe une fonction j (définie auparavant) telle que $f \leq j$, c'est-à-dire $\forall n, f(n) \leq j(n)$,

La classe $\mathcal{E}_i$ est la classe de fonctions close par composition et récurrence bornée qui contient les fonctions de base et la fonction E_i définie comme suit,

- $E_0(x,y) = x + 1$;
- $E_1(x,y) = x + y$;
- $E_2(x) = (x + 1)(y + 1)$;
- et pour tout $n \geq 2$,

$$\begin{aligned} E_{n+1}(0,y) &= E_n(y,y) \\ E_{n+1}(x+1,y) &= E_n(x,E_n(x,y)) \end{aligned}$$

11. Comme nous l'avons vu précédemment, la démonstration originale de Kleene donne une équivalence avec les fonctions récursives générales de Gödel.

Nous donnons ici quelques théorèmes concernant la hiérarchie de Grzegorczyk, théorèmes que nous réutiliserons par la suite.

Théorème 1.6.14.1 (Grzegorczyk [Grz53]).

1. $\forall i < j \ \mathcal{E}_i \subset \mathcal{E}_j$;
2. $\bigcup_{i \in \mathbb{N}} \mathcal{E}_i = \mathcal{PRF}$.

Théorème 1.6.14.2 (Grzegorczyk [Grz53]).

$$\mathcal{E}_3 = \mathcal{E}lémentaire$$

Théorème 1.6.14.3 (Ritchie [Rit63]).

$$\mathcal{E}_2 = \text{LINSPACE}$$

1.6.4 Hiérarchies de fonctions sous-récursives

Dans le but d'obtenir des classes de fonctions plus grandes que $\mathcal{PRF}$, Löb et Wainer [LW70a, LW70b, LW71, Wai72] ont étendu le procédé de Grzegorczyk en proposant des suites transfinies de classes de fonctions. Pour cela, ils s'appuient sur la notion de diagonalisation.

Définition 1.6.15. Étant donné une famille $(f_n)_{n \in \mathbb{N}}$ de fonctions de $\mathbb{N}$ dans $\mathbb{N}$, la *diagonale* de la famille est la fonction Δf :

$$\begin{array}{rccc} \Delta f : & \mathbb{N} & \to & \mathbb{N} \\ & n & \mapsto & f_n(n) \end{array}$$

Si maintenant, on suppose que le taux d'accroissement des fonctions f_n croît avec n, le taux d'accroissement de la diagonale est plus grand encore que celui de chacune des fonctions de la famille. De manière similaire, les ordinaux limites sont les limites de chaînes d'ordinaux plus petits. Ce parallèle aboutit à la construction des hiérarchies de fonctions à la manière de Löb et Wainer ; nous nommons ces fonctions les fonctions sous-récursives.

Définition 1.6.16 (Suites fondamentales). Étant donné un ordinal α, une suite fondamentale de α est une suite croissante d'ordinaux $(\alpha_\gamma)_{\gamma \in \Lambda}$ (où Λ est un ordinal) satisfaisant :

$$\alpha = \sup\{\alpha_\gamma \mid \gamma \in \Lambda\}$$

En fait, nous n'utiliserons que des suites indexées par ω. Nous utilisons donc le terme de suite fondamentale pour désigner les ω-chaînes d'ordinaux.

Définition 1.6.17 (Hiérarchies à croissance lente). Nous supposons que nous disposons pour un segment initial de Ω, d'une assignation d'une séquence fondamentale $(\lambda_x)_{x\in\omega}$ pour chaque ordinal limite λ, la hiérarchie de fonctions à croissance lente est donnée par :

$$\begin{aligned} G_0(x) &= 0 \\ G_{\alpha+1}(x) &= G_\alpha(x) + 1 \\ G_\lambda(x) &= G_{\lambda_x}(x) \end{aligned}$$

Définition 1.6.18 (Hiérarchie à croissance rapide).

$$\begin{aligned} F_0(x) &= x + 1 \\ F_{\alpha+1}(x) &= F_\alpha^{(x+1)}(x) \\ F_\lambda(x) &= F_{\lambda_x}(x) \end{aligned}$$

Définition 1.6.19 (Hiérarchie de Hardy).

$$\begin{aligned} H_0(x) &= x \\ H_{\alpha+1}(x) &= H_\alpha(x+1) \\ H_\lambda(x) &= H_{\lambda_x}(x) \end{aligned}$$

Exemple 1.6.20. Pour les ordinaux finis, n, on peut donner une expression littérale des fonctions G_n, H_n. On en déduit une expression de G_ω et de H_ω en prenant comme suite fondamentale de ω la séquence $\omega_x = x$ pour tout $x \in \omega$.

$$\begin{aligned} G_n(x) &= n & \qquad G_\omega(x) &= x \\ H_n(x) &= x + n & \qquad H_\omega(x) &= 2 * x \end{aligned}$$

1.7 λ-calculs

Dans cette section, nous proposons une brève description de trois λ-calculs typés : le λ-calcul simplement typé ordinaire, le calcul linéaire et la théorie des types dépendants. Le λ-calcul va bien sûr au delà des cas qui nous préoccupent. Il y a lieu de mentionner par exemple le calcul non typé que le lecteur découvrira dans le livre de Barendregt [Bar84] ou celui de Hindley et Seldin [HS86].

Aux chapitres 6 et 7, nous définirons une notion d'algèbre pour laquelle nous donnerons une syntaxe. Comme nous aurons à analyser différentes sortes de λ-calculs (linéaire, types simples ordinaires), nous avons choisi de présenter le λ-calcul de manière suffisamment générale pour recouvrir chacun des cas que nous rencontrerons.

1.7.1 Calculs simplement typés

Nous utilisons le qualificatif de « simplement typé » pour recouvrir le calcul linéaire et le calcul simplement typé ordinaire ; cela est à mettre en opposition avec les types dépendants.

Définition 1.7.1 (Types simples). Nous supposons que nous disposons d'un ensemble `At` de types atomiques. Les types sont obtenus inductivement par les deux règles :

$$\frac{}{\texttt{At} \in \text{ type}} \qquad \frac{X \in \text{ type} \qquad Y \in \text{ type}}{X \multimap Y \in \text{ type}}$$

En ce qui concerne les termes, nous supposons que nous disposons d'un ensemble Σ de termes atomiques typés. Plus précisément, nous nous donnons pour chaque symbole $f \in \Sigma$ une suite de types $(X_1, \ldots, X_n, Y)$ où la suite des X_i est éventuellement vide. Nous écrivons ceci $f : X_1, \ldots, X_n \to Y$. Par analogie avec les termes, nous appelons Σ la signature.

En outre, pour chaque type X, nous nous donnons un ensemble V_X de variables.

Définition 1.7.2 (Calcul linéaire). Nous considérons les règles de formation de termes suivantes :

$$\frac{x \in V_X}{x : X \vdash x : X}\ \text{Axiome} \qquad \frac{\Gamma, x : X, y : Y, \Delta \vdash t : Y}{\Gamma, y : Y, x : X, \Delta \vdash t : Y}\ \text{Échange}^{12}$$

$$\frac{\forall i \leq n, \Gamma_i \vdash t_i : X_i}{\bigcup_{i=1}^{n} \Gamma_i \vdash f(t_1, \ldots, t_n) : Y}\ f \in \Sigma$$

$$\frac{\Gamma, x : X \vdash t : Y}{\Gamma \vdash \lambda x.t : X \multimap Y}\ \text{Abstraction} \qquad \frac{\Gamma \vdash t : X \multimap Y \qquad \Delta \vdash a : X}{\Gamma, \Delta \vdash (t)a : Y}\ \text{Application}$$

avec les contraintes suivantes qui expriment la linéarité du calcul :

- dans la règle d'application, on suppose que *les contextes* Γ *et* Δ *sont disjoints*,
- dans la règle d'introduction du symbole atomique f, *les contextes* Γ_i *sont disjoints deux à deux.*

Définition 1.7.3 (Calcul simplement typé ordinaire). Le calcul simplement typé ordinaire est le calcul dans lequel on utilise les règles de formation des termes ci-dessus mais avec des contextes éventuellement non-disjoints dans la règle d'introduction du symbole atomique f et dans la règle d'application. En plus de cela, on ajoute la règle d'affaiblissement :

$$\frac{}{x_1 : X_1, \ldots, x_n : X_n \vdash x_i : X_i}\ \text{Affaiblissement}$$

Pour le λ-calcul simplement typé ordinaire, nous utilisons le symbole « $\rightarrow$ » à la place du symbole « $\multimap$ ».

Définition 1.7.4. Une occurence de la variable x est dite *libre* dans le terme t dans les cas suivants :

- x est libre dans x,
- si x est libre dans r ou s, alors x est libre dans $t = (r)s$,
- si x est libre dans r, alors x est libre dans $t = \lambda y.r$ où $x \neq y$.

Une occurence d'une variable qui n'est pas libre est dite *liée*. Si x est une variable de t, on dit que x est liée par λ dans $\lambda x.t$. Si

12. Noter que la règle peut être dérivée des autres règles

une variable apparaît plusieurs fois dans un terme, certaines de ces occurences peuvent être libres, d'autres liées. Par exemple, dans le terme $(\lambda x.x)x$, la première occurence (c'est-à-dire le deuxième x, on ne compte pas le premier) est libre, la deuxième est liée.

Définition 1.7.5. Soit deux termes t et r et une variable x. On note $t[x\backslash r]$ le terme obtenu en substituant la variable x par r dans le terme t (sans capture de variables). De manière plus formelle :

- $f[x\backslash r] = f$ pour tout symbole f de la signature,
- $x[x\backslash r] = r$ pour toute variable x,
- $y[x\backslash r] = y$ pour toute variable $y \neq x$,
- $(u)v[x\backslash r] = (u[x\backslash r])v[x\backslash r]$, pour tous termes u et v,
- $\lambda x.u[x\backslash r] = \lambda x.u$ pour tout terme u,
- $\lambda y.u[x\backslash r] = \lambda y.u[x\backslash r]$ si $y \neq x$. Pour cette règle, nous supposons en outre que la variable y n'apparaît pas dans r (pour éviter toute capture de variables, c'est-à-dire qu'une variable libre dans r soit liée dans $\lambda y.u[x\backslash r]$).

Lemme 1.7.6 (Lemme de substitution). Par induction, on obtient la règle de substitution (syntaxique) :

$$\frac{x_1 : X_1, \ldots, x_n : X_n \vdash t : Y \qquad \forall i \leq n,\ \Delta_i \vdash a_i : X_i}{\bigcup_{i=1}^{n} \Delta_i \vdash t[x_1\backslash a_1, \ldots, x_n\backslash a_n] : Y}$$

où nous supposons que les contextes Δ_i sont disjoints deux à deux pour le calcul linéaire et quelconques pour le calcul simplement typé ordinaire. $t[x_1\backslash a_1, \ldots, x_n\backslash a_n]$ représente le terme t dans lequel on a remplacé les variables x_i par a_i pour tout $i \leq n$. Pour abréger l'écriture, on note ce terme $t[\vec{x}\backslash\vec{a}]$. Lorsqu'une famille $(a)_-$ de variables/de termes dépend de plusieurs indices, (i,j,k) par exemple, $\vec{a}_{(i,j)}$ désigne la sous-famille $a_{(i,j,1)}, \ldots, a_{(i,j,n)}$. En d'autres termes, on ne note pas les indices sur lesquels on fait la quantification.

Équivalence

Définition 1.7.7 (α-équivalence). Deux termes t et r sont α-équivalents s'ils ne diffèrent que par des renommages de variables liées n'introduisant pas de capture. Nous avons par exemple $\lambda x.\lambda y.xy =_\alpha \lambda z\lambda y.zy$ mais pas $\lambda x.\lambda y.xy =_\alpha \lambda y.\lambda y.yy$. Dans le membre de droite de la deuxième (in)égalité, les deux y sont liés par un seul λ (celui qui est le plus à l'intérieur) alors que dans les trois autres termes, les variables sont liées à un λ différent. Dorénavant, nous considérons égaux par définition les termes α-équivalents[13].

Définition 1.7.8 (β-équivalence). La relation $=_\beta$ est définie comme étant la plus petite relation d'équivalence telle que $(\lambda x.t)r =_\beta t[x\backslash r]$ et qui est close par contexte, c'est-à-dire pour tout $t =_\beta u$:

- $(t)r =_\beta (u)r$
- $(r)t =_\beta (r)u$
- $\lambda x.t =_\beta \lambda x.u$

Définition 1.7.9 (η-équivalence). La relation $=_\eta$ est définie comme étant la plus petite relation d'équivalence telle que $(\lambda x.tx) =_\eta t$ et qui est close par contexte, c'est-à-dire pour tout $t =_\eta u$:

- $(t)r =_\eta (u)r$
- $(r)t =_\eta (r)u$
- $\lambda x.t =_\beta \lambda x.u$

Confluence, Normalisation

Même si nous n'utilisons pas en tant que tel le λ-calcul comme un modèle de calcul, cet aspect du λ-calcul fait partie de nos perspectives. Pour calculer, on oriente les équations de la β-équivalence, c'est la β-réduction. La relation $\rhd$ est la plus petite relation qui contient $(\lambda x.t)r \rhd t[x\backslash r]$ et qui est close par contexte, c'est-à-dire pour tout $t \rhd u$:

- $(t)r \rhd (u)r$
- $(r)t \rhd (r)u$

13. Pour éviter le problème de l'α-équivalence, nous pourrions utiliser des indices de de Bruijn [dB72] ; nous utilisons une présentation avec des variables pour une raison de lisibilité.

– $\lambda x.t \rhd \lambda x.u$

Si $t \rhd r$, on dit que t se β-réduit en une étape en r. On note $\rhd^*$ la fermeture réflexive transitive de la relation $\rhd$; si $t \rhd^* r$, on dit que t se β-réduit en r.

Théorème 1.7.9.1 (Church-Rosser [CR36]).
Le λ-calcul simplement typé est confluent pour la réduction β. C'est-à-dire que pour tous termes t, u, v tels que $t \rhd^* u$ et $t \rhd^* v$, il existe un terme r tel que $u \rhd^* r$ et $v \rhd^* r$.

Définition 1.7.10. Un terme t est dit en forme normale s'il n'y a pas de terme u tel que $t \rhd u$.

Théorème 1.7.10.1 (Tait [Tai67]).
Le λ-calcul simplement typé est fortement normalisable. En d'autres termes, toute dérivation $t_1 \rhd t_2 \cdots$ est finie.

Les deux théorèmes nous fournissent un procédé d'évaluation. Pour calculer la « valeur » d'un terme t, il suffit de le β-réduire jusqu'à obtenir une forme normale. En raison de la confluence, on peut affirmer que cette forme normale est unique, elle ne dépend pas du choix des redex que l'on réduit lors de la normalisation.

Remarque 1.7.11. Outre la preuve du théorème due à Tait, nous mentionnons celle de de Groote qui utilise des arguments syntaxiques [dG93] (suivant une idée de Gandy).

1.7.2 λ-calcul avec types dépendants

Dans la théorie des types dépendants, la syntaxe des types peut contenir des termes. Ceci permet par exemple de considérer des types comme *les listes de k éléments de type X* où k est un terme représentant un entier naturel. En ce qui nous concerne, nous serons surtout intéressés par les types *« est une preuve que x est plus petit que y »*. Ici encore, le type dépend des termes x et y.

La syntaxe est un peu plus compliquée que celle de la théorie des types simples. Il y a trois constructions mutuellement dépendantes, les *contextes* (qui définissent le type des variables), les *types* et les *termes*.

Contextes, types et termes

Nous considérons trois types de jugements : « Γ ctx » qui signifie que Γ est un contexte correctement défini, « $\Gamma \vdash X$ type » qui signifie que pour le contexte correctement défini Γ, le type X est correctement défini, et enfin, le jugement « $\Gamma \vdash t : X$ » qui signifie que dans le contexte Γ, le terme t a le type X. Nous rajoutons les jugements : « $\Gamma \vdash X = Y$ type » indique que X et Y sont des types égaux par définition, « $\Gamma \vdash t = r : X$ » pour les termes égaux par définition.

Nota bene 1.7.12. Dans les calculs que nous définissons au chapitre 7, nous n'utilisons que des constructeurs de types ayant la forme :

$$x_1 : X_1, \ldots, x_n : X_n \vdash S(x_1, \ldots, x_n) \text{ type}$$

où $X_1, \ldots, X_n$ sont des types définis au préalable, et qui ne contiennent que des variables.

Ces deux faits nous permettent de disposer sans trop de problème du lemme de substitution, de la règle d'échange et de la contraction.

Note 1.7.1. Contrairement à ce que nous avons vu pour le calcul simplement typé, les variables ne sont pas typées d'avance, mais dans un contexte. On suppose que l'on dispose d'une réserve dénombrable de variables dans laquelle on peut piocher.

Définition 1.7.13 (Contextes). Les contextes sont des suites finies de paires (x,X) où x est une variable et X est un type. Nous utilisons pour les contextes la notation : $\Gamma = \{x_1 : X_1, x_2 : X_2, \cdots, x_n : X_n\}$. Le contexte vide est noté $\Diamond$, et si Γ est un contexte, $\Gamma, x : X$ désigne le contexte Γ concaténé avec $\{x : X\}$. Les règles de formation pour les contextes sont :

$$\frac{}{\Diamond \text{ ctx}} \qquad \frac{\Gamma \vdash X \text{ type}}{\Gamma, x : X \text{ ctx}} \ (*)$$

(*) x est une variable qui n'apparait pas dans Γ, une variable « fraîche ».

On introduit les règles d'égalités générales sur les types :

$$\frac{\Gamma \vdash X \text{ type}}{\Gamma \vdash X = X \text{ type}} \qquad \frac{\Gamma \vdash X = Y \text{ type}}{\Gamma \vdash Y = X \text{ type}}$$

$$\frac{\Gamma \vdash X = Y \text{ type} \qquad \Gamma \vdash Y = Z \text{ type}}{\Gamma \vdash X = Z \text{ type}}$$

et pour les termes :

$$\frac{\Gamma \vdash t : X}{\Gamma \vdash t = t : X} \qquad \frac{\Gamma \vdash t = r : X}{\Gamma \vdash r = t : X} \qquad \frac{\Gamma \vdash t = r : X \qquad \Gamma \vdash r = s : X}{\Gamma \vdash t = s : X}$$

Une règle pour les jugements de types :

$$\frac{\Gamma \vdash t : X \qquad \Gamma \vdash X = Y \text{ type}}{\Gamma \vdash t : Y}$$

Et deux règles de base, pour tout jugement $\mathcal{J}$ de type « X type », « $X = Y$ type », « $t : X$ », « $t = r : X$ »,

Projection
$$\frac{\Gamma \vdash X \text{ type}}{\Gamma, x : X \vdash x : X}$$

Affaiblissement
$$\frac{\Gamma \vdash X \text{ type} \qquad \Gamma \vdash \mathcal{J}}{\Gamma, x : X \vdash \mathcal{J}}$$

Le calcul (du chapitre 7) est alors construit pour que nous puissions obtenir les trois règles dérivées suivantes (par induction sur la construction des jugements) :

Contraction
$$\frac{\Gamma, x : X, y : X, \Delta \vdash \mathcal{J}}{\Gamma, x : X, \Delta[y \backslash x] \vdash \mathcal{J}[y \backslash x]}$$

Échange
$$\frac{\Gamma, x : X, y : Y \vdash \mathcal{J}}{\Gamma, y : Y, x : X \vdash \mathcal{J}}(*)$$

Substitution
$$\frac{\Gamma \vdash t : X \qquad \Gamma, x : X, \Delta \vdash \mathcal{J}}{\Gamma, \Delta[x \backslash t] \vdash \mathcal{J}[x \backslash t]}$$

Dans la règle d'échange, on suppose que la variable x est fraîche.

Définition des types produit et somme

Nous présentons quelques constructions usuelles sur les types et nous donnons une interprétation de ceux-ci dans la théorie des ensembles.

Définition 1.7.14 (Type produit). Le produit dépendant ou type Π correspond à la notion de produit cartésien sur une famille d'ensembles $\prod_{x \in X} Y_x$ dont les éléments sont les fonctions qui envoient chaque indice x sur un élément de Y_x. La règle d'introduction correspondante est :

$$\frac{\Gamma \vdash X \text{ type} \quad \Gamma, x : X \vdash Y \text{ type}}{\Gamma \vdash \Pi x : X.Y \text{ type}}$$

Il faut rajouter le jugement pour l'égalité :

$$\frac{\Gamma \vdash X = X' \text{ type} \quad \Gamma, x : X \vdash Y = Y' \text{ type}}{\Gamma \vdash \Pi x : X.Y = \Pi x : X'.Y' \text{ type}}$$

Définition 1.7.15. La règle d'introduction du produit est :

$$\frac{\Gamma, x : X \vdash t : Y}{\Gamma \vdash \lambda x.t : \Pi x : X.Y}\ (\lambda)$$

pour lequel nous avons l'égalité :

$$\frac{\Gamma, x : X \vdash t = t' : Y \qquad \Gamma \vdash X = X' \text{ type} \qquad \Gamma, x : X \vdash Y = Y'}{\Gamma \vdash \lambda x.t = \lambda x.t' : \Pi x : X'.Y'}\ (\lambda)$$

Définition 1.7.16. La règle d'élimination est :

$$\frac{\Gamma \vdash t : \Pi x : X.Y \quad \Gamma \vdash a : X}{\Gamma \vdash (t)a : Y[x \backslash a]}$$

Le type $Y[x \backslash a]$ est le type Y (dépendant de la variable x) dans lequel on a substitué le terme a à la place de x. Nous prenons les précautions habituelles, c'est-à-dire que la substitution doit être faite sans capture de variables.

Nous avons l'égalité :

$$\frac{\Gamma \vdash t = t' : \Pi x : X.Y \quad \Gamma \vdash a = a' : X}{\Gamma \vdash (t)a = (t')a'Y[x\backslash a]}$$

Par la suite, nous ne mettrons plus les jugements d'égalités. Ceux-ci s'obtiennent de manière analogue à ce que nous venons de voir.

Lorsque le type Y ne dépend pas de la variable x, on retrouve le schéma d'introduction du type $X \to Y$ du calcul ordinaire ; il est d'usage d'utiliser dans ce cas la notation $X \to Y$ plutôt que $\Pi x : X.Y$.

Définition 1.7.17 (Type somme). Les sommes disjointes ou type Σ correspondent aux sommes disjointes de la théorie des ensembles. $\sum_{x \in X} Y_x = \{(x,y) \mid x \in X \text{ et } y \in Y_x\}$. La règle d'introduction du type est :

$$\frac{\Gamma \vdash X \text{ type} \quad \Gamma, x : X \vdash Y \text{ type}}{\Gamma \vdash \sum x : X.Y \text{ type}}$$

Définition 1.7.18. La règle d'introduction de la somme est :

$$\frac{\begin{array}{l}\Gamma \vdash t : X \\ \Gamma \vdash r : Y[x\backslash t]\end{array}}{\Gamma \vdash \left\langle t,r \right\rangle : \Sigma x : X.Y}$$

Définition 1.7.19. Les règles d'élimination de la somme sont :

$$\frac{\Gamma \vdash t : \Sigma x : X.Y}{\Gamma \vdash \pi t : X} \qquad \frac{\Gamma \vdash t : \Sigma x : X.Y}{\Gamma \vdash \pi' t : Y[x\backslash \pi t]}$$

Pour cette règle, le lecteur notera que l'on substitue également un terme dans un type. Nous prenons les mêmes précautions que dans le cas précédent.

Lorsque le type Y ne dépend pas de X, on note « $X \times Y$ » le type « $\Sigma x : X.Y$ ».

Réductions des termes

Il y a deux règles de réductions, l'une correspond à la β-réduction du λ-calcul simplement typé et l'autre est une règle d'élimination pour

la somme.

Définition 1.7.20 (Règles de réduction standards).

$$\begin{array}{rcl} (\lambda x.t)a & \rhd_\beta & t[x\backslash a] \\ \lambda x.tx & \rhd_\beta & t \\ \pi\left\langle t,r\right\rangle & \rhd_\beta & t \\ \pi'\left\langle t,r\right\rangle & \rhd_\beta & r \\ \left\langle \pi t,\pi' t\right\rangle & \rhd_\beta & t \end{array}$$

L'égalité des termes est alors obtenue en prenant en compte la réduction β.

Chapitre 2

Éléments de théorie des catégories

Nous proposons ici une présentation de la théorie des catégories ; pour en limiter la taille, nous avons choisi de travailler avec des moyens élémentaires, et de rester le plus concret possible. Nous donnons de nombreux exemples pour illustrer les définitions. Le lecteur qui préfère un exposé plus détaillé utilisant des procédés plus élaborés sera (sans doute) satisfait par la lecture de Mac Lane [ML91] où celle de Lambek et Scott [LS86]. Il y trouvera en particulier les preuves des propositions et des théorèmes énoncés dans la section.

La théorie des catégories a montré son utilité dans de nombreux domaines de l'informatique ; elle fournit en particulier des outils concernant la sémantique du λ-calcul. C'est surtout dans ce domaine que nous en avons fait usage ; en particulier, l'utilisation des catégories nous permet de comparer différents modèles de calculs dans un même cadre. L'introduction que nous proposons est donc orientée vers ce but.

2.1 Graphes, multi-graphes et catégories

Définition 2.1.1 (Multigraphes). Un multi-graphe est un quadruplet $G = (|G|, R_G, \mathtt{dom}_G, \mathtt{cod}_G)$, où $|G|$ et R_G sont des ensembles, $\mathtt{dom}_G$

et $\mathtt{cod}_G$ deux fonctions $R_G \underset{\mathtt{cod}_G}{\overset{\mathtt{dom}_G}{\rightrightarrows}} |G|$. Nous appelons $|G|$ l'ensemble des *sommets*, et R_G l'ensemble des *arêtes* ; étant donné p dans R_G, $\mathtt{dom}_G(p)$ s'appelle le *domaine* de p et $\mathtt{cod}_G(p)$ son *codomaine*.

Dans la littérature, le mot de *graphe* est utilisé pour désigner soit les multi-graphes comme nous venons de les voir (cf. Mac Lane [ML91]) soit les multi-graphes qui ont au plus une seule arête entre deux sommets (cf Berge [Ber70]). Nous utilisons la terminologie de Berge ; si nous voulons mettre l'accent sur le fait que les graphes sont des multi-graphes particuliers, nous employons alors le terme de *graphe squelettique*.

Par $p : x - y \in G$, nous entendons que p est une arête partant de x allant vers y dans le multi-graphe G. Plus précisément, $p \in R_G$, $\mathtt{dom}_G(p) = x$ et $\mathtt{cod}_G(p) = y$. Nous omettons l'indice G quand le contexte est suffisament clair.

Un *homomorphisme* $G \to G'$ de multi-graphes est une paire $(|f|, R_f)$ de fonctions $|f| : |G| \to |G'|$, $R_f : R_G \to R_{G'}$ telle que $|f| \circ \mathtt{dom}_G = \mathtt{dom}_{G'} \circ R_f$ et $|f| \circ \mathtt{cod}_G = \mathtt{cod}_{G'} \circ R_f$.

$$\begin{array}{ccccc} |G| & \xleftarrow{\mathtt{dom}_G} & R_G & \xrightarrow{\mathtt{cod}_G} & |G| \\ {\scriptstyle |f|}\downarrow & & {\scriptstyle R_f}\downarrow & & \downarrow{\scriptstyle |f|} \\ |G'| & \xleftarrow{\mathtt{dom}_{G'}} & R_{G'} & \xrightarrow{\mathtt{cod}_{G'}} & |G'| \end{array}$$

À l'image de la notation usuelle pour les foncteurs en théorie des catégories, nous emploierons le symbole f pour désigner à la fois $|f|$ et R_f.

Définition 2.1.2. Un *chemin* dans un multi-graphe G est une suite finie d'arêtes $f_1, f_2, \ldots, f_n$ telles que pour tout $i < n$, $\mathtt{cod}(f_i) = \mathtt{dom}(f_{i+1})$.

Définition 2.1.3. Une petite catégorie est un multi-graphe $\mathcal{C}$ muni d'une opération binaire $\circ$, dite de composition, sur les arêtes, et, pour chaque élément a de $|\mathcal{C}|$, d'une arête notée 1_a, appelée identité de a. Les sommets d'une catégorie sont appelés *objets* et ses arêtes, *mor-*

phismes/flèches. Nous notons par $f : a \to b$ le fait qu'un morphisme ait a pour domaine et b pour codomaine.

L'opération binaire doit de plus vérifier les équations suivantes:

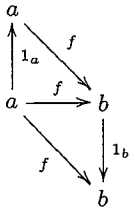
(i) $f \circ 1_a = f$,
(ii) $1_b \circ f = f$,
(iii) $(f \circ g) \circ h = f \circ (g \circ h)$

pour tous $f : a \to b$, $g : b \to c$, $h : c \to d$.

Note 2.1.1.

- L'écriture $a \in \mathcal{C}$ pour dire que a est un objet de la catégorie $\mathcal{C}$ est un abus que nous employons par commodité. Il en est de même pour $f \in \mathcal{C}$, f étant un morphisme.
- $\mathrm{Hom}_{\mathcal{C}}(a,b)$ désigne l'ensemble des morphismes qui ont pour domaine a et pour codomaine b dans la catégorie $\mathcal{C}$. L'autre notation standard pour désigner l'ensemble des morphismes de a vers b dans la catégorie $\mathcal{C}$ est $\mathcal{C}(a,b)$.

Comme la composition est associative, le parenthésage est inutile. Par conséquent, tout morphisme obtenu par composition peut être représenté par un chemin dans un multi-graphe, chemin dont les arêtes sont les morphismes composés. Par exemple, le chemin composé des arêtes $a \xrightarrow{f} b \xrightarrow{g} c \xrightarrow{h} d$ représente les deux morphismes composés du point (iii) des propriétés de la composition. Par la suite, nous identifions un chemin à la composition des morphismes composés. Les égalités se décrivent alors comme des *diagrammes* pour lesquels tout chemin est égal à tout autre ayant même origine et même but. Par exemple, les deux premières propriétés de la composition se résument par le diagramme :



Définition 2.1.4. Nous avons également besoin de « grandes » catégories : il s'agit des catégories pour lesquelles la collection des objets (et par là même, la collection des morphismes) est une classe. Lorsque la collection d'objet est une classe mais que la collection des morphismes *entre deux objets* est un ensemble, on parle de catégorie *localement petite.*

Définition 2.1.5. Étant donné une catégorie $\mathcal{C}$, une *sous-catégorie* $\mathcal{D}$ de $\mathcal{C}$ est une collection $|\mathcal{D}|$ d'objets de $\mathcal{C}$ et une collection $R_{\mathcal{D}}$ de morphismes de $\mathcal{C}$ telle que :

(i) si $f : a \to b \in R_{\mathcal{D}}$, alors $a \in |\mathcal{D}|$ et $b \in |\mathcal{D}|$,

(ii) si $a \in |\mathcal{D}|$, alors $1_a \in R_{\mathcal{D}}$,

(iii) si $f : a \to b \in R_{\mathcal{D}}$ et $g : b \to c \in R_{\mathcal{D}}$ alors $g \circ f \in R_{\mathcal{D}}$.

Les conditions (i-iii) assurent que $\mathcal{D}$ constitue une catégorie.

On dit d'une sous-catégorie qu'elle est *pleine* si $\mathrm{Hom}_{\mathcal{D}}(a,b) = \mathrm{Hom}_{\mathcal{C}}(a,b)$ pour tous $a,b \in \mathcal{D}$

Exemple 2.1.6. Les ensembles constituent une catégorie (localement petite) notée **Ens** définie comme suit. Les objets de **Ens** sont les ensembles et les morphismes sont les fonctions entre ces ensembles. L'opération de composition est la composition ordinaire (d'où la notation !) et les identités sont les fonctions identité des ensembles.

Exemple 2.1.7. Tout ensemble pré-ordonné $(X, \leq)$ constitue une catégorie. Les objets sont les éléments de X et les morphismes, les couples d'éléments (x,y) tels que $x \leq y$. Les identités correspondent à la réflexivité de l'ordre sur X et la composition à la transitivité de l'ordre.

Exemple 2.1.8. Dans le même ordre d'idée, on peut définir une catégorie à partir d'un multi-graphe. Les objets sont les sommets du multi-graphe et les arêtes sont les chemins entre ces sommets. La concaténation de chemins donnant la composition. De manière plus savante, cette catégorie est la catégorie librement engendrée par le multi-graphe.

Exemple 2.1.9. Un monoïde $(M,e,\cdot)$ définit une catégorie $\mathcal{M} : |\mathcal{M}| = \{\star\}$ (un ensemble à un seul élément choisi arbitrairement) et $R_{\mathcal{M}} = M$.

L'identité de $\star$ est e, l'élément neutre du monoïde, et la composition des morphismes est donnée par l'opération « $\cdot$ ».

Exemple 2.1.10. Les objets de la catégorie des monoïdes **Mon** sont les monoïdes, les morphismes sont les morphismes de monoïdes ; la composition et les identités sont héritées des ensembles. De même, la catégorie **Ann** des anneaux est composée des anneaux et des morphismes d'anneaux.

Exemple 2.1.11. La catégorie des multi-graphes est notée **MGr**. Les objets de **MGr** sont les multi-graphes et les morphismes sont les homomorphismes de multi-graphes. Les identités et la composition sont héritées de celles des ensembles.

De même, nous pouvons définir la sous-catégorie des graphes **Gr** : les objets sont les multi-graphes squelettiques et les morphismes sont les morphismes de graphes. Notons que **Gr** est une sous-catégorie pleine de **MGr**.

Exemple 2.1.12. En notant que les relations binaires s'identifient aux graphes squelettiques[14], on peut considérer la sous-catégorie (de la catégorie **Gr**) des pré-ordres (**Ord**), celle des ordres partiels, ou encore celle des ordres totaux. Toutes ces catégories sont des sous-catégories pleines de **Gr**. Ce sont toutes des sous-catégories pleines.

Définition 2.1.13. Soit une catégorie $\mathcal{C}$, nous considérons alors la catégorie $\mathcal{C}^{\mathrm{op}}$:

- $|\mathcal{C}^{\mathrm{op}}| = |\mathcal{C}|$,
- $f \in \mathrm{Hom}_{\mathcal{C}^{\mathrm{op}}}(a,b)$ ssi $f \in \mathrm{Hom}_{\mathcal{C}}(b,a)$. En d'autres termes, les morphismes de $\mathcal{C}^{\mathrm{op}}$ sont les morphismes de $\mathcal{C}$ mais dont le domaine et le codomaine ont été inversés : $\mathtt{cod}_{\mathcal{C}^{\mathrm{op}}}(f) = \mathtt{dom}_{\mathcal{C}}(f)$ et $\mathtt{dom}_{\mathcal{C}^{\mathrm{op}}}(f) = \mathtt{cod}_{\mathcal{C}}(f)$. Les identités de $\mathcal{C}^{\mathrm{op}}$ sont les identités de $\mathcal{C}$ et la composition est la composition de $\mathcal{C}$.

Cette construction nous sert lorsque nous considérons la notion de foncteur contravariant. Nous nous en servons également pour la

14. On peut donner un sens catégorique à cette identification, mais cela nous mènerait à des considérations catégoriques « l'équivalence de catégorie » inutiles pour la suite.

dualité. Informellement, la construction duale d'une construction F dans la catégorie $\mathcal{C}$ est la construction F mais dans la catégorie $\mathcal{C}^{\mathrm{op}}$.

2.2 Propriétés et constructions sur les objets et les morphismes

Définition 2.2.1. Un morphisme $f : a \to b$ est qualifié d'iso-(morphisme) s'il existe un morphisme $g : b \to a$ tel que $f \circ g = 1_b$ et $g \circ f = 1_a$.

On remarquera que g est également un isomorphisme et qu'il est déterminé de façon unique par f.

Enfin, par extension, deux objets a et b sont isomorphes s'il existe un isomorphisme $f : a \to b$ (où de manière équivalente un isomorphisme $g : b \to a$). On note ceci $a \simeq b$ ou bien $f : a \simeq b : g$ si l'on veut préciser quels sont les (iso)morphismes.

Exemple 2.2.2. Dans **Ens**, les isomorphismes sont les fonctions bijectives.

Définition 2.2.3 (Objet terminal). Un objet t d'une catégorie $\mathcal{C}$ est terminal si pour tout objet $a \in \mathcal{C}$, il existe un unique morphisme $!_a : a \to t$. Cette propriété est appelée la propriété universelle des objets terminaux.

Proposition 2.2.4. Deux objets terminaux sont isomorphes, de plus, l'isomorphisme entre deux objets terminaux est unique.

Démonstration. Soient t et t' deux objets terminaux. En raison de la propriété des terminaux, il existe deux morphismes uniques $t \underset{b}{\overset{a}{\rightleftarrows}} t'$. Il existe également un morphisme unique $t \to t$, ce ne peut être que $1_t : t \to t$ (il en est de même pour t'). Par conséquent, $a \circ b = 1_{t'}$ et $b \circ a = 1_t$, et ces deux morphismes sont uniques. □

Définition 2.2.5 (Objet initial). Un objet initial i est un objet tel que pour tout autre objet a de $\mathcal{C}$, il y a un unique morphisme $i_a : i \to a$. Deux objets initiaux sont isomorphes à un unique isomorphisme

près. Observons que la notion d'objet initial est duale de celle d'objet terminal.

Exemple 2.2.6. Dans **Ens**, tout singleton est terminal (et tous les singletons sont isomorphes). L'objet initial est $\emptyset$.

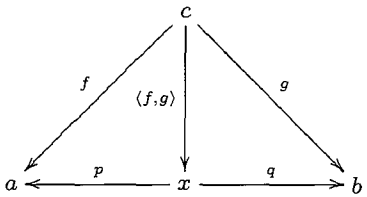
Exemple 2.2.7. Dans **MGr**, l'objet terminal est le graphe composé d'un unique sommet et d'une unique arête. L'objet initial de la catégorie est le graphe composé d'aucun sommet (et d'aucune arête).

Exemple 2.2.8. L'objet initial des monoïdes **Mon** est le monoïde réduit à un élément $(\{e\},e,.)$. C'est également l'objet terminal. L'objet initial des anneaux **Ann** est l'anneau des entiers relatifs $\mathbb{Z}$, et cet anneau est également l'objet terminal de **Ann**.

2.2.1 Produit catégorique

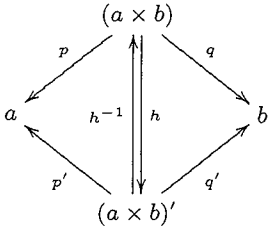
Définition 2.2.9. Étant donné deux objets a et b d'une catégorie $\mathcal{C}$, un produit de a et b est un triplet (x,p,q) tel que :

- x est un objet de $\mathcal{C}$,
- p et q sont deux morphismes appelés *projection* avec $p : x \to a$ et $q : x \to b$,
- pour tout diagramme $a \xleftarrow{f} c \xrightarrow{g} b$, il existe un unique morphisme h tel que le diagramme commute (on appelle cette propriété la propriété universelle du produit) :

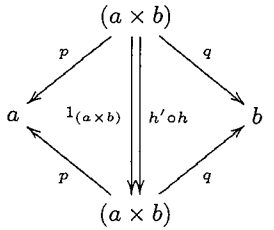


La notation que nous emploierons en général pour désigner un produit de a et b est $(a \times b, p_{a,b}, q_{a,b})$. Nous supprimons les indices lorsque le contexte est clair. De même, on note $\langle f,g \rangle$ l'unique morphisme h provenant des morphismes f et g.

Proposition 2.2.10. Soient a et b deux objets. Supposons que nous ayons deux produits $((a \times b),p,q)$ et $((a \times b)',p',q')$. Alors $(a \times b)$ et $(a \times b)'$ sont isomorphes, et il y a un unique isomorphisme h qui fait commuter le diagramme suivant :



Démonstration. Comme $((a \times b),p,q)$ est un produit, d'après le diagramme $a \xleftarrow{p'} (a \times b)' \xrightarrow{q'} b$, il existe un unique morphisme $h : (a \times b) \to (a \times b)'$ tel que $p' \circ h = p$ et $q' \circ h = q$. De manière analogue, il existe un morphisme $h' : (a \times b)' \to (a \times b)$ tel que $p \circ h' = p'$ et $q \circ h' = q'$. On dispose donc de deux morphismes qui font commuter le diagramme :



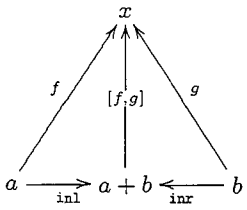
Par unicité, $h' \circ h = 1_{(a\times b)}$. Par symétrie, $h \circ h' = 1_{(a\times b)'}$. □

Exemple 2.2.11. Dans la catégorie des ensembles, un produit catégorique de a et b est le produit cartésien $a \times b$ des ensembles a et b. Les deux projections sont alors les projections usuelles.

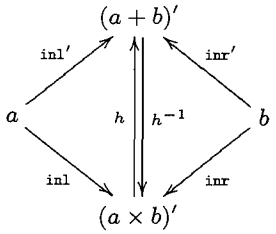
Exemple 2.2.12. Dans un ordre vu comme une catégorie, le produit de a et b est la borne inférieure de a et b (dans ce cas le produit est déterminé de manière unique).

Exemple 2.2.13. Pour les structures algébriques ordinaires comme les monoïdes, ou les anneaux, le produit s'obtient généralement à l'aide du produit des ensembles sous-jacent.

Définition 2.2.14 (Coproduit). La construction duale du produit s'appelle le *coproduit*. Étant donné deux objets a et b, un coproduit est la donnée d'un diagramme $a \xrightarrow{\texttt{inl}} a + b \xleftarrow{\texttt{inr}} b$ tel que pour tout autre diagramme $a \xrightarrow{f} x \xleftarrow{g} b$, il existe un unique morphisme $[f,g]$ qui fait commuter le diagramme :



De la même manière que le produit, le coproduit bénéficie d'une propriété universelle. Tous les coproduits de deux objets a et b sont isomorphes et il y a un unique isomorphisme qui fait commuter le diagramme :



Exemple 2.2.15. Dans les ensembles, le coproduit de X et Y est la somme disjointe $X + Y$. Les deux morphismes `inl` et `inr` sont les injections canoniques.

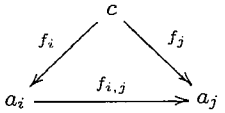
Remarque 2.2.16. Nous avons quatre exemples de propriétés universelles :

- objet initial, coproduit,
- objet terminal, produit,

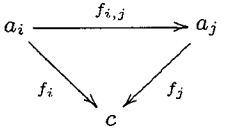
En général, les constructions universelles peuvent se décrire comme un objet initial dans une catégorie bien choisie.

2.2.2 Limites, colimites

Définition 2.2.17 (Cône, Cocône). Soit une catégorie $\mathcal{C}$. Étant donné un diagramme $\Delta = (A,F)$, c'est-à-dire une famille $A = (a_i)_{i\in I}$ d'objets de la catégorie $\mathcal{C}$ et une famille $F = (f_{i,j})_{i,j\in I}$ de morphismes entre ces objets[15], on appelle cône la donnée d'un objet c et d'une famille de morphismes $(f_i : c \to a_i)_{i\in I}$ telle que pour tout morphisme $f_{i,j} \in F$, on ait l'égalité $f_{i,j} \circ f_i = f_j$.

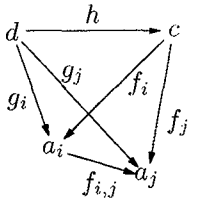


Un cocône est la donnée d'un objet c et d'une famille de morphismes $(f_i : a_i \to c)_{i\in I}$ telle que pour tout morphisme $f_{i,j} \in F$, on ait l'égalité $f_j \circ f_{i,j} = f_i$.

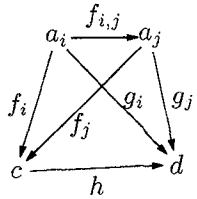


15. Pour la simplicité de l'exposé, nous considérons ici une notion restreinte de diagramme, de manière plus générale, un diagramme s'exprime en termes de foncteur (cf [ML91] p. 71).

Définition 2.2.18 (Limites, Colimites). Étant donné une catégorie $\mathcal{C}$ et un diagramme $\Delta = (A,F)$, une limite est un cône $(c,(f_i)_{i\in I})$ tel que pour tout autre cône $(d,(g_i)_{i\in I})$, il existe un *unique* morphisme $h : d \to c$ tel que pour tout $i \in I$, on ait l'égalité $g_i = f_i \circ h$. Deux limites pour un diagramme donné sont isomorphes, et l'isomorphisme (qui fait commuter le diagramme qui suit) est unique. Ceci est un autre exemple de propriété universelle.



Une colimite est un cocône $(c,(f_i)_{i\in I})$ tel que pour tout autre cocône $(d,(g_i)_{i\in I})$, il existe un *unique* morphisme $h : c \to d$ tel que pour tout morphisme $i \in I$, on ait l'égalité $g_i = h \circ f_i$. Nous appelons ce morphisme l'*épine*. Deux cocônes sont également isomorphes, et l'isomorphisme qui fait commuter le diagramme qui suit est unique.



Notons que les deux notions de cône/limite et de cocône/colimites sont en fait duales.

Exemple 2.2.19. La terminologie de limite (respectivement colimite) remplace la terminologie traditionnelle de limite projective (respectivement inductive). Notons toutefois que dans le cas des limites projectives (et inductives), on suppose que l'ensemble des index I est ordonné par un ordre $<$ dirigé : c'est-à-dire que pour tous x et y, il existe z tel que $x < z$ et $y < z$.

Exemple 2.2.20. Étant donné un ordre $(X, \leq)$ vu comme une catégorie, les limites s'interprètent comme des bornes inférieures et les colimites comme des bornes supérieures.

Exemple 2.2.21. Étant donné deux objets a et b, un produit de ces deux objets $(a \times b, p, q)$ est une limite du diagramme $A = \{a,b\}$, $F = \emptyset$. Le coproduit est la colimite du même diagramme.

Exemple 2.2.22. Un produit fibré (par la suite, nous utilisons plutôt le terme pull-back) est la limite pour un diagramme de la forme $a \xrightarrow{f} c \xleftarrow{g} b$.

Dans le cas des ensembles, on a une description directe du pull-back $(a \times_c b, p, q)$:

$$\begin{array}{ccc} a \times_c b & \xrightarrow{p} & a \\ {\scriptstyle q}\downarrow & & \downarrow{\scriptstyle f} \\ b & \xrightarrow[g]{} & c \end{array}$$

$a \times_c b = \{(x,y) \in a \times b \mid f(x) = g(y)\}$. Les deux fonctions p et q sont les projections sur a et b.

Exemple 2.2.23. Nous considérons également les sommes amalgamées (par la suite, les pushouts) sont les colimites pour les diagrammes de la forme $a \xleftarrow{f} c \xrightarrow{g} b$.

Dans le cas des ensembles, nous avons une description directe du pushout $(a +_c b, i, j)$:

$a +_c b = a + b/\simeq$ où $\simeq$ est la plus petite relation d'équivalence sur $a + b$ qui contient $f(z) \simeq g(z)$ pour tout $z \in c$.

Exemple 2.2.24. Enfin, un diagramme qui a la forme $a_0 \xrightarrow{f_0} a_1 \xrightarrow{f_1} a_2 \cdots$ est appelé une ω-chaîne. Un exemple typique de colimite de ω-chaîne est celui qu'on trouve en sémantique des langages de programmation lorsque l'on considère des points fixes, comme dans la

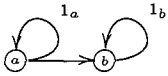
définition traditionnelle (cf Gunter [Gun92]) : $[\![\mu f.M]\!]_\rho = \bigsqcup_{i=0}^{\infty} F_i$ avec

$$\begin{cases} F_0 &= \bot \\ F_{n+1} &= [\![M]\!]_{\rho[f \mapsto F_n]} \end{cases}$$

Exemple 2.2.25. Étant donné un ensemble E et une suite $E_1 \subseteq E_2 \subseteq \cdots$ de parties de E, l'ensemble $\bigcup_{i=1}^{\infty} E_i$ est la colimite de la chaîne $(E_i)_i$. Le morphisme $E_n \to \bigcup_{i=1}^{\infty} E_i$ est l'inclusion canonique de E_n dans $\bigcup_{i=1}^{\infty} E_i$.

2.3 Construction de catégories

Définition 2.3.1.

- **0** est la catégorie vide : sans objets ni morphismes ;
- **1** est la catégorie ne contenant qu'un seul objet 0  ;
- **2** est la catégorie constituée de deux objets a et b, et d'une (unique) flèche entre a et b  .

Définition 2.3.2. Étant donné deux catégories $\mathcal{B}$ et $\mathcal{C}$, on considère la catégorie produit $\mathcal{B} \times \mathcal{C}$ définie comme suit. Un objet de $\mathcal{B} \times \mathcal{C}$ est une paire (b,c) d'un objet $b \in \mathcal{B}$ et d'un objet $c \in \mathcal{C}$. Un morphisme $(b,c) \to (b',c')$ dans $\mathcal{B} \times \mathcal{C}$ est une paire (f,g) ou $f : b \to b'$ est un morphisme de $\mathcal{B}$ et $g : c \to c'$ est un morphisme de $\mathcal{C}$. La composition de deux tels morphismes :

$$(b,c) \xrightarrow{(f,g)} (b',c') \xrightarrow{(f',g')} (b'',c'')$$

est définie par $(f',g') \circ (f,g) = (f' \circ f, g' \circ g)$. Les identités sont définies par $1_{(b,c)} = (1_b, 1_c)$.

2.4 Foncteurs et transformations naturelles

Définition 2.4.1 (Foncteurs). Un foncteur entre les catégories $\mathcal{C}$ et $\mathcal{D}$ est un homomorphisme de multi-graphes qui conserve les identités et la composition. Ce concept joue le même rôle que celui des homomorphismes de groupes, d'anneaux, etc : il conserve la structure de catégorie.

En d'autres termes, nous avons deux fonctions $|F| : |\mathcal{C}| \to |\mathcal{D}|$ et $R_F : R_\mathcal{C} \to R_\mathcal{D}$ telles que

(i) pour tout morphisme $a \xrightarrow{f} b$ de $\mathcal{C}$, $|F|(a) \xrightarrow{R_F(f)} |F|(b)$ est un morphisme de $\mathcal{D}$,

(ii) pour tout objet $a \in \mathcal{C}$, $R_F(1_a) = 1_{|F|(a)}$,

(iii) et pour tout diagramme $a \xrightarrow{f} b \xrightarrow{g} c$, on a l'égalité $R_F(g) \circ R_F(f) = R_F(g \circ f)$.

Note 2.4.1. Un peu de vocabulaire de la théorie des catégories.

- Par un abus de langage, on dit qu'un foncteur $F : \mathcal{C}^{\mathrm{op}} \to \mathcal{D}$ est *contravariant de $\mathcal{C}$ vers $\mathcal{D}$*. Dans le cas usuel $F : \mathcal{C} \to \mathcal{D}$, on dit que le foncteur est *covariant de $\mathcal{C}$ vers $\mathcal{D}$*.
- Un *endofoncteur* est un foncteur dont la catégorie d'arrivée est celle de départ.
- Un *bifoncteur* désigne un foncteur défini sur le produit de deux catégories : $F : \mathcal{C} \times \mathcal{D} \to \mathcal{E}$. Dans le cas où F est un tel bifoncteur, pour tout $d \in \mathcal{D}$, on définit le foncteur $F(-,d)$:

$$\begin{array}{rrcl} F(-,d) : & \mathcal{C} & \to & \mathcal{E} \\ & c & \mapsto & F(c,d) \\ & f : c \to c' & \mapsto & F(f,1_d) \end{array}$$

De même pour tout $c \in \mathcal{C}$, on définit le foncteur $F(c,-) : \mathcal{D} \to \mathcal{E}$:

$$\begin{array}{rrcl} F(c,-) : & \mathcal{D} & \to & \mathcal{E} \\ & d & \mapsto & F(c,d) \\ & g : d \to d' & \mapsto & F(1_c,g) \end{array}$$

Exemple 2.4.2. Étant donné deux ordres $(X, \leq_X)$ et $(Y, \leq_Y)$ vus comme des catégories, un foncteur $F : (X, \leq_X) \to (Y, \leq_Y)$ est une fonction monotone de X vers Y.

Exemple 2.4.3. Étant donné deux monoïdes $(M,e,\cdot)$ et $(M',e',\cdot')$ vus comme des catégories, un foncteur $F : (M,e,\cdot) \to (M',e',\cdot')$ est un morphisme de monoïde.

Exemple 2.4.4. Dans une catégorie $\mathcal{C}$, la donnée d'un diagramme produit $(a \times c, p_{a,c}, q_{a,c})$ pour toute paire d'objets a et c peut être étendue en un bifoncteur $\times : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$. Soient deux morphismes $f : a \to b$ et $g : c \to d$, on définit $f \times g = \langle f \circ p_{a,c}, g \circ q_{a,c} \rangle : (a \times c) \to (b \times d)$ comme l'unique morphisme qui fait commuter le diagramme :

$$\begin{array}{ccccc} a & \xleftarrow{p_{a,c}} & a \times c & \xrightarrow{q_{a,c}} & c \\ \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle f \times g} & & \downarrow{\scriptstyle g} \\ b & \xleftarrow[p_{b,d}]{} & b \times d & \xrightarrow[q_{b,d}]{} & d \end{array}$$

Lorsque nous parlons dans la suite *du* produit dans une catégorie, nous désignons par là le foncteur issu d'un certain *choix* de produits $(a \times b, p_{a,b}, q_{a,b})$ pour toute paire d'objets a et b.

Définition 2.4.5 (Cat, la catégorie des petites catégories). **Cat** désigne la catégorie dont les objets sont les petites catégories et les morphismes, les foncteurs entre ces dernières. La composition et les identités sont héritées de la composition et des identités des morphismes de multi-graphes.

Le produit de catégories est un produit catégorique dans la catégorie **Cat** (cf, par exemple, Mac Lane [ML91]). La projection $\mathcal{B} \times \mathcal{C} \to \mathcal{B}$ est donnée par la projection des ensembles sous-jacents.

Exemple 2.4.6 (Foncteur constant). Étant donné une catégorie $\mathcal{C}$ et un objet $a \in \mathcal{C}$, pour toute autre catégorie $\mathcal{B}$, on construit le foncteur constant $C_a^{\mathcal{B}} : \mathcal{B} \to \mathcal{C}$ défini par :

$$\begin{cases} x \in |\mathcal{B}| & \mapsto & a \\ f \in R_{\mathcal{B}} & \mapsto & 1_a \end{cases}$$

Nous utiliserons ce foncteur lorsque nous ferons de l'algèbre universelle. Nous le noterons a, le contexte indiquant alors que nous désignons le foncteur constant et non l'objet.

Exemple 2.4.7 (Lift). Dans la catégorie des ordres, le lift d'un ordre est l'opération qui consiste à rajouter un plus petit élément. Cette opération peut être prolongée en un foncteur. Plus formellement, le lift d'un ensemble ordonné $(X, \leq_X)$ est l'ensemble $X_\perp = X + \{\perp\}$ ordonné par la relation d'ordre $x \leq_{X_\perp} y$ si $x = \perp$ ou si $x \leq_X y$. Étant donné une fonction monotone $f : X \to Y$, on considère son prolongement $f_\perp : X_\perp \to Y_\perp$,

$$\begin{array}{rcl} x \in X & \mapsto & f(x) \\ \perp & \mapsto & \perp \end{array}$$

Ceci définit un (endo)foncteur (que nous utiliserons pour construire ω) :

$$\begin{array}{rcl} \mathbf{Ord} & \to & \mathbf{Ord} \\ X & \mapsto & X_\perp \\ f & \mapsto & f_\perp \end{array}$$

Exemple 2.4.8 (Hom-Foncteurs). Étant donné une catégorie localement petite $\mathcal{C}$ et un objet $d \in \mathcal{C}$, nous considérons le foncteur $\mathcal{C}(-,d) : \mathcal{C}^{\mathrm{op}} \to \mathbf{Ens}$ (contravariant de $\mathcal{C}$ vers **Ens**) défini par :

$$\mathcal{C}(-,d)(a) = \mathcal{C}(a,d)$$

et pour tout morphisme $f : a \to b$,

$$\begin{array}{rcccc} \mathcal{C}(-,d)(f) : & \mathcal{C}(b,d) & \to & \mathcal{C}(a,d) \\ & g & \mapsto & g \circ f \end{array}$$

Pour simplifier, nous notons $\mathcal{C}(f,d)$ le morphisme $\mathcal{C}(-,d)(f)$.

De même, nous considérons le foncteur $\mathcal{C}(c,-) : \mathcal{C} \to \mathbf{Ens}$ défini par :

$$\mathcal{C}(c,-)(a) = \mathcal{C}(c,a)$$

et pour tout morphisme $f : a \to b$,

$$\begin{array}{rcccc} \mathcal{C}(c,-)(f) : & \mathcal{C}(c,a) & \to & \mathcal{C}(c,b) \\ & g & \mapsto & f \circ g \end{array}$$

De même, nous notons $\mathcal{C}(c,f)$ le morphisme $\mathcal{C}(c,-)(f)$.

La combinaisons de ces deux définition en fait un bifoncteur $\mathcal{C}(-,-) : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathbf{Ens}$, contravariant en la première variable et covariant en la seconde, appelé foncteur d'homomorphisme (externe).

Définition 2.4.9 (Transformations naturelles). Étant donné deux foncteurs $F, G : \mathcal{C} \to \mathcal{D}$, une transformation naturelle $\alpha : F \to G$ est la donnée pour chaque objet a d'un morphisme $\alpha_a : Fa \to Ga$ tel que pour tout morphisme $f : a \to b \in \mathcal{C}$, les diagrammes commutent :

$$\begin{array}{ccc} Fa & \xrightarrow{\alpha_a} & Ga \\ {\scriptstyle Ff}\downarrow & & \downarrow{\scriptstyle Gf} \\ Fb & \xrightarrow[\alpha_b]{} & Gb \end{array}$$

La notation usuelle pour les transformations naturelles est $\mathcal{C} \overset{F}{\underset{G}{\overset{\alpha\downarrow}{\rightrightarrows}}} \mathcal{D}$.

On dit qu'une transformation naturelle est un *isomorphisme naturel* lorsque toutes ses composantes sont des isomorphismes.

Nous mentionnons ici un résultat et deux définitions qui, si on ne s'en servira pas de manière explicite, sous-tendent certains résultats dont nous avons besoin.

Remarque 2.4.10 (Composition verticale). Les transformations naturelles peuvent être composées : soient trois foncteurs $F,G,H : \mathcal{C} \to \mathcal{D}$. Soient deux transformations naturelles $\alpha : F \to G$ et $\beta : G \to H$. Nous avons alors une transformation naturelle $\beta \circ \alpha : F \to H$ définie par $(\beta \circ \alpha)_a = \beta_a \circ \alpha_a$.

Exemple 2.4.11. Pour tout foncteur $F : \mathcal{C} \to \mathcal{D}$, nous avons une transformation naturelle $1_F : F \to F$ définie par :

$$(1_F)_a = 1_{Fa}$$

Définition 2.4.12 (Catégories de foncteurs). D'après la remarque et l'exemple qui précèdent, nous définissons pour toutes catégories $\mathcal{C}$ et $\mathcal{D}$ la catégorie $[\mathcal{C} \Rightarrow \mathcal{D}]$: les objets de $[\mathcal{C} \Rightarrow \mathcal{D}]$ sont les foncteurs

de $\mathcal{C}$ vers $\mathcal{D}$, les morphismes sont les transformations naturelles entre les foncteurs.

Note 2.4.2. Il y a ici un problème de « taille ». Nous ne considérons que les cas où $\mathcal{C}$ est une petite catégorie. Dans le cas où la catégorie $\mathcal{D}$ est petite, la catégorie $[\mathcal{C} \Rightarrow \mathcal{D}]$ est petite. Dans le cas où la catégorie $\mathcal{D}$ est localement petite, la catégorie $[\mathcal{C} \Rightarrow \mathcal{D}]$ est localement petite.

Remarque 2.4.13. Un cas important de catégorie de foncteurs est celui des catégories de foncteurs de la forme $[\mathcal{C} \Rightarrow \mathbf{Ens}]$ où $\mathcal{C}$ est petite. Ces catégories qualifiées de catégories de pré-faisceaux bénéficient de nombreuses propriétés, comme d'être cartésiennes fermées. Nous verrons des exemples.

Théorème 2.4.13.1 (Lemme de Yoneda).
Soit un foncteur $F : \mathcal{C} \to \mathbf{Ens}$. Pour tout objet $C \in \mathcal{C}$, l'ensemble des transformations naturelles $Hom(C,-) \to F$ est en correspondance bijective avec $F(C)$.

Il existe une deuxième version du lemme : *si $F : \mathcal{C}^{op} \to \mathbf{Ens}$ est contravariant, l'ensemble des transformations naturelles $Hom(-,C) \to F$ est en correspondance bijective avec $F(C)$.*

Définition 2.4.14 (Adjonction). Étant donné deux catégories $\mathcal{A}$ et $\mathcal{X}$ et deux foncteurs $F : \mathcal{A} \to \mathcal{X}$, $G : \mathcal{X} \to \mathcal{A}$, une adjonction est la donnée d'un isomorphisme naturel :

$$\alpha : \mathrm{Hom}_{\mathcal{X}}(F(-),-) \simeq \mathrm{Hom}_{\mathcal{A}}(-,G(-))$$

On dit alors que F est un adjoint à gauche de G, et inversement G est un adjoint à droite de F.

Exemple 2.4.15 (Foncteur diagonal). Étant donné une catégorie $\mathcal{C}$, nous définissons le foncteur diagonal $\Delta : \mathcal{C} \to \mathcal{C} \times \mathcal{C}$ comme suit :

$$\begin{array}{rccl} \Delta : & \mathcal{C} & \to & \mathcal{C} \times \mathcal{C} \\ & a & \mapsto & (a,a) \\ f : a \to b & & \mapsto & (f,f) : (a,a) \to (b,b) \end{array}$$

Supposons que la catégorie $\mathcal{C}$ dispose d'un bifoncteur produit. On a alors un isomorphisme :

$$\alpha_{c,(a,b)} : \mathrm{Hom}_{\mathcal{C}\times\mathcal{C}}(\Delta c,(a,b)) \simeq \mathrm{Hom}_{\mathcal{C}}(c,a \times b)$$

Il suffit de définir $\alpha_{c,(a,b)}(f,g) = \langle f,g\rangle$. Inversement, étant donné $f : c \to a \times b$, on lui associe le morphisme $(p_{a,b} \circ f, q_{a,b} \circ f) : (c,c) \to (a,b)$.

2.5 Catégories monoïdales et cartésiennes fermées

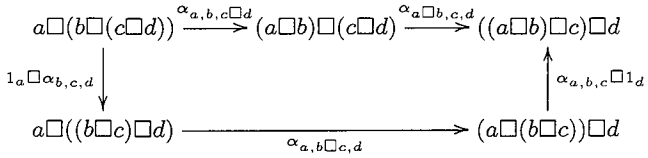
L'un des emplois de la théorie des catégories est de fournir des sémantiques pour les langages de programmation. Le concept correspondant au λ-calcul (simplement typé) est celui de catégorie cartésienne fermée. Mais nous commençons par la notion de catégorie monoïdale fermée qui donne une sémantique au λ-calcul linéaire.

Pour interpréter les contextes, nous avons besoin d'une notion catégorique qui permet de considérer des collections finies d'objets. On utilise pour cela un bifoncteur. Le problème est que celui-ci associe les éléments deux par deux. Or, pour que l'on puisse utiliser la collection comme un multi-ensemble, nous avons besoin de propriétés supplémentaires sur le bifoncteur, comme l'associativité ou la commutativité. La notion de catégorie monoïdale symétrique dégage les constructions catégoriques dont on a besoin dans ce contexte.

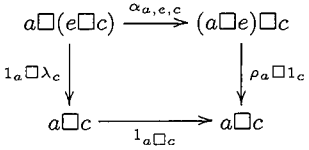
Une structure monoïdale sur une catégorie est la donnée d'un bifoncteur $\Box$ associatif appelé habituellement *tenseur*. Comme en règle générale, nous n'avons pas l'associativité à strictement parler mais une associativité *à un certain isomorphisme naturel près*, il nous faut une notion moins restrictive que celle de structure monoïdale comme nous venons de le voir. Néanmoins, il est nécessaire que l'utilisation de la transitivité soit « transparente » : c'est le théorème de cohérence que nous énonçons ci-dessous.

Définition 2.5.1. Une structure monoïdale relâchée est une catégorie $\mathcal{C}$ munie d'un bifoncteur $\Box : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, un objet $e \in \mathcal{C}$ et de trois isomorphismes naturels α, λ et ρ tels que

- $\alpha_{a,b,c} : a\Box(b\Box c) \simeq (a\Box b)\Box c$,
- le diagramme suivant commute :



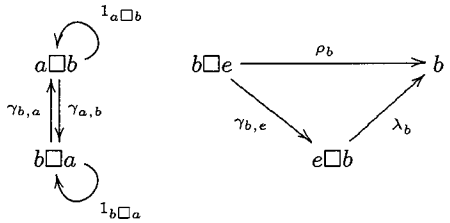
- $\lambda_a : e\Box a \simeq a$,
- $\rho_a : a\Box e \simeq a$,
- le diagramme commute :

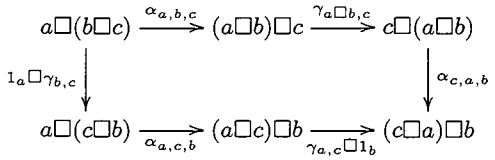


Définition 2.5.2. Une catégorie monoïdale est *symétrique* si elle est équipée d'isomorphismes

$$\gamma_{a,b} : a\Box b \simeq b\Box a$$

naturels en a et b et tels que les diagrammes suivants commutent.

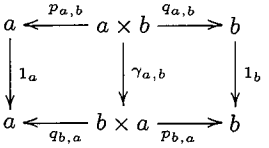




Exemple 2.5.3. Supposons qu'une catégorie $\mathcal{C}$ admette un produit $\times$ et un objet terminal 1. On dit que la catégorie est *cartésienne* [16]. Une telle catégorie peut être munie d'une structure monoïdale. On construit les transformations naturelles de la manière suivante.

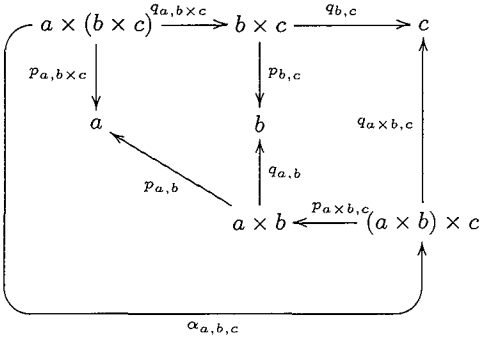
on définit $\lambda_a = p_{a,1}$, λ_a a comme inverse $\langle 1_a, !_a \rangle$. On définit $\rho_a = q_{a,1}$, il a pour inverse $\langle !_a, 1_a \rangle$.

De même, on définit $\gamma_{a,b}$ comme étant l'unique morphisme qui fait commuter le diagramme :



Enfin, on définit le morphisme $\alpha_{a,b,c}$ comme étant l'unique morphisme qui fait commuter le diagramme :

16. Certains auteurs définissent comme cartésiennes les catégories admettant les limites finies.



Remarque 2.5.4. Notons que la présence du produit [17] est une propriété de la catégorie, alors que la structure monoïdale est une *structure additionnelle* sur la catégorie.

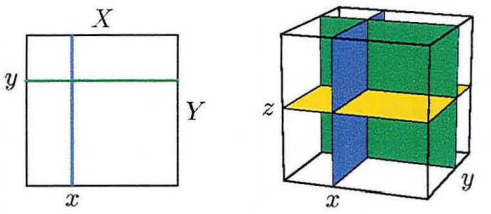
Exemple 2.5.5. De manière duale, si une catégorie $\mathcal{C}$ admet un coproduit $+$ et un objet initial 0, la propriété universelle du coproduit nous donne des isomorphismes naturels (en toutes leurs variables) $a+(b+c) \simeq (a+b)+c$, $0+b \simeq b$, $b+0 \simeq b$ et $a+b \simeq b+a$ qui induisent une structure monoïdale sur la catégorie.

Exemple 2.5.6. Étant donné un monoïde $(M,e,\cdot)$, on construit la catégorie M dont les objets sont les éléments du monoïde et dont les seuls morphismes sont les identités. L'associativité de « $\cdot$ » et le fait que e est un élément neutre pour « $\cdot$ » induisent une structure monoïdale sur M.

Exemple 2.5.7. Les *ensembles pointés* donnent un autre exemple de structure monoïdale. La catégorie des ensembles pointés **EnsPt** est définie comme suit. Les objets sont les paires (X,x) où X est un ensemble et x un élément de X. Les morphismes $(X,x) \to (Y,y)$ sont les fonctions $f : X \to Y$ telles que $f(x) = y$. Soient (X,x), (Y,y) deux ensembles pointés. Le smash des deux ensembles pointés noté $(X,x) \wedge (Y,y)$ est l'ensemble $\{(x,y)/\simeq\}$ où $\simeq$ est la plus petite relation

17. comme de toute autre propriété universelle.

d'équivalence qui contient $(x',y) \simeq (x,y) \simeq (x,y')$ pour tout $x' \in X$ et tout $y' \in Y$, le point de base étant la classe d'équivalence de (x,y). En d'autres termes, on identifie les éléments de la croix, et seulement ceux-ci (à gauche) ; l'associativité se « lit » dans le cube (à droite) :



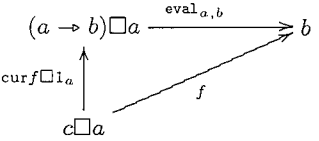
Théorème 2.5.7.1 (Théorème de cohérence).
Soit une structure monoïdale $(C,e,\square,\alpha,\lambda,\rho,\gamma)$. *Étant donné* n *variables* $x_1,x_2,\ldots,x_n$, *nous considérons deux termes* $t(x_1,\cdots,x_n)$, $s(x_1,\cdots,x_n)$ *sur l'algèbre* $\{\square,e\}$ *avec* $\mathbf{ar}(\square) = 2$ *et* $\mathbf{ar}(e) = 0$. *Nous supposons en outre que les variables apparaissent une et une seule fois dans chacun des termes. Ainsi,* t *et* s *définissent deux foncteurs :* $T : C^n \to C$ *et* $S : C^n \to C$.

- *Si les variables de* s *et* t *apparaissent dans le « même ordre », il existe un unique isomorphisme naturel* $S \simeq T$ *construit uniquement au moyen de* α,λ *et* ρ.
- *Si les variables sont dans un ordre quelconque, il existe un unique isomorphisme naturel* $S \simeq T$ *construit uniquement au moyen de* α,λ,ρ *et* γ.
- *Si les termes* s *et* t *ne contiennent pas de symbole* e *et que les variables sont rangées dans l'ordre, il existe un unique isomorphisme naturel* $S \simeq T$ *construit uniquement au moyen de* α.
- *Si les termes* s *et* t *ne contiennent pas de symbole* e *mais que les variables ne sont pas dans l'ordre, il existe un unique isomorphisme naturel* $S \simeq T$ *construit uniquement au moyen de* α *et* γ.

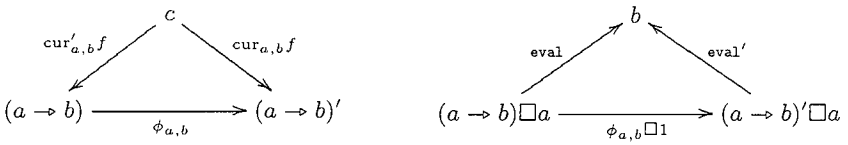
Par la suite, nous ne ferons plus de distinctions entre structure monoïdale relâchée ou non.

Le théorème de cohérence nous permet de passer d'un terme n-aire à un autre via un unique isomorphisme. Ceci nous permet de manipuler les tenseurs n-aire d'objets comme des multi-ensembles de n éléments ; nous utiliserons le théorème pour interpréter les contextes des λ-calculs.

Définition 2.5.8. Une catégorie monoïdale $(\mathcal{C},e,\Box,\alpha,\lambda,\rho)$ est *monoïdale fermée* si pour toute paire d'objets a et b de la catégorie, il y a un objet $a \multimap b$ appelé Hom-interne[18] de b par a et un morphisme $\mathtt{eval}_{a,b} : (a \multimap b)\Box a \to b$ tel que pour tout morphisme $f : (a\Box c) \to b$, il existe un unique morphisme[19] $\mathrm{cur}_{a,b} f : c \to (a \multimap b)$ tel que le diagramme commute :



Notons que l'objet $(a \multimap b)$ est défini à isomorphisme unique près comme toute propriété universelle [BW90] : étant donné une autre construction de Hom-interne $((a \multimap b)', \mathtt{eval}'_{a,b}, \mathrm{cur}'_{a,b})$, il existe un unique isomorphisme $\phi_{a,b} : (a \multimap b) \to (a \multimap b)'$ tel que le diagramme suivant commute pour tout $c \in \mathcal{C}$:



La donnée d'un objet de Hom-interne pour tous a et b peut être prolongée en un bifoncteur appelé « bifoncteur d'Homomorphisme interne», le bifoncteur «externe» étant le bifoncteur défini à l'exemple 2.4.8. Ce bifoncteur est contravariant en la première variable et covariant en

18. Nous évitons le terme d'exponentielle, eu égard à la logique linéaire pour laquelle ce mot a un autre emploi.

19. C'est la propriété universelle du Hom-interne.

la seconde. Pour tout $f : b \to a$ et $g : c \to d$, on définit le morphisme $f \multimap g = \mathrm{cur}(g \circ \mathtt{eval}_{b,c}) \circ \mathrm{cur}(\mathtt{eval}_{a,b} \circ (e \square g)) : (a \multimap c) \to (b \multimap d)$.

Proposition 2.5.9. Le bifoncteur d'Homomorphisme interne nous donne un deuxième exemple d'adjonction. Le foncteur $(b \multimap -) : \mathcal{C} \to \mathcal{C}$ est l'adjoint à droite du foncteur $(- \square b) : \mathcal{C} \to \mathcal{C}$:

$$\mathrm{Hom}_{\mathcal{C}}(a \square b, c) \simeq \mathrm{Hom}_{\mathcal{C}}(a, b \multimap c)$$

En fait, les deux propriétés sont équivalentes : si le foncteurs $(b \multimap -)$ est adjoint à droite du foncteur $(- \square b)$, alors on défini $\mathrm{cur}(f) = \alpha(f)$ où α est l'isomorphisme $\alpha_{a,b,c} : \mathrm{Hom}(a \square b, c) \to \mathrm{Hom}(a, b \multimap c)$, et $\mathtt{eval}_{b,c} = \phi^{-1}(1_{b \multimap c})$ d'après l'isomorphisme $\mathrm{Hom}((b \multimap c) \square b, c) \simeq \mathrm{Hom}(b \multimap c, b \multimap b)$.

Exemple 2.5.10. La construction du Hom-interne catégorique généralise la notion de currification des fonctions sur les ensembles : à toute fonction de deux variables (de types A et C), on peut associer une fonction d'une variable (de type C) dont le résultat est une fonction à une variable (de type A). Plus précisément, si $f : A \times C \to B$, la fonction currifiée $\mathrm{cur}f : C \to (A \Rightarrow B)$ (où $A \Rightarrow B$ est l'ensemble des fonctions de A vers B) est définie par :

$$\begin{array}{rccl} \mathrm{cur}f : & C & \to & A \Rightarrow B \\ & c & \mapsto & \left\{ \begin{array}{ccc} A & \to & B \\ a & \mapsto & f(a,c) \end{array} \right. \end{array}$$

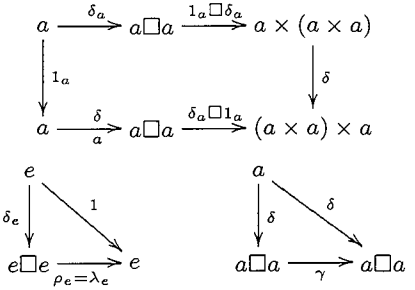
Nous définissons le morphisme $\mathtt{eval} : (A \Rightarrow B) \times B \to B$ par

$$(f, b) \mapsto f(b)$$

Comme le bifoncteur $\square$ sur la catégorie **Ens** est le produit, la catégorie **Ens** est cartésienne fermée.

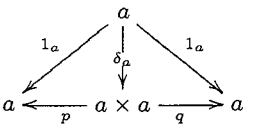
Exemple 2.5.11. Nous avons vu que la catégorie des ensembles pointés est monoïdale. C'est en fait une catégorie monoïdale fermée. Le Hom-interne $(X,x) \multimap (Y,y)$ est $(\mathrm{Hom}_{\mathbf{EnsPt}}(X,Y), [u \in X \mapsto y])$. L'évaluation et la currification sont héritées des ensembles.

Définition 2.5.12 (Diagonale). Supposons que nous disposions d'une catégorie monoïdale symétrique $(\mathcal{C},e,\Box,\alpha,\gamma,\lambda,\rho)$; la structure *dispose des diagonales* (cf Jacobs [Jac94]) s'il existe une transformation naturelle δ telle que les diagrammes suivants commutent :



Cette construction est utilisée pour interpréter la contraction dans les λ-calculs.

Exemple 2.5.13. Un exemple important de structure qui dispose des diagonales est celui des catégories cartésiennes $\mathcal{C}$. Pour tout objet a, on définit le morphisme δ_a comme étant l'unique morphisme qui fait commuter le diagramme :



Exemple 2.5.14. La structure monoïdale des ensembles pointés dispose également des diagonales.

2.6 Interprétation catégorique du λ-calcul

Nous supposons dans cette section que nous disposons d'un calcul simplement typé linéaire sur une signature Σ. Nous supposons en outre que nous disposons d'une catégorie monoïdale $(\mathcal{C},e,\Box,\alpha,\gamma,\lambda,\rho)$.

Le principe de l'interprétation du λ-calcul dans la catégorie est d'associer à tout type X, un objet $[\![X]\!]$ de $\mathcal{C}$, à tout contexte $\Gamma = x_1 : X_1, \ldots, x_n : X_n$, l'objet $[\![\Gamma]\!] = (\cdots([\![X_1]\!]\Box[\![X_2]\!])\cdots)\Box[\![X_n]\!]$ et à tout terme en contexte $\Gamma \vdash t : X$ un morphisme $[\![t]\!] : [\![\Gamma]\!] \to [\![X]\!]$. Nous montrerons que le lemme de substitution est interprété par la composition dans le modèle catégorique.

Pour éviter un trop grand nombre de crochets sémantiques, nous utilisons la notation X à la place de $[\![X]\!]$ lorsque l'environnement le permet.

Définition 2.6.1. Étant donné un λ-calcul sur une signature Σ, une sémantique catégorique est la donnée :

(o) d'une catégorie $\mathcal{C}$,

(i) d'un objet $\mathrm{A} \in \mathcal{C}$ pour tout type atomique A,

(ii) d'un objet $\mathrm{X} \multimap \mathrm{Y} \in \mathcal{C}$ pour tous types X et Y,

(iii) d'une structure monoïdale symétrique sur $\mathcal{C}$: $(\mathcal{C}, e, \Box, \alpha, \beta, \lambda, \rho)$,

(iv) d'un morphisme $\mathrm{f} : (\cdots(\mathrm{X}_1\Box\mathrm{X}_2)\cdots)\Box\mathrm{X}_n \to \mathrm{Y}$ pour tout symbole $f \in \Sigma$ d'arité $X_1, \cdots, X_n \to Y$,

(v) d'une transformation naturelle, $\mathrm{cur}_{X,Y} : \mathcal{C}(-\Box\mathrm{X},\mathrm{Y}) \to \mathcal{C}(-,\mathrm{X} \multimap \mathrm{Y})$ pour tous types X et Y ; en d'autres termes, d'un morphisme $\mathcal{C}(c\Box\mathrm{X},\mathrm{Y}) \to \mathcal{C}(c,\mathrm{X} \multimap \mathrm{Y})$ pour tout objet $c \in \mathcal{C}$ tel que pour tout $f : c' \to c$, le diagramme suivant commute :

$$\begin{array}{ccc} \mathcal{C}(c\Box\mathrm{X},\mathrm{Y}) & \xrightarrow{\mathrm{cur}_{c,X,Y}} & \mathcal{C}(c,\mathrm{X} \multimap \mathrm{Y}) \\ \downarrow{\scriptstyle \mathcal{C}(f\Box 1,\mathrm{Y})} & & \downarrow{\scriptstyle \mathcal{C}(f,\mathrm{X}\multimap\mathrm{Y})} \\ \mathcal{C}(c'\Box\mathrm{X},\mathrm{Y}) & \xrightarrow{\mathrm{cur}_{c',X,Y}} & \mathcal{C}(c',\mathrm{X} \multimap \mathrm{Y}) \end{array}$$

(vi) d'un morphisme $\mathtt{eval}_{X,Y} : (\mathrm{X} \multimap \mathrm{Y})\Box\mathrm{X} \to \mathrm{Y}$ pour tous types X et Y.

Notons que la structure donnée s'interprète dans toute structure monoïdale fermée, il s'agit en effet de la même structure à ceci près que les constructions $(\mathrm{X} \multimap \mathrm{Y}, \mathtt{eval}_{X,Y}, \mathrm{cur}(f))$ ne vérifient pas la propriété universelle des Hom-interne. Dans le cas où l'on aura cette dernière, nous pourrons interpréter la β et la η réduction.

Le point (i) et le point (ii) servent à interpréter les types du λ-calcul.

Le point (iii) permet de construire les contextes. L'interprétation du contexte vide est e, et celle d'un contexte Γ,$x : X$ est $\Gamma\Box$X. Ainsi, $\Gamma = x_1 : X_1, \ldots, x_n : X_n$ s'interprète comme ceci :

$$\Gamma = (\cdots(\mathrm{X}_1\Box\mathrm{X}_2)\Box\cdots)\Box\mathrm{X}_n$$

Dorénavant, une expression de la forme $E_1\Box E_2\cdots\Box E_n$ est considérée comme parenthésée à gauche. Nous utiliserons également l'abréviation $\bigsqcap_{i=1}^{n} E_i$.

Les points (iv)-(vi) servent à interpréter les termes. On procède par induction sur les jugements. Un jugement de type $\Gamma \vdash t : X$ s'interprète comme un morphisme $\mathrm{t} : \Gamma \to \mathrm{X}$.

Définition 2.6.2 (Interprétation catégorique du calcul).

$$\frac{}{x : X \vdash x : X}\ \text{Axiome} \qquad \mathrm{X} \xrightarrow{1_\mathrm{X}} \mathrm{X}$$

$$\frac{\Gamma, x : X, y : Y, \Delta \vdash t : Y}{\Gamma, y : Y, x : X, \Delta \vdash t : Y}\ \text{Éch.} \qquad \frac{\Gamma\Box\mathrm{X}\Box\mathrm{Y}\Box\Delta \xrightarrow{\mathrm{t}} \mathrm{Y}}{\Gamma\Box\mathrm{Y}\Box\mathrm{X}\Box\Delta \overset{\delta}{\cong} \Gamma\Box\mathrm{X}\Box\mathrm{Y}\Box\Delta \xrightarrow{\mathrm{t}} \mathrm{Y}}$$

$$\frac{\forall i \leq n,\ \Gamma_i \vdash t_i : X_i}{\bigcup_{i=1}^{n} \Gamma_i \vdash f(t_1, \ldots, t_n) : Y}\ f \in \Sigma \qquad \frac{\forall i \leq n,\ \Gamma_i \xrightarrow{\mathrm{t}_i} \mathrm{X}_i}{\bigsqcap_{i=1}^{n} \Gamma_i \xrightarrow{\mathrm{f}\circ(\mathrm{t}_1\Box\cdots\Box\mathrm{t}_n)} \mathrm{Y}}$$

$$\frac{\Gamma, x : X \vdash t : Y}{\Gamma \vdash \lambda x.t : X \multimap Y}\ \text{Abst.} \qquad \frac{\Gamma\Box\mathrm{X} \xrightarrow{\mathrm{t}} \mathrm{Y}}{\Gamma \xrightarrow{\mathrm{cur(t)}} \mathrm{X} \multimap \mathrm{Y}}$$

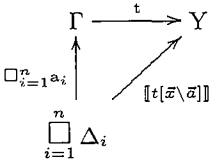
$$\frac{\Gamma \vdash t : X \multimap Y \qquad \Delta \vdash a : X}{\Gamma, \Delta \vdash (t)a : Y}\ \text{App.} \qquad \frac{\Gamma \xrightarrow{\mathrm{t}} \mathrm{X} \multimap \mathrm{Y} \qquad \Delta \xrightarrow{\mathrm{a}} \mathrm{X}}{\Gamma\Box\Delta \xrightarrow{\mathrm{eval}_{X,Y}\circ(\mathrm{t}\Box\mathrm{a})} \mathrm{Y}}$$

Dans l'interprétation de la règle d'échange, le morphisme δ est l'unique morphisme construit à l'aide du théorème de cohérence.

Remarque 2.6.3. Notons que l'α-équivalence est transparente dans l'interprétation catégorique.

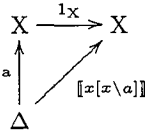
La correction du modèle catégorique est traduite par le fait que la substitution dans la syntaxe peut être interprétée par la composition dans la sémantique.

Propriété 2.6.4 (Sémantique de la substitution). La substitution d'un terme pour une variable dans un terme en contexte est interprétée via la composition dans la catégorie. Supposons que nous ayons $\Gamma \vdash t : Y$ avec $\Gamma = x_1 : X_1, \ldots, x_n : X_n$ et pour tout $i \leq n$, $\Delta_i \vdash a_i : X_i$. Alors, le diagramme suivant commute :



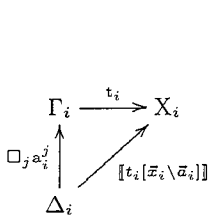
Démonstration. Par induction sur la construction de $t[x\backslash a]$.

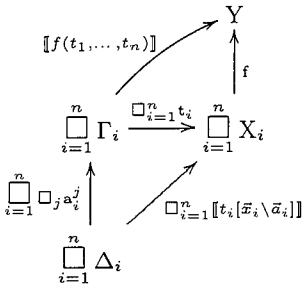
- Cas $\dfrac{\overline{x : X \backslash x : X} \quad \Delta \vdash a}{\Delta \vdash x[x\backslash a] = a : X}$. On observe alors le diagramme :



- Cas $\dfrac{\dfrac{\forall i \leq n, \Gamma_i \vdash t_i : X_i}{\bigcup_{i=1}^{n} \Gamma_i \vdash f(t_1, \ldots, t_n) : Y} \quad \Delta_i^k \vdash a_i^k : X_i^k}{\bigcup_{i,k} \Delta_i^k \vdash f(t_1, \ldots, t_n)[\vec{x}\backslash\vec{a}] : Y}$

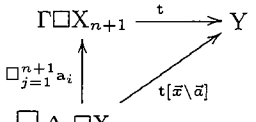
où $x_i^k : X_i^k$ désigne la $k^{\text{ème}}$ variable du contexte Γ_i. Par définition, le terme $[\![f(t_1,\ldots,t_n)[\vec{x}\backslash\vec{a}]]\!]$ est égal à $\mathrm{f}\circ([\![t_1[\vec{x}_1\backslash\vec{a}_1]]\!]\Box\cdots[\![t_1[\vec{x}_n\backslash\vec{a}_n]]\!])$. Par induction, nous avons le diagramme de gauche pour tout $i \leq n$. On en déduit alors le diagramme de droite :

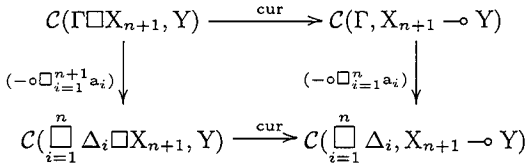




– Cas
$$\frac{\dfrac{\Gamma, x_{n+1} : X_{n+1} \vdash t : Y}{\Gamma \vdash \lambda x_{n+1}.t : X_{n+1} \multimap Y} \quad \Delta_i \vdash a_i : X_i}{\bigcup_i \Delta_i \vdash (\lambda x_{n+1}.t)[\vec{x}\backslash\vec{a}] : X \multimap Y}$$

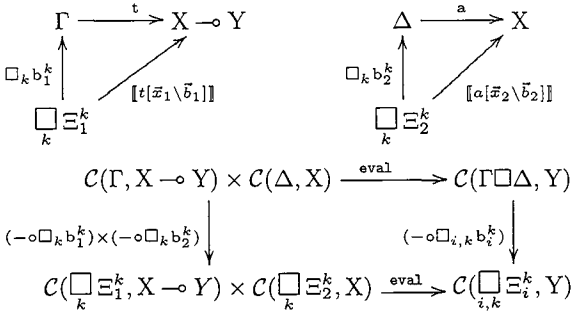
On définit $a_{n+1} = x_{n+1}$ dont le jugement de formation est $x_{n+1} : X_{n+1} \vdash a_{n+1} : X_{n+1}$. On a $(\lambda x_{n+1}.t)[\vec{x}\backslash\vec{a}] = \lambda x_{n+1}.t[\vec{x}\backslash\vec{a}]$. Par induction, on dispose du jugement de gauche. On utilise alors la naturalité de « cur » pour conclure.





- Cas
$$\dfrac{\dfrac{\Gamma \vdash t : X \multimap Y \quad \Delta \vdash a : X}{\Gamma,\Delta \vdash (t)a : Y} \qquad \Xi_i^k \vdash b_i^k : X_i^k}{\bigcup_{i,k} \Xi_i^k \vdash (t)a[\vec{x}\backslash\vec{b}]}$$

où l'on note x_1^k la $k^{\text{ème}}$ variable du contexte Γ et x_2^k la $k^{\text{ème}}$ variable du contexte Δ. On utilise des conventions équivalentes à celles vues plus haut pour les types et les termes $\Xi_i^k \vdash b_i^k : X_i^k$. Par induction, on dispose des deux diagrammes de gauche. On se sert alors de la naturalité de l'application pour conclure.



□

Propriété 2.6.5 (Contraction et Affaiblissement). Lorsque le tenseur est un produit catégorique, on peut interpréter le λ-calcul ordinaire.

Donnons l'interprétation catégorique de

$$\dfrac{\forall i \leq n, \Gamma_i \vdash t_i : X_i}{\bigcup_{i=1}^n \Gamma_i \vdash f(t_1, \ldots, t_n) : Y}$$

On construit pour tout $i \leq n$, la projection $\Gamma \xrightarrow{\pi_i} \Gamma_i$. On définit alors pour des contextes non disjoints, $\mathrm{f}(\mathrm{t}_1, \ldots, \mathrm{t}_n) = \mathrm{f} \circ \prod_{i=1}^n (\mathrm{t}_i \circ \pi_i) \circ \prod_{i=2}^n \delta$ (où δ est la diagonale).

L'interprétation catégorique de l'application est analogue : étant donné le jugement

$$\frac{\Gamma \vdash t : X \to Y \qquad \Delta \vdash a : X}{\Gamma,\Delta \vdash (t)a : Y}$$

où Γ et Δ ne sont pas nécessairement disjoints. On note π_Γ la projection $[\![\Gamma \cup \Delta]\!] \xrightarrow{\pi_\Gamma} \Gamma$ et π_Δ la projection $[\![\Gamma \cup \Delta]\!] \xrightarrow{\pi_\Delta} \Delta$. On définit alors $[\![(t)a]\!] = \mathtt{eval} \circ ((\mathrm{t} \circ \pi_\Gamma) \times (\mathrm{a} \circ \pi_\Delta)) \circ \delta$.

D'autre part, on interprète la règle d'affaiblissement :

$$\frac{}{x_1 : X_1, \ldots, x_n : X_n \vdash x_i : X_i}$$

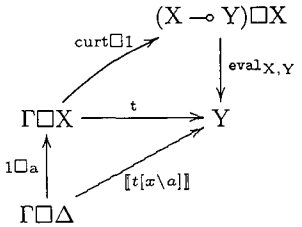
par la projection catégorique : $\mathrm{X}_1 \times \cdots \times \mathrm{X}_n \xrightarrow{\pi_i} \mathrm{X}_i$.

Avec une telle interprétation, le lemme de substitution est toujours interprété par la composition catégorique (cf Jacobs [Jac94]). Plus généralement, les conditions nécessaires pour pouvoir interpréter la contraction et l'affaiblissement sont décrites dans [Jac94]. Pour la contraction, il faut que la catégorie dispose des diagonales, et pour l'affaiblissement, il suffit de disposer de morphismes $X \square Y \to X$ pour tous objet X,Y.

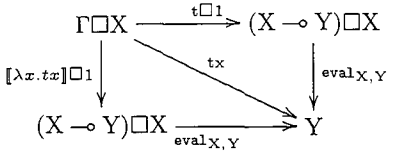
Propriété 2.6.6 (β-réduction et η-réduction). Lorsque le type « $X \multimap Y$ », la construction « cur$f$ » et « `eval` » sont issus d'un foncteur d'hom-interne, en d'autres termes, lorsque l'on dispose d'une structure monoïdale fermée, on peut également interpréter la β et la η-réduction. La β-réduction (étant une substitution particulière) correspond à la composition catégorique.

Plus précisément, la β-égalité correspond à *l'égalité* $\mathtt{eval} \circ (\mathrm{cur}f \square 1) = f$. La η-réduction correspond à *l'unicité* de « cur$f$ ».

Démonstration. On suppose disposer des jugements $\Gamma \vdash \lambda x.t : X \multimap Y$ et $\Delta \vdash a : X$. On peut alors conclure $\Gamma,\Delta \vdash ((\lambda x.t)a) : Y$. La β-réduction donne l'égalité $((\lambda x.t)a) = t[x \backslash a]$. On retrouve cette égalité dans la sémantique si l'on suppose $\mathtt{eval} \circ (\mathrm{cur}f \square 1) = f$ pour tout f :



En ce qui concerne la η-égalité, notons tout d'abord que l'on dispose du diagramme :



où le triangle inférieur est obtenu par currification de la diagonale. On en déduit l'égalité $\texttt{eval} \circ (\texttt{t}\Box 1) = \texttt{tx} = \texttt{eval} \circ ([\![\lambda x.tx]\!]\Box 1)$. Par unicité, on aboutit au résultat cherché. □

2.7 Propriétés des multi-graphes

Nous présentons dans cette section quelques résultats classiques concernant les multi-graphes auxquels nous ferons référence dans le chapitre 6. Nous nous sommes inspirés du livre de Barr et Wells [BW90] qui traite la catégorie des multi-graphes de manière très détaillée (et à l'aide de moyens élémentaires).

Proposition 2.7.1. Comme nous l'avons déjà fait remarquer, la catégorie des graphes squelettique **Gr** constitue une sous-catégorie pleine des multi-graphes. L'inclusion admet un adjoint à gauche $(\widehat{-})$ donné par : pour tout multi-graphe G, $\widehat{G}$ est le graphe squeletique obtenu en identifiant toute les arêtes entre deux sommets.

- $|\widehat{G}| = |G|$,
- $(x,y) : x - y \in R_{\widehat{G}}$ ssi $\exists p, p : x - y \in G$.

On vérifiera en particulier que si X est un multi-graphe et G un graphe que nous avons

$$\mathrm{Hom}_{\mathbf{Gr}}(\widehat{X},G) \simeq \mathrm{Hom}_{\mathbf{MGr}}(X,G)$$

Proposition 2.7.2. La catégorie des multi-graphes est cartésienne fermée.

Démonstration. Le graphe sous-jacent de la catégorie **1** (constituée d'un seul sommet et d'une seule arête) est un objet terminal dans **MGr**. Il nous reste à construire le produit binaire $(- \times -)$ sur **MGr** et un hom-interne.

Définition 2.7.3. Soient G et H deux multi-graphes, nous définissons leur produit $G \times H$:

- $|G \times H| = |G| \times |H|$,
- $R_{G\times H} = R_G \times R_H$. Pour toutes arêtes $p : x - x' \in G$ et $q : y - y' \in H$, $(p,q) : (x,y) - (x',y') \in (G \times H)$.

Pour le choix de produit que nous venons de faire, nous pouvons décrire directement le bifoncteur produit correspondant : pour tous morphismes $g : G \to G'$ et $h : H \to H'$,

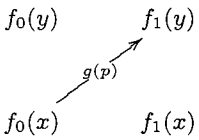
$$\begin{array}{rcl} g \times h : (G \times H) & \to & G' \times H' \\ (x,y) & \mapsto & (g(x),h(y)) \\ (p : x - x', q : y - y') & \mapsto & (g(p),h(q)) : (g(x),h(y)) - (g(x'),h(y')) \end{array}$$

Définition 2.7.4. Étant donné deux multi-graphes F et G, nous considérons le graphe $F \Rightarrow G$ défini comme suit :

- $|F \Rightarrow G| = \mathbf{Ens}(|F|,|G|)$,
- les arêtes consistent en des triplets de fonctions (f_0,f_1,g) avec $f_0 : |F| \to |G|$, $f_1 : |F| \to |G|$ et $g : R_F \to R_G$ tels que le diagramme commute (à comparer avec les morphismes de multi-graphes) :

$$\begin{array}{ccccc} |F| & \xleftarrow{\mathrm{dom}_F} & R_F & \xrightarrow{\mathrm{cod}_F} & |F| \\ {\scriptstyle f_0}\downarrow & & {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle f_1} \\ |G| & \xleftarrow{\mathrm{dom}_G} & R_G & \xrightarrow{\mathrm{cod}_G} & |G| \end{array}$$

Le domaine de (f_0,f_1,g) est f_0, son codomaine est f_1. De manière visuelle, g est une arête entre une fonction f_0 et une fonction f_1 si pour tout $p : x - y$, on a (les arêtes sont représentées par des flèches) :



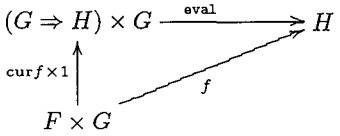
Nous définissons le morphisme $\texttt{eval} : (F \Rightarrow G) \times F \to G$:

$$\begin{cases} |\texttt{eval}|(f : |F| \to |G|,x) & = & f(x) \\ R_{\texttt{eval}}((f_0,f_1,g),p : x - y) & = & g(p) : f_0(x) - f_1(y) \end{cases}$$

Pour tout morphisme $f : F \times G \to H$, nous définissons alors le morphisme $\mathrm{cur}f : F \to (G \Rightarrow H)$.

$$\begin{array}{rrcl} \mathrm{cur}f : & F & \to & (G \Rightarrow H) \\ & x & \mapsto & [y \mapsto f(x,y)] \\ & q : x - x' & \mapsto & ([y \in |G| \mapsto f(x,y)], [y \in |G| \mapsto f(x',y)], \\ & & & \quad [q : y - y' \mapsto f(p,q)]) \end{array}$$

On vérifie alors aisément que le diagramme commute :



On vérifie en outre que $\mathrm{cur}(f)$ est déterminé de manière unique par f. □

Remarque 2.7.5. Nous aurions pu obtenir le résultat en observant l'équivalence entre les catégories $\mathbf{MGr} \simeq [(s \underset{c}{\overset{d}{\rightrightarrows}} a) \Rightarrow \mathbf{Ens}]$; nous pouvons en déduire que la catégorie est cartésienne fermée (à l'aide du lemme de Yoneda).

2.7.1 Propriété des multi-graphes réflexifs

Définition 2.7.6. Un multi-graphe *réflexif* est la donnée d'une paire $(F,\mathtt{ref}(-))$ où F est un multi-graphe et $\mathtt{ref}(-)$ est une fonction, dite de réflexivité, $|F| \to R_F$ telle que pour tout $b \in |G|$, nous avons $\mathtt{ref}(b) : b - b$. Un morphisme de graphe réflexif $(F,\mathtt{Ref}_F) \to (G,\mathtt{Ref}_G)$ est la donnée d'un morphisme $f : F \to G$ tel que le diagramme commute :

$$\begin{array}{ccc} |F| & \xrightarrow{\mathtt{Ref}_F} & R_F \\ \downarrow{\scriptstyle |f|} & & \downarrow{\scriptstyle R_f} \\ |G| & \xrightarrow{\mathtt{Ref}_G} & R_G \end{array}$$

MRef désigne la catégorie des multi-graphes réflexifs.

On notera en particulier qu'une catégorie est un multi-graphe réflexif.

Proposition 2.7.7. La catégorie des multi-graphes réflexifs est cartésienne fermée.

Démonstration. Comme pour les multi-graphes, le graphe **1** est terminal. Plus précisément, $(\mathbf{1},[0 \mapsto 1_0])$ constitue un graphe réflexif ; ce graphe réflexif est terminal.

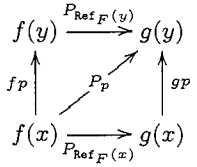
Définition 2.7.8 (Produit). Étant donné deux multi-graphes réflexifs $(F,\mathtt{Ref}_F)$ et $(G,\mathtt{Ref}_G)$, nous construisons le multi-graphe réflexif produit composante par composante (comme pour les multi-graphes) :

- $|F \times G| = |F| \times |G|$,
- $R_{F\times G} = R_F \times R_G$. Pour toutes arêtes $p : x - x' \in F$ et $q : y - y' \in G$, $(p,q) : (x,y) - (x',y') \in (F \times G)$,
- $\mathtt{Ref}_{F\times G}(x,y) = (\mathtt{Ref}_F(x),\mathtt{Ref}_G(y))$.

Définition 2.7.9 (Hom-interne). Étant donné deux multi-graphes réflexifs $(F,\mathtt{Ref}_F)$ et $(G,\mathtt{Ref}_G)$, nous construisons le multi-graphe réflexif hom-interne $(F,\mathtt{Ref}_F) \Rightarrow (G,\mathtt{Ref}_G)$ de la manière suivante.

- $|(F,\mathtt{Ref}_F) \Rightarrow (G,\mathtt{Ref}_G)| = \mathbf{MRef}((F,\mathtt{Ref}_F),(G,\mathtt{Ref}_G))$,

- une arête $P : f-g \in F \Rightarrow G$ est la donnée pour tout $p : x-y \in F$ d'une arête $P_p : f(x) - g(y)$. Nous pouvons représenter ceci sur un diagramme :



- Étant donné un morphisme de multi-graphe $f : F \Rightarrow G$, la fonction $p : x - y \in F \mapsto fp$ est l'arête de réflexivité $f - f$.

Le morphisme `eval` s'obtient de manière analogue à celui des multi-graphes :

$$\begin{cases} |\mathtt{eval}|_{(F,\mathtt{Ref}_F),(G,\mathtt{Ref}_G)}(f : F \to G, x) & = & f(x) \\ R_{\mathtt{eval}}(P : (f,\mathtt{Ref}_f) - (g,\mathtt{Ref}_g), p : x - y) & = & P_p : f(x) - g(y) \end{cases}$$

La currification est, comme pour les multi-graphes, directement inspirée de la currification sur les ensembles. Pour tout morphisme $f : ((F,\mathtt{Ref}_F) \times (G,\mathtt{Ref}_G)) \to (H,\mathtt{Ref}_H)$, on définit le morphisme $\mathrm{cur}f : F \to (G \to H)$:

$$\begin{array}{rrcl} \mathrm{cur}f : & F & \to & (G \Rightarrow H) \\ & x & \mapsto & [y \mapsto f(x,y),\ q : y - y' \mapsto f(\mathtt{Ref}(x),q)] \\ p : x - x' & & \mapsto & [q : y - y' \mapsto f(p,q) : \mathrm{cur}f(x) - \mathrm{cur}f(y)] \end{array}$$

□

Remarque 2.7.10. Soit la catégorie engendrée par le multigraphe $s \underset{c}{\overset{d}{\rightleftarrows}} a$. On considère la catégorie $\mathcal{C}$ qui a les même sommets, mais qui identifie $d \circ r = 1_s$, $c \circ r = 1_s$.Pour la construction du produit et du hom-interne, on peut utiliser l'équivalence $\mathbf{MRef} \simeq [\mathcal{C} \Rightarrow \mathbf{Ens}]$.

2.7.2 Autres propriétés des multi-graphes

Les deux catégories **MGr** et **MRef** sont des catégories de pré-faisceaux (remarque 2.4.13). Ces catégories ont des propriétés très particulières, notamment d'être cartésiennes fermées comme en témoignent les deux exemples.

Propriété 2.7.11. La sous-catégorie des graphes squelettique **Gr** est également cartésienne fermée. La construction est héritée de celles de la catégorie **MGr**. Un sommet de la relation binaire $(X,R_X) \Rightarrow (Y,R_Y)$ est une fonction $X \to Y$, et il y a une arête entre deux fonctions $f, g : X \to Y$ si $fx - gy$ pour tout $p : x - y \in X$.

Pour les graphes réflexifs, l'ensemble des sommets du Hom-foncteur $(X,R_X) \Rightarrow (Y,R_Y)$ s'interprète comme l'ensemble des fonctions monotones $X \to Y$. Il y a un arc entre deux telles fonctions si pour tout $p : x - y$, on a $fx - gy$. Notons qu'en raison de la réflexivité, on a $fx - gx$ pour tout $x \in X$.

Dans le cas des ordres, l'ensemble des sommets du Hom-foncteur $(X, \leq_X) \Rightarrow (Y, \leq_Y)$ s'interprète comme l'ensemble des fonctions monotones $X \to Y$. Et il y a une arêtes entre deux telles fonctions si pour tout $x \leq y \in X$, on a $fx \leq_Y gy$. Mais, en raison de la transitivité *et* de la réflexivité, cela est équivalent au fait que pour tout x, $fx \leq gx$. En d'autres termes, il s'agit également de l'ordre « pointwise » (nous dirons également *point à point*).

Note 2.7.1. Nous utiliserons une autre notation standard pour le Hom-foncteur $(X, \leq_X) \Rightarrow (Y, \leq_Y)$, il s'agit de $(Y, \leq_Y)^{(X,\leq_X)}$.

Propriété 2.7.12. Dans les catégories de pré-faisceaux, les limites et les colimites se calculent de manière « pointwise ». Le premier exemple que nous avons vu est celui des produits de **MGr** et **MRef**. De manière plus générale, étant donné un diagramme $\Delta = ((F_i)_{i\in I},(h_{i,j})_{i,j\in I})$, la limite dans **MGr** de ce diagramme est le multi-graphe $(|F|,R_F)$:

- $|F|$ est un I-uplet $(a_i)_{i\in I}$ de sommets $a_i \in F_i$ tel que pour tout morphisme $f_{i,j} : F_i \to F_j$ on a $f_{i,j}(a_i) = a_j$.
- une arête est un I-uplet $(p_i)_{i\in I}$ d'arêtes $p_i \in F_i$ tel que pour tout morphisme $f_{i,j} : F_i \to F_j$ on a $f_{i,j}(p_i) = p_j$. Cette arête va de $(\mathtt{dom}(p_i))_{i\in I}$ vers $(\mathtt{cod}(p_i))_{i\in I}$.

Interprétation des types dépendants

Nous n'allons pas ici donner la construction sémantique (catégorique) générale pour les types dépendants comme nous avons fait pour les types simples, nous nous contentons de discuter les conditions qui nous permettent d'interpréter les types dépendants dans trois catégories, celle des ensembles, celle des graphes et enfin celle des graphes réflexifs. Les types dépendants sont interprétés au moyen de morphismes qui bénéficient de propriété particulières, les « *display maps* » qui représentent les « familles de types ».

Dans la catégorie des ensembles, toute fonction peut servir de display map. Étant donné une famille d'ensemble $i : I \vdash X_i$, cette famille est représentée par la fonction $\sum_{i \in I} X_i \to I$ qui a un élément $(x,i) \in X_i$ vu dans $\sum_{i \in I} X_i$ associe son index $i \in I$. C'est le display map du type « $i : I \vdash X_i$ type ».

Il se trouve que dans la catégorie des multi-graphes et dans celle des multi-graphes réflexifs, ce phénomène (que tout morphisme peut servir de display map) apparait ; et ceci nous permet d'interpréter les types dépendants. Nous y reviendrons au chapitre 7.

Chapitre 3

Preuves de terminaison et complexité

Pour calculer avec un système de réécriture par normalisation, nous avons besoin de la propriété de terminaison du système. Même si cette propriété est indécidable en général, il existe de nombreuses méthodes de preuves de terminaison, comme les preuves par interprétation sur un ordre bien fondé ou encore comme les ordres récursifs sur les chemins (*recursive path orderings*) [Der82].

Longueur de dérivation, complexité

En fait, une analyse fine des preuves de terminaison des systèmes de réécriture donne en général des bornes supérieures sur la longueur de dérivation de ces derniers. De nombreuses recherches ont été menées dans ce sens. Ainsi, pour les preuves pour l'*ordre multi-ensemble sur les chemins*, Hofbauer montre que la longueur de dérivation est au plus primitive récursive [Hof92b]. Cichon parvient au même résultat de manière indépendante, en utilisant une technique de théorie de la preuve, cf. [Cic90]. Pour l'*ordre lexicographique sur les chemins*, Weiermann donne une borne multiplement récursive [Wei95]. En ce qui concerne les preuves par *interprétation polynomiale* sur les entiers, Hofbauer et Lautemann prouvent que la longueur de dérivation

est au plus (doublement) exponentielle [HL89] (voir également Geupel [Geu88]). Enfin, Hofbauer donne dans sa thèse une borne multiplement récursive (de rang 4 selon Péter [Pét67]) pour les preuves de terminaison à l'aide de l'*ordre de Knuth-Bendix* [Hof92a].

Le problème de ces auteurs est d'établir des restrictions imposées par les méthodes de preuve de terminaison sur la complexité des fonctions calculées. Mais cela ne résout pas la question : « Que peut-on calculer avec de tels systèmes de réécriture ? ». Nous avons choisi de nous référer pour cela aux classes de complexité. Nous appelons problème d'*équivalence* le problème suivant. *Étant donné une méthode de preuve de terminaison M et une classe de complexité C, il y a équivalence si :*

- *toute fonction calculée par un système de réécriture qui termine suivant M est dans C,*
- *pour toute fonction ϕ dans C, il existe un système de réécriture qui termine suivant M et qui calcule ϕ.*

Un des intérêts du problème d'équivalence vient du fait que la longueur de dérivation du système de réécriture ne mesure pas la complexité de manière très fine. Une des raisons est que du fait des duplications de variables dans les règles de réécriture, on peut construire des termes de taille exponentielle en la longueur de dérivation. Prenons par exemple le système composé des deux règles :

$$\begin{aligned} f(s(x),y) &\to f(x,g(y,y)) \\ f(0,y) &\to y \end{aligned}$$

La longueur de dérivation de ce système est de n pour des données de la forme $f(s^n(0),0)$. Mais le terme que l'on obtient est de taille exponentielle. La fonction décrite par le système, une forme de l'exponentielle, n'est pas calculable en temps « linéaire » (ni même polynomial) par une machine de Turing par exemple. De ce fait, une réponse positive au problème d'équivalence fournit une borne de complexité plus précise qu'une simple borne sur la longueur de dérivation.

Le problème d'équivalence a un autre intéret : si toute fonction d'une certaine classe admet une preuve de terminaison par une certaine méthode, cela témoigne de la richesse de cette méthode.

Les premières caractérisations de classe de complexité indépendantes d'un modèle de calcul (analogue à notre problème d'équivalence) ont été proposées par Cobham [Cob62]. Son approche est en termes de « récursions bornées sur la notation » où le taux d'accroissement des fonction est limité par une fonction déjà définie dans la classe. Par rapport à ces travaux, nous utilisons des conditions « locales » (chaque règle de réécriture doit respecter une contrainte d'ordre de réduction) qui ne tiennent pas compte des propriétés extensionnelles de la fonction.

Quelques résultats

Le problème d'équivalence a été étudié pour les ordres de terminaison par interprétation polynomiale par Bonfante, Cichon, Marion et Touzet [BCMT98], [BCMT99]. Le problème d'équivalence pour l'ordre de Knuth-Bendix fait l'objet de la communication de Bonfante [Bon00] et de l'article de Lepper [Lep00]. Avec un point de vue un peu différent quoique lié, Marion et Cichon étudient dans [Mar99, CM99] le contenu algorithmique des ordres MPO et LPO. Dans ces deux articles, l'idée clef est d'extraire un algorithme *efficace* à partir de la spécification du problème (sous la forme d'un système de réécriture) et de la preuve de terminaison de la spécification. Ils montrent ainsi comment extraire des algorithmes calculables en temps polynomial à partir d'un système de réécriture qui termine a priori en temps exponentiel.

Dans ce chapitre, nous traitons le problème d'équivalence pour les preuves de terminaison par interprétation polynomiale. Dans un premier temps, nous classons les systèmes de réécriture suivant la forme de l'interprétation polynomiale des constructeurs ($x+c$, $ax+c$, ax^k+c, 2^x+c). Nous montrons alors que cette classification correspond à une hiérarchie de classes de complexité (PTIME, ETIME, E_2TIME, TIME($\exp_\infty(n+k)$)). Une conséquence pratique de la hiérarchie que nous établissons est de pouvoir déterminer la complexité d'un calcul en fonction de sa preuve de terminaison par interprétation polynomiale. Que nous considérions la réécriture comme un modèle de calcul ou comme une méthode de spécification, une preuve de terminaison nous assure de la faisabilité du calcul ; si nous avons en outre une borne sur la longueur de dérivation qui n'est pas trop élevée, nous pouvons

alors certifier l'*efficacité* du calcul.

Dans un deuxième temps, nous nous intéressons aux liens entre les interprétations polynomiales et les classes de complexité en espace. En particulier, nous étudions les contraintes sur la taille des termes obtenus au cours d'une dérivation lorsque le système admet une preuve de terminaison par interprétation polynomiale. Nous dégageons ensuite une notion de pseudo-preuve de terminaison par interprétation polynomiale qui nous permet d'établir une hiérarchie de complexité parallèle à celle que nous avons présentée pour les interprétations polynomiales (PSPACE, ESPACE, E$_2$SPACE, SPACE($\exp_\infty(n+k)$)).

Peut-on comparer les algorithmes?

Reprenons un instant le problème d'équivalence, étant donné une méthode de preuve de terminaison, M, et une classe de complexité $\mathcal{C}$ compatible avec la preuve de terminaison. Étant donné une fonction ϕ de $\mathcal{C}$, elle est calculée par un système de réécriture qui termine via M. Observons maintenant qu'une telle fonction, étant dans la classe de complexité $\mathcal{C}$, est calculée par une machine de Turing. Nous avons donc deux algorithmes, celui de la machine de Turing et celui de la réécriture. Pouvons-nous les comparer? La difficulté vient du fait que les machines de Turing ne sont pas très adaptées pour une telle comparaison [Gur99]. En effet, même si tout algorithme peut être simulé par une machine de Turing, la machine de Turing peut prendre plus de temps que le système de réécriture. Nous devrions donc, si l'on accepte la thèse de Gurevich [Gur99], utiliser les *Abstract State Machines* comme modèle de calcul de référence.

Néanmoins, lorsque nous ferons une preuve d'équivalence, nous procéderons en simulant les machines de Turing par la réécriture et inversement. Nous n'utilisons pas les propriétés extensionnelles des fonctions, mais les algorithmes de nos modèles qui les calculent. Comme nous prenons soin que la complexité soit respectée dans ces simulations (à un facteur polynomial près), nous montrons une équivalence en termes d'algorithmes (autant que le modèle de calcul des machines de Turing le permet).

D'autre part, du fait que nous faisons des simulations étape par étape, nous pouvons étendre (gratuitement) les résultats aux machines

non déterministes/systèmes de réécriture non confluents. La sémantique des fonctions calculées étant donnée par la maximisation.

3.1 Calculs par interprétation polynomiale

Les preuves de terminaison par interprétation polynomiale ont attiré notre attention pour plusieurs raisons. Premièrement, elles sont sans doute les preuves par interprétation sur les entiers les plus utilisées. Notons à titre d'exemple le cas de systèmes comme le système `Reve` [BCL87] ou le système `Polo` [Gie95a, Gie95b], ou encore le système `Larch` [Lar]. Ensuite, elles concernent des problèmes de petite complexité (la longueur de dérivation est au plus doublement exponentielle, alors qu'elle est largement supérieure pour les autres ordres cités). Par conséquent, elles sont susceptibles de témoigner de l'efficacité des algorithmes spécifiés (par un système de réécriture).

3.1.1 Preuves de terminaison par interprétation polynomiale

Les preuves par interprétation polynomiale ont été suggérées par Lankford [Lan79].

Définition 3.1.1 (Interprétation polynomiale). Un système de réécriture $(\mathcal{R},\Sigma)$ admet une interprétation polynomiale si pour chaque symbole g de Σ, il existe un polynôme $[g]$ appelé interprétation (polynomiale) de g tel que :

1. si g est n-aire, alors, $[g]$ est un polynôme à n variables ;
2. $[g](1,\cdots,1) \geq n+1$ où n est l'arité de g ;
3. pour tous $0 < n < m$, $[g](\cdots,n,\cdots) < [g](\cdots,m,\cdots)$;
4. on étend de manière canonique l'interprétation aux termes clos :

$$[f(t_1,\ldots,t_n)] = [f]([t_1],\ldots,[t_n])$$

Pour toute règle $l \to r$ de $\mathcal{R}$, et pour toute substitution close σ, on doit avoir :

$$[l\sigma] > [r\sigma]$$

Par la suite, nous notons $(\mathcal{R},\Sigma,[])$ les systèmes de réécriture admettant une interprétation polynomiale.

Propriété 3.1.2. Si le système de réécriture admet une interprétation polynomiale, alors il termine. En effet, soit $t_1 \to t_2 \to \cdots \to t_n$ une dérivation. Comme l'interprétation diminue à chaque étape de réécriture, nous sommes assurés que :

$$n \leq [t_1]$$

Nous parlons alors de preuve de terminaison par interprétation polynomiale.

Proposition 3.1.3. Supposons que nous ayons un système de réécriture $(\mathcal{R},\Sigma)$ avec une interprétation $[]$ qui respecte les trois premiers points de la définition précédente. Nous pouvons prolonger l'interprétation polynomiale aux termes non clos en associant à chaque variable x de la signature une variable X. Pour tout terme t, $[t]$ est alors un polynôme en les variables $(X_1,\cdots,X_n)$ où $(x_1,\ldots,x_n)$ sont les variables de t. Une condition suffisante pour que le critère 4 de la définition 3.1.1 soit vérifié est que

$$[l](X_1,\ldots,X_n) > [r](X_1,\ldots,X_n)$$

si $X_i \geq a$ pour tout $i \leq n$ où $a = \min\{[c] \mid c \in \Sigma_0\}$.

Nous utilisons par la suite ce critère. Notons par ailleurs que nous ne ferons plus la distinction entre la variable du polynôme et celle du terme. Nous utiliserons la variable du terme pour dénoter la variable du polynôme.

Définition 3.1.4. Nous utiliserons également des interprétations exponentielles. Étant donné un symbole g n-aire, une interprétation exponentielle de ce symbole est une fonction de la forme :

$$P(x_1,\cdots,x_n,2^{x_1},\cdots,2^{x_n})$$

où P est un polynôme.

Une preuve de terminaison par interprétation exponentielle est la donnée d'une interprétation exponentielle à chaque symbole qui vérifie les conditions 2-3-4 de la définition précédente.

3.1.2 Bornes supérieures générales sur la longueur de dérivation et la taille des termes

Nous supposons dans cette section que le système de réécriture $(\mathcal{R},\Sigma,[])$ admet une preuve de terminaison par interprétation polynomiale.

Propriété 3.1.5. Il existe une constante c telle que pour tout terme t, $[t] \leq 2^{2^{c|t|}}$. La preuve est un cas particulier de la proposition 3.2.6. De cette observation découle le théorème :

Théorème 3.1.5.1 (Hofbauer et Lautemann [HL89]).
La longueur de dérivation des systèmes de réécriture admettant une preuve par interprétation polynomiale est au plus doublement exponentielle.

De plus, cette borne ne peut être affinée, ainsi que le montre l'exemple suivant :

Exemple 3.1.6 (d'après Hofbauer et Lautemann [HL89]). Reprenons l'exemple 1.4.7 :

$$\begin{array}{lcl@{\qquad}lcl} 0+y & \to & y & \mathtt{sq}(0) & \to & 0 \\ sx+y & \to & s(x+y) & \mathtt{sq}(sx) & \to & s((x+x)+\mathtt{sq}(x)) \end{array}$$

Ce système admet une preuve par interprétation polynomiale :

$$\begin{array}{ll} [0] = 1 & [s](x) = x+1 \\ [+](x,y) = 2x+y & [\mathtt{sq}](x) = 3x^2 \end{array}$$

Rappelons que nous avons $\mathtt{sq}\underline{n} \xrightarrow{*}_{\mathcal{R}} \underline{n^2}$. Par conséquent :

$$\underbrace{\mathtt{sq}\cdots\mathtt{sq}}_{n \text{ fois } \mathtt{sq}}(s(s(0))) \xrightarrow{*}_{\mathcal{R}} \underline{2^{2^n}}$$

Or, la hauteur du terme augmente d'au plus une unité pour chacune des règles de réécriture, comme le terme final a une hauteur de 2^{2^n}, la longueur de dérivation est doublement exponentielle.

Remarque 3.1.7. Pour les interprétations exponentielles, nous verrons que l'on peut borner l'interprétation d'un terme t par $[t] \leq \exp_\infty(|t| + k)$ pour une certaine constante k où la fonction $\exp_\infty$ est définie comme suit.

$$\begin{aligned} \exp_\infty(x+1) &= 2^{\exp_\infty(x)} \\ \exp_\infty(0) &= 1 \end{aligned}$$

Nous établissons maintenant des bornes sur la taille des termes obtenus au cours d'une dérivation, et en particulier, sur les formes normales.

Lemme 3.1.8. Pour tout symbole g n-aire ($n > 0$), nous avons $[g](x_1, \cdots, x_n) > \sum_{i=1}^n x_i$.

Démonstration. L'argument clef de la preuve est que pour tout $m > 0$, nous avons :

$$[g](\cdots, m+1, \cdots) \geq [g](\cdots, m, \cdots) + 1$$

Par conséquent, en appliquant l'inégalité à chacune des variables, il vient :

$$\begin{aligned} [g](x_1, \cdots, x_n) &\geq \textstyle\sum_{i=1}^n (x_i - 1) + [g](1, \cdots, 1) \\ &\geq 1 + \textstyle\sum_{i=1}^n x_i \qquad \text{(2)- def 3.1.1} \end{aligned}$$

□

Proposition 3.1.9. Pour tout terme clos t, nous avons une borne sur la taille, $|t| \leq [t]$.

Démonstration. Par induction sur les termes,

- si t est une constante, alors, $|t| = 1 \leq [t]$ d'après l'hypothèse (2) de la définition 3.1.1.
- si $t = g(t_1, \cdots, t_n)$, alors

$$\begin{aligned} |t| &= 1 + \textstyle\sum_{i=1}^n |t_i| && \text{par définition} \\ &\leq 1 + \textstyle\sum_{i=1}^n [t_i] && \text{par induction} \\ &\leq [t] && \text{d'après le lemme 3.1.8} \end{aligned}$$

□

Définition 3.1.10. Une classe de fonctions $\mathbb{C}$ *capture les polynômes* si pour toute fonction $\phi \in \mathbb{C}$:

(i) ϕ est croissante ;

(ii) pour tout polynôme P, il existe une fonction $\phi' \in \mathbb{C}$ telle que pour tout $x > 0$, on a

$$P(\phi(x),\phi(x),\cdots,\phi(x)) < \phi'(x)$$

En particulier, nous considérons par la suite quatre classes de fonctions qui capturent les polynômes :

- la classe des polynômes eux-mêmes ;
- la classe des fonctions exponentielles : $\{x \mapsto 2^{kx} \mid k \in \mathbb{N}\}$;
- la classe des fonctions doublement exponentielles : $\{x \mapsto 2^{2^{kx}} \mid k \in \mathbb{N}\}$;
- la classe des fonctions tours : $\{x \mapsto \exp_\infty(x+k) \mid k \in \mathbb{N}\}$.

Nous prenons maintenant en compte le fait que l'on effectue les calcul sur une sous-algèbre de la signature. Nous supposons donc que l'on dispose d'une algèbre de constructeurs $\mathbb{A}$ incluse dans la signature.

Proposition 3.1.11. Étant donné une classe $\mathbb{C}$ de fonctions qui capture les polynômes, supposons que nous disposons d'une fonction $\phi \in \mathbb{C}$ telle que pour tout terme $t \in \mathcal{T}(\mathbb{A})$, nous avons l'inégalité : $[t] \leq \phi(|t|)$. Alors, il existe une fonction $\phi' \in \mathbb{C}$ telle que pour tout n-uplet $(t_1,\cdots,t_n) \in \mathcal{T}(\mathbb{A})$ et tout symbole f, on a

(i) $[f(t_1,t_2,\cdots,t_n)] \leq \phi'(|t_1,\ldots,t_n|)$. En conséquence de quoi, la longueur de dérivation est bornée par ϕ' ;

(ii) si $f(t_1,t_2,\cdots,t_n) \xrightarrow{*}_{\mathcal{R}} v$, alors $|v| \leq \phi'(|t_1,\ldots,t_n|)$.

Démonstration. (i) est une conséquence immédiate du concept de classe de fonctions polynômes-capturante. (ii) est une conséquence immédiate de (i) et de la proposition 3.1.9. □

3.2 Classification des systèmes de réécriture

Pour classifier les systèmes de réécriture, nous rangeons les polynômes suivant trois catégories. À cela, nous ajoutons une classe pour l'exponentielle.

Définition 3.2.1. Une fonction $P(x_1,\cdots,x_n)$ est dite de
genre 0 si P est un polynôme de la forme $\sum_{i=1}^{n} x_i + b$;
genre 1 si P est un polynôme de degré égal à 1 en toutes les variables ;
genre 2 si P est un polynôme ;
genre 3 si P est une fonction de la forme $Q(x_1,\cdots,x_n,2^{x_1},\cdots,2^{x_n})$ où Q est un polynôme.

Cette définition classe toutes les fonctions que nous utilisons pour les interprétations. Notons toutefois que si une interprétation est de genre i, elle est de genre j pour tout $j \leq i$.

Définition 3.2.2 (Classification des systèmes). Un système de réécriture $(\mathcal{R},\mathbb{A},\mathbb{F},f,\mathtt{inp},\mathtt{out},[])$ est dit de genre $\Delta(i)$ ($i \in \{0,1,2\}$) si l'interprétation est polynomiale et si tout symbole de $\mathbb{A}$ est de genre i. Le système est dit de genre $\Delta(3)$ si l'interprétation est exponentielle. Un système de genre $\Delta(i)$ qui de surcroît est confluent est dit de type $\Pi(i)$. Par extension, une fonction sur une algèbre $\mathcal{A}$ est dite de genre $\Delta(i)$ (resp. $\Pi(i)$) si elle est calculée par un système de réécriture de genre $\Delta(i)$ (resp. $\Pi(i)$).

Exemple 3.2.3. Étant donné l'algèbre de calcul $\mathtt{Nat} = \{0,s\}$, nous considérons le système de réécriture suivant. Prenons comme algèbre de constructeurs l'ensemble $\{s,q,0\}$ et ajoutons les symboles $\{+,\times,\mathtt{E}\}$ avec $\mathbf{ar}(0) = 0$, $\mathbf{ar}(s) = \mathbf{ar}(q) = \mathbf{ar}(\mathtt{E}) = 1$ et $\mathbf{ar}(+) = \mathbf{ar}(\times) = 2$. Nous considérons le système de réécriture composé des règles :

$$\begin{array}{lcllcl} 0+y & \to & y & \qquad sx+y & \to & s(x+y) \\ 0\times y & \to & 0 & \qquad sx\times y & \to & y+(x\times y) \\ \mathtt{E}(0) & \to & s(0) & \qquad \mathtt{E}(qx) & \to & \mathtt{E}(x)+\mathtt{E}(x) \end{array}$$

À ce système, nous pouvons attribuer une interprétation polynomiale :

$$\begin{array}{rclcrcl} [0] & = & 1 & \qquad & [+](x,y) & = & 2x+y \\ [s](x) & = & x+1 & \qquad & [\times](x,y) & = & 3xy \\ [q](x) & = & 3x+3 & \qquad & [\mathtt{E}](x) & = & x+3 \end{array}$$

En observant que le système de réécriture est confluent, nous pouvons noter que :

- la fonction produit $(\underline{n},\underline{m}) \mapsto \underline{nm}$ est calculée par $\times$. On choisit pour cela `inp` = `out` comme étant les identités de `Nat`. La fonction produit est donc dans $\Pi(0)$;
- la fonction exponentielle $\underline{n} \mapsto \underline{2^n}$ est calculée par `E`. On remarque pour cela que $\mathtt{E}(q^n(0)) \to s^{2^n}(0)$. Nous choisissons alors :

`inp` :	`Nat`	$\to$	$\{0,s,q\}$	`out` :	`Nat`	$\to$	$\{0,s,q\}$
	0	$\mapsto$	0		0	$\mapsto$	0
	s	$\mapsto$	q		s	$\mapsto$	s

L'exponentielle est donc une fonction $\Pi(1)$.

En fait, et c'est un résultat de Cichon et Lescanne [CL92], on ne peut pas trouver de système de réécriture qui calcule l'exponentielle et pour lequel le codage d'entrée est égal au codage de sortie. La raison est que pour de tels systèmes, la taille du résultat est bornée polynomialement par celle des entrées.

Exemple 3.2.4. En reprenant le système présenté en 1.4.18, nous pouvons observer que la recherche du plus long sous-mot d'un ensemble fini de mots est une fonction $\Delta(0)$. En effet, le système admet l'interprétation polynomiale :

$[a](x) = x+1$	$[g](x) = 5x$	$[\mathtt{r}](x) = x+1$
$[b](x) = x+1$	$[f](x) = 2x$	$[\mathbf{sous}](x) = 5x+2$
$[*](x,y) = x+y+1$	$[d](x) = 2x$	

Théorème 3.2.4.1 (Hiérarchie des systèmes de réécriture). *Les classes de systèmes de réécriture constituent une hiérarchie en termes de complexité échelonnée comme suit :*

Classes de complexité	*Systèmes de réécriture*
PTIME	$\Pi(0)$
ETIME	$\Pi(1)$
E$_2$TIME	$\Pi(2)$
TIME$(\exp_\infty(n+k))$	$\Pi(3)$

Classes de complexité	*Systèmes de réécriture*
NPTIME	$\Delta(0)$
NETIME	$\Delta(1)$
NE$_2$TIME	$\Delta(2)$
NTIME$(\exp_\infty(n+k))$	$\Delta(3)$

Remarque 3.2.5. Premièrement, observons que d'après le théorème, la recherche du plus long sous-mot d'un ensemble de mots est dans NPTIME, le produit est dans PTIME et l'exponentielle dans ETIME.

Deuxièmement, comme nous en avions déjà fait la remarque, le déterminisme de la machine de Turing correspond à la propriété de confluence du système de réécriture. Nous ne ferons les preuves d'équivalence que pour les systèmes déterministes, mais les preuves pour les cas non déterministes s'en déduisent immédiatement.

D'autre part, du fait que NTIME$(C(n)) \subset$ TIME$(k^{C(n)})$, nous pouvons immédiatement déduire l'égalité entre les classes : TIME$(O(\exp_\infty(n+k)) =$ NTIME$(O(\exp_\infty(n+k))$. En conséquence de quoi, les fonctions de $\Delta(3)$ sont également dans $\Pi(3)$.

Enfin, notons que cette hiérarchie peut être étendue en enrichissant le champ des interprétations à des fonctions plus grandes que l'exponentielle ; une telle extension est suggérée par Hofbauer dans sa thèse [Hof92a].

Les deux sous-sections suivantes sont consacrées à la preuve du théorème.

3.2.1 Bornes sur la longueur de dérivation et la taille des termes

Proposition 3.2.6 (Lemme fondamental). Étant donné un système de réécriture $(\mathcal{R},\mathbb{A},\mathbb{F},f,[])$, il existe $c \in \mathbb{N}$ tel que pour tout $t \in \mathcal{T}(\mathbb{A})$:

0. $[t] \leq c.|t|$ si le système est $\Delta(0)$ (ou $\Pi(0)$) ;
1. $[t] \leq 2^{c.|t|}$ si le système est $\Delta(1)$ (ou $\Pi(1)$) ;
2. $[t] \leq 2^{2^{c.|t|}}$ si le système est $\Delta(2)$ (ou $\Pi(2)$) ;
3. $[t] \leq \exp_\infty(|t|+c)$ si le système est $\Delta(3)$ (ou $\Pi(3)$).

Démonstration. Notons tout d'abord que les interprétations sont toujours strictement positives. Pour cette raison, pour toutes les inégalités qui suivent, nous ferons l'hypothèse que les variables sont strictement positives. Avant d'entamer la démonstration proprement dite, nous montrons le lemme suivant qui nous servira pour les interprétations exponentielles.

Lemme 3.2.7. Pour tout $k > 2$, nous définissons la fonction $t(k,n)$ par :

$$\begin{aligned} t(k,0) &= 1 \\ t(k,n+1) &= 2^{kt(k,n)} \end{aligned}$$

Nous avons l'inégalité :

$$t(k,n) \leq \exp_\infty(2k+n)$$

Preuve du lemme. Nous écrivons pour tout $m \leq n$:

$$t(k,n) = 2^{2^{\cdot^{\cdot^{\cdot^{2^{u_m t(k,n-m)}}}}}} \Bigg\} \; m \text{ étages}$$

Nous avons $u_1 = k$ et $u_{m+1} = k + \log(u_m)/2^k$ pour $m < n$. Nous observons que

$$t(k,n) = 2^{2^{\cdot^{\cdot^{\cdot^{2^{u_n}}}}}} \Bigg\} \; n \text{ étages}$$

Il suffit alors de vérifier que pour tout m, $u_m \leq 2k$. On conclut en notant que $u_n \leq \exp_\infty(2k)$. □

0. Supposons que le système soit $\Delta(0)$; nous écrivons les interprétations des constructeurs comme suit : $[g](x_1, \cdots, x_{n_g}) = b_g + \sum_{i=1}^{n_g} x_i$. Il suffit alors de prendre $c = \max_{g \in \mathbb{A}}(b_g + n_g)$. L'inégalité vient par induction sur les termes.
1. Si le système est $\Delta(1)$, nous pouvons écrire les polynômes de l'algèbre de constructeurs sous la forme

$$[g](x_1, x_2, \cdots, x_n) = b_g + \sum_{I_g \subset \{1..n_g\}} a_{I_g} \prod_{i \in I_g} x_i$$

Notons $m = \max_{g \in \mathbb{A}, I_g \subset \{1..n_g\}}(a_{I_g}, b_g)$. Nous pouvons alors observer que :

$$\begin{array}{rcl} [g](x_1, x_2, \cdots, x_n) & \leq & m(1 + 2^n . \prod_{i=1}^n x_i) \\ & \leq & m(2^{n+1} \prod_{i=1}^n x_i) \end{array}$$

En écrivant $c = \log(m) \max_{g \in \mathbb{A}}(n_g + 1)$, nous montrons par induction que $[t] \leq 2^{c.|t|}$. Tout d'abord, l'inégalité est vraie pour les constantes. Supposons maintenant que $t = g(t_1, \cdots, t_n)$, nous avons alors :

$$\begin{array}{lll} [g(t_1, \cdots, t_n)] & = [g]([t_1], \cdots, [t_n]) & \\ & \leq m.2^{(n_g+1)} \prod_{i=1}^n [t_i] & \text{d'après l'inégalité ci-dessus} \\ & \leq m.2^{(n_g+1)} \prod_{i=1}^n 2^{|t_i|} & \text{par induction} \\ & \leq 2^{c(1+\sum_{i=1}^n |t_i|)} & \\ & \leq 2^{c|g(t_1, \cdots, t_n)|} & \end{array}$$

2. Si le système est $\Delta(2)$, nous pouvons écrire les polynômes sous la forme :

$$[g](x_1, x_2, \cdots, x_n) = b_g + \sum_{I_g \subset \{1..n_g\}} a_{I_g} \prod_{i \in I_g} x_i^{\beta_i}$$

Dès lors, en écrivant $B = \max_{g \in \mathbb{A}, i \in I_g}(\beta_i)$ et $m = \max_{g \in \mathbb{A}, I_g \subset \{1..n_g\}}(a_{I_g}, b_g)$, nous avons :

$$[g](x_1, x_2, \cdots, x_n) \leq m.2^{n+1} . \prod_{i=1}^n x_i^B$$

Nous posons $c = \max(\log(m) \max_{g \in \mathbb{A}}(n_g + 1), B, 2)$. Nous procédons par induction. Notons que l'inégalité $[t] \leq 2^{2^{c.|t|}}$ est réalisée pour les constantes. Supposons maintenant que $t = g(t_1, \cdots, t_n)$, nous avons :

$$\begin{array}{lll} [g(t_1, \cdots, t_n)] & \leq m.2^{n+1}.\prod_{i=1}^{n}[t_i]^B & \text{d'après l'inégalité ci-dessus} \\ & \leq 2^{c+c\sum_{i=1}^{n} 2^{c|t_i|}} & \text{par induction} \\ & \leq 2^{c+c2\sum_{i=1}^{n} c|t_i|} & \text{par convexité de l'exponentielle} \\ & \leq 2^{2^{c(1+\sum_{i=1}^{n}|t_i|)}} & \text{car } 2c \leq 2^c \\ & \leq 2^{2^{c|g(t_1,\cdots,t_n)|}} & \end{array}$$

3. Supposons que le système est $\Delta(3)$. Les interprétations des constructeurs sont de la forme :

$$[g](x_1, \cdots, x_n) = P(x_1, \cdots, x_n, 2^{x_1}, \cdots, 2^{x_n})$$

Comme l'exponentielle domine les polynômes, il existe une constante c telle que

$$[g](x_1, \cdots, x_n) \leq 2^{c\sum_{i=1}^{n} x_i}$$

On choisit plus précisément c en sorte que

(a) $c > 1$,

(b) l'inégalité précédente soit vraie pour tous les symboles constructeurs,

(c) pour toutes les constantes $[f] \leq c$.

On montre par induction que $[t] \leq t(c,|t|)$. L'égalité est réalisée pour les constantes. Pour un terme $f(t)$, on a

$$\begin{array}{rcl} [f(t)] & = & [f]([t]) \\ & \leq & 2^{c\sum_{i=1}^{n} t(c,|t_i|)} \\ & \leq & 2^{ct(c,\sum_{i=1}^{n}|t_i|)} \\ & \leq & t(c,|f(t)|) \end{array}$$

Le lemme 3.2.7 permet alors de conclure : $[t] \leq \exp_\infty(|t| + 2c)$.

□

Proposition 3.2.8 (Simulation des systèmes de réécriture). Étant donné une classe de fonctions $\mathbb{C}$ qui capture les polynômes et une fonction ϕ de cette classe, si un système de réécriture vérifie les deux hypothèses suivantes :

- la longueur de dérivation des termes est bornée par ϕ,
- la taille des termes obtenus dans la dérivation est bornée par ϕ,

alors, il existe une fonction ϕ' dans $\mathbb{C}$ et une machine de Turing qui travaille en temps borné par ϕ' et qui simule le système de réécriture.

Démonstration. Premièrement, eu égard au fait que la classe de fonctions $\mathbb{C}$ capture les polynômes, nous pouvons choisir notre modèle de machine de Turing. En effet, les simulations d'un type de machine par un autre sont (toujours) polynomiales. (En particulier, le passage du modèle à deux rubans au modèle à un ruban est quadratique).

Nous utilisons donc une machine à deux piles, la première (dite *principale*) code le terme, la seconde sert pour compter les parenthèses ; enfin l'état de la machine indique la tâche courante : reconnaissance d'un redex, réécriture d'un redex, etc. Pour coder les termes, nous utilisons l'alphabet de l'algèbre et ajoutons les parenthèses pour écrire les termes en notation polonaise. Ainsi, le terme $f(x,y)$ sera écrit sur le ruban $f(x)(y)$. Notons que la taille de ce codage est bornée linéairement par rapport à la taille du terme. Ensuite, pour marquer les symboles des redex, nous utilisons une copie de l'algèbre : les lettres primées sont celles qui font partie d'un redex. Ainsi, le terme $s(0)+0$ de l'exemple 3.2.3 sera noté sur le ruban $+'(s(0))(0')$ (après avoir reconnu le redex). L'intérêt d'utiliser une deuxième pile est que la recherche d'un sous-terme se fait alors en temps linéaire : en empilant les parenthèses rencontrées, on peut savoir dans quelle partie du terme on se trouve.

Nous faisons ici l'hypothèse qu'il n'y a pas de stratégie dans la réduction pour le système de réécriture comme c'est le cas pour notre sémantique des calculs (à l'aide de systèmes confluents ou non). Nous décrivons ici l'algorithme pour les systèmes confluents ; la machine correspondant aux systèmes non confluents est très semblable et nous laissons le lecteur adapter lui-même la description ci-dessous.

L'alphabet de la machine de Turing qui simule le système de ré-

écriture est donc $\Sigma = \mathbb{A} \cup \mathbb{A}' \cup \{(,)\}$. L'algorithme est en clair :

```
entrée(terme t); tant qu'il y a des redex sur le ruban
faire
        chercher un redex;
        réécrire le redex;
        fintantque
retourner le ruban principal;
fin.
```

Réécrire le redex se fait en temps linéaire ; pour ce qui est de la procédure de recherche, l'algorithme est :

```
chercher un redex(terme t)
caractère-courant := racine(t);
trouve-redex := faux;
tant que ((caractère-courant ≠ caractère-final) et
(trouve-redex = faux)) faire
        pour chaque règle de l → r ∈ R faire
                si le caractère-courant est tête de redex
pour la règle l → r
                        alors
                        primer tous les caractères du
redex;
                        trouve-redex := vrai;
                        fsi
        si (trouve-redex = faux) alors
caractere-courant := caractere-suivant;
        finpour
fintantque
fin.
```

La vérification qu'un caractère est bien la tête d'un redex pour la règle $l \to r$ se fait par simple unification symbole par symbole, donc en temps linéaire en la taille du ruban.

Par conséquent, la complexité de la recherche d'un redex peut être estimée à $|\mathcal{R}| * n * n = O(n^2)$ où n est la longueur de l'entrée ; en outre, la recherche nécessite n cases sur le deuxième ruban pour la

vérification que le caractère courant est la tête d'un redex. À cela, il faut rajouter $O(n)$ étapes pour la réécriture des redex. Par conséquent, chaque étape demande un temps de $O(n^2)$ et un espace $O(n)$. La simulation complète d'une dérivation prend donc de l'ordre de $\phi(n) * \phi(n)^2$ où n est la longueur de la chaîne initiale. Comme $\mathbb{C}$ capture les polynômes, il existe une fonction ϕ' qui capture la fonction $\phi(n)^3$ et, par là même, qui borne le temps (et l'espace) de calcul de la machine. □

Corollaire 3.2.9. En se rappelant la proposition 3.2.6, nous déduisons que les fonctions calculées par des systèmes de réécriture $\Pi(i)$ et $\Delta(i)$ sont inclus dans les classes énoncées dans le théorème 3.2.4.1.

3.2.2 Simulation des machines de Turing par la réécriture

Afin de rendre plus modulaire la simulation des machines par la réécriture, nous nous proposons de montrer le lemme suivant :

Lemme 3.2.10 (Lemme de composition faible). Supposons que la fonction $\phi : \mathcal{A}^n \to \mathcal{A}$ soit calculée par un système de réécriture $(\mathcal{R}_i,\mathbb{A}_i,\mathbb{F}_i,f_i,[]_i,\mathtt{inp}_i,\mathtt{out}_i)$ dans $\Delta(i)$ (respectivement $\Pi(i)$) pour $i \in \{0,1,2\}$ et que la fonction $\psi : \mathcal{A}^{m+1} \to \mathcal{A}$ soit calculée par un système $(\mathcal{R},\mathbb{A},\mathbb{F},f,[],\mathtt{inp},\mathtt{out})$ dans $\Delta(0)$ (resp. $\Pi(0)$). Alors, il existe un système $(\mathcal{R}',\mathbb{A}',\mathbb{F}',f',[]',\mathtt{inp}',\mathtt{out}')$ dans $\Delta(i)$ (resp. $\Pi(i)$) qui calcule la fonction composée

$$\begin{array}{rccl} \psi \circ \phi : & \mathcal{A}^{n+m} & \to \mathcal{A} \\ & (t_1,t_2,\cdots,t_{n+m}) & \mapsto \psi(\phi(t_1,\cdots,t_n),t_{n+1},t_{n+m}) \end{array}$$

Démonstration. Sans perte de généralité, nous pouvons supposer que les signatures $\mathbb{A}_i, \mathbb{F}_i, \mathbb{A}, \mathbb{F}$ sont disjointes deux à deux. Nous construisons le système de réécriture suivant. Prenons $\mathbb{A}' = \mathbb{A} \cup \mathbb{A}_i$ et $\mathbb{F}' = \mathbb{F} \cup \mathbb{F}_i \cup \{\mathtt{Tr}, f'\}$. Le point clef est de transformer le résultat du calcul de f_i en son correspondant dans le système $\mathcal{R}$. Pour cela, nous utilisons le nouveau symbole $\mathtt{Tr}$ avec les règles de réécriture :

$$\mathtt{Tr}(g(t_1,\cdots,t_n)) \to (\mathtt{inp} \circ \mathtt{out}_i^{-1})(g)(\mathtt{Tr}(t_1),\cdots,\mathtt{Tr}(t_n))$$

pour chaque symbole de $\mathbb{A}_i$ dans l'image de out.

Pour donner une interprétation au symbole Tr, nous écrivons tout d'abord les polynômes $[g]$ pour $g \in \mathbb{A}$ sous la forme $[g](x_1, \cdots, x_{n_g}) = b_g + \sum_{i=1}^{n_g} x_i$. Nous posons $c = \max_{g \in \mathbb{A}}(n_g + b_g)$. Nous observons alors qu'en prenant $[\mathtt{Tr}](x) = cx + 1$, l'interprétation respecte les règles ci-dessus. En effet :

$$\begin{aligned}
[\mathtt{Tr}(g(t_1, \cdots, t_n))] &= c[b_g] + c\textstyle\sum_{i=1}^{n}[t_i] + 1 \\
&> c + c\textstyle\sum_{i=1}^{n}[t_i] \\
&> n + b_{(\mathtt{inp} \circ \mathtt{out}_i^{-1})(g)} + c\textstyle\sum_{i=1}^{n}[t_i] \\
&> [(\mathtt{inp} \circ \mathtt{out}_i^{-1})(g)(\mathtt{Tr}(t_1), \cdots, \mathtt{Tr}(t_n))]
\end{aligned}$$

Enfin, il faut ajouter une règle pour initier le calcul :

$$f'(x_1, x_2, \cdots, x_{n+m}) \to f(\mathtt{Tr}(f_i(x_1, \cdots, x_n)), x_{n+1}, \cdots, x_{n+m})$$

pour laquelle nous prenons

$$[f'](x_1, x_2, \cdots, x_{n+m}) = [f(\mathtt{Tr}(f_i(x_1, \cdots, x_n)), x_{n+1}, \cdots, x_{n+m})] + 1$$

Les autres interprétations restent inchangées. □

Nous nous attachons maintenant à la simulation des machines de Turing par les systèmes de réécriture proprement dite. Le principe global est de faire la simulation de la machine de Turing relativement à un compteur que nous appelons *horloge* : pour chaque transition, nous faisons décroître l'horloge d'une unité ; ceci nous permet de donner une preuve de terminaison par interprétation polynomiale au système de réécriture.

La proposition 3.2.11 décrit la simulation de chaque transition (relativement à l'horloge). La proposition 3.2.12 montre comment construire une horloge de taille suffisante pour que la simulation de la machine puisse être entièrement accomplie.

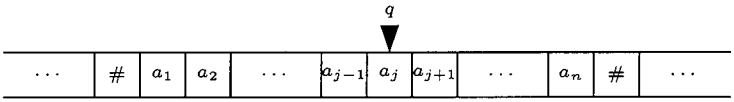
Proposition 3.2.11. Soit une machine de Turing $\mathcal{M} = (L,Q,q_0,q_f,\delta : L \times Q \times L \times Q \times \{-,0,+\})$ qui calcule une fonction $\phi : \mathcal{A}^n \to \mathcal{A}$ en un temps donné par $t(n)$ où n est la taille des entrées. Nous pouvons construire un système de réécriture dans $\Pi(0)$ (si la machine est déterministe) ou dans $\Delta(0)$ (si la machine ne l'est pas) qui calcule la fonction donnée par :

$$\begin{aligned} &\phi' : \mathtt{Nat} \times \mathbb{A}^n \to \mathbb{A} \\ &\underline{m},t_1,\cdots,t_n \mapsto \phi(t_1,\cdots,t_n) \qquad \text{si } m > 5|t_1,\cdots,t_n| \end{aligned}$$

(la constante 5 est un artifice technique qui sera expliqué plus tard).

Démonstration. Nous faisons la preuve pour les machines non déterministes. Le cas des machines déterministes est très similaire. Nous choisissons comme algèbre de constructeurs $\mathbb{A} = L \cup \{s,\epsilon\}$. Les symboles sont d'arité 1 sauf ϵ qui est une constante. En plus de cela, nous avons besoin des symboles $\mathbb{F} = Q \cup \{f\}$. Le symbole f est unaire tandis que les symboles $q \in Q$ sont ternaires.

Une configuration de la machine de Turing :



est représentée par le terme :

$$q(t,a_j a_{j-1} \cdots a_1 \# \epsilon, a_{j+1} \cdots a_m \# \epsilon)$$

Le premier sous-terme « t » est utilisé pour assurer la terminaison du système. Il n'a pas proprement dit de rôle calculatoire. Nous introduisons les règles de réécriture qui simulent les transitions :

$$\begin{aligned} q(s(t),a_1(w_1),w_2) &\to q'(t,a_1'(w_1),w_2) && \text{si } \delta(a_1,q) = (a_1',q',0) \\ q(s(t),a_1(w_1),a_2(w_2)) &\to q'(t,a_2(a_1'(w_1)),w_2) && \text{si } \delta(a_1,q) = (a_1',q',+1) \\ q(s(t),a_1(w_1),w_2) &\to q'(t,w_1,a_1'(w_2)) && \text{si } \delta(a_1,q) = (a_1',q',-1) \end{aligned}$$

Les règles qui gèrent le bord du ruban :

$$q(s(t),\epsilon,w_2) \rightarrow q'(t,\#(\epsilon),w_2) \tag{3.1}$$

$$q(s(t),w_1,\epsilon) \rightarrow q'(t,w_1,\#(\epsilon)) \tag{3.2}$$

$$q(s(t),a_1(\epsilon),w_2) \rightarrow q'(t,\#(\epsilon),a_1'(w_2)) \text{ si } (q,a_1,q',a_1',-1) \in \delta \tag{3.3}$$

Une règle initie le calcul :

$$f(t,a_1(w)) \rightarrow q_0(t,a_1(\epsilon),w)$$

Et pour terminer le calcul :

$$q_f(s(t),a_1(w_1),w_2) \rightarrow q_f(t,w_1,a_1(w_2)) \tag{3.4}$$

$$q_f(s(t),\epsilon,w_2) \rightarrow w_2 \tag{3.5}$$

En observant les règles ci-dessus, nous pouvons remarquer que le système simule la machine de Turing pendant un temps déterminé par le premier argument. Dès lors, il nous suffira de « brancher » un nombre suffisant de « s » pour simuler la machine. La constante 5 introduite plus haut s'explique maintenant. Elle sert à prendre en compte l'ajout de nouvelles lettres sur le ruban par les règles 3.1, 3.2, 3.3, 3.4 et 3.5 dont le nombre d'applications est borné par $t(n)$. Nous sommes par conséquent assurés que si le premier argument vaut $\underline{m}$ avec $m > 5|t_1,\cdots,t_n|$, la simulation sera menée jusqu'au bout du calcul.

Les morphismes de codage sont tout simplement les identités. Il ne reste plus qu'à donner une interprétation aux symboles. Nous prenons :

$$[s](x) = x+1 \qquad [\epsilon] = 1$$
$$[q](t,x,y) = 3t+x+y \qquad [a_i](x) = x+1$$
$$[f](t,x) = 3t+x+4$$

□

Proposition 3.2.12.

0. Pour tout polynôme P donné, la fonction $\underline{n} \mapsto \underline{P(n)}$ est dans $\Pi(0)$ (et donc a fortiori dans $\Delta(0)$) ;
1. pour tout $c \in \mathbb{N}$, la fonction $\underline{n} \mapsto \underline{2^{c.n}}$ est dans $\Pi(1)$ (et $\Delta(1)$) ;
2. pour tout $c \in \mathbb{N}$, la fonction $\underline{n} \mapsto \underline{2^{2^{c.n}}}$ est dans $\Pi(2)$ (et $\Delta(2)$) ;
3. pour tout $c \in \mathbb{N}$, la fonction $\underline{n} \mapsto \underline{\exp_\infty(n+c)}$ est dans $\Pi(3)$ (et $\Delta(3)$).

Démonstration.

0. Dans l'exemple 3.2.3, nous avons vu que lon pouvait définir l'addition et la multiplication ; en les composant (ce qui est possible d'après le lemme de composition faible 3.2.10), nous obtenons les polynômes. Les morphismes de codage sont l'identité.
1. Pour l'exponentielle, nous rajoutons le symbole C au système de réécriture de l'exemple 3.2.3 et les règles :

$$C(0) \to 0$$
$$C(qx) \to \underbrace{q(q(\cdots q(}_{c \text{ fois } q} C(x)))$$

Nous disposons d'une interprétation pour C :

$$[C](x) = \underbrace{[q] \circ \cdots \circ [q]}_{c \text{ fois } [q]}(x) + 1$$

L'exponentielle est alors calculée par le symbole 2^c pour lequel on introduit la règle $2^c(x) \to \mathtt{E}(C(x))$; une interprétation de 2^c est : $[2^c](x) = [\mathtt{E}(C(x))] + 1$.
Les morphismes de codage sont :

inp :	Nat	$\to$	$\{0,s,q\}$	out :	Nat	$\to$	$\{0,s,q\}$
	0	$\mapsto$	0		0	$\mapsto$	0
	s	$\mapsto$	q		s	$\mapsto$	s

2. Pour la double exponentielle, nous utilisons également le symbole C et le système de l'exemple 3.2.3. Nous y ajoutons un symbole DE, unaire, et les deux règles :

$$\begin{aligned} &\mathtt{DE}(0) \to s(s(0)) \\ &\mathtt{DE}(qx) \to \mathtt{DE}(x) \times \mathtt{DE}(x) \end{aligned}$$

Cette fois-ci, l'interprétation polynomiale est :

$$[\mathtt{DE}](x) = x + 3 \qquad [q](x) = 3(x+3)^2 + 1$$

Les interprétations des autres symboles restent inchangées (le lecteur scrupuleux remarquera toutefois que l'interprétation de C dépend de celle de q). Les morphismes de codage sont ceux du cas 1.

3. Pour les tours, nous rajoutons au système 3.2.3 un symbole constructeur r, un symbole Tr de traduction et le symbole t_c qui calcule la tour (en partant de la constante c). Les règles sont les suivantes :

$$\begin{array}{rcl@{\qquad}rcl} \mathtt{Tr}(0) & \to & 0 & t_c(0) & \to & s^{\exp_\infty(c)}(0) \\ \mathtt{Tr}(sx) & \to & q\mathtt{Tr}(x) & t_c(rx) & \to & \mathtt{E}(\mathtt{Tr}(t_c(x))) \end{array}$$

Nous avons pour ce système l'interprétation (exponentielle) :

$$[r](x) = 3^{x+\exp_\infty(c)+1} + 3 \qquad [\mathtt{Tr}(x)] = 3^x$$
$$[t_c](x) = x + \exp_\infty(c) + 1$$

Les morphismes de codage sont ceux du cas 1.

□

Corollaire 3.2.13. Nous déduisons du lemme 3.2.10 et de la proposition 3.2.12 les quatre inclusions suivantes :

0. les fonctions définies par des machines dans PTIME (resp. NPTIME) sont dans Π_0 (resp. Δ_0) ;
1. les fonctions définies par des machines dans ETIME (resp. NETIME) sont dans Π_1 (resp. Δ_1) ;

2. les fonctions définies par des machines dans E_2TIME (resp. NE_2TIME) sont dans Π_2 (resp. Δ_2) ;
3. les fonctions définies par des machines dans TIME($\exp_\infty(n+c)$) (resp. NTIME($\exp_\infty(n+c)$)) sont dans Π_3 (resp. Δ_3) ;

3.3 Tentative de caractérisations en espace

Dans cette section, nous raffinons les résultats de la section précédente en étudiant le comportement des systèmes relativement au genre des interprétations des symboles de sortie. En effet, jusque-là, nous n'avons pris en compte que l'interprétation des données. En nous appuyant sur la stratification des polynômes, nous pouvons observer un lien entre les classes de complexité en espace et les preuves par interprétation polynomiale. Mais pour cela, il est nécessaire de se restreindre aux fonctions définies sur les mots. En effet, il n'y a pas de correspondance intéressante (en termes de complexité) entre la taille des termes et leur interprétation polynomiale, comme le montre l'exemple ci-dessous.

Exemple 3.3.1. Soit la signature $\mathbb{A} = \{0,*\}$ avec $\mathbf{ar}(0) = 0$ et $\mathbf{ar}(*) = 2$. Choisissons l'interprétation comme étant $[0] = 1$ et $[*](x,y) = 2x + y$. Considérons alors les termes :

$$\begin{cases} a_0 = & 0 \\ a_{n+1} = & *(0,a_n) \end{cases} \qquad \begin{cases} b_0 = & 0 \\ b_{n+1} = & *(b_n,0) \end{cases}$$

Nous avons :

$$\begin{aligned} [a_n] &= 2n+1 &&= |a_n| \\ [b_n] &= 2^{n+1}-1 &&= 2^{|b_n|/2+1/2}-1 \end{aligned}$$

Or, et nous le verrons en détail par la suite, il existe un lien direct entre l'interprétation et la taille dans le cas des mots. Nous nous restreignons donc au cas où les constructeurs sont unaires (ou des constantes), leurs interprétations consistent donc en des polynômes à 0 ou 1 variable.

Définition 3.3.2. Nous disons d'un système de réécriture $(\mathcal{R},\mathbb{A},\mathbb{F},f,[],\texttt{inp},\texttt{out})$ qu'il est $\Delta(i,j)$ (où $j \leq i \leq 2$) si les constructeurs dans $\texttt{inp}(\mathcal{A})$ sont de genre au plus i et si les constructeurs dans $\texttt{out}(\mathcal{A})$ sont de genre au moins j. $\Pi(i,j)$ est la classe des systèmes dans $\Delta(i,j)$ qui de surcroît sont confluents. Par extension, nous disons qu'une fonction est de genre $\Delta(i,j)$ (resp. $\Pi(i,j)$) si elle est calculée par un système de genre $\Delta(i,j)$ (resp. $\Pi(i,j)$). Dans la suite de la section, nous ne nous intéressons qu'aux cas où $j > 0$ qui seuls n'ont pas été étudiés.

Proposition 3.3.3. Nous pouvons donner quelques bornes sur la croissance des fonctions calculées par des systèmes dans $\Delta(i,j)$.

- si ϕ est dans $\Delta(i,i)$ (avec $i \in \{1,2\}$), alors il existe une constante c telle que $|\phi(t_1,\cdots t_n)| \leq c|t_1,\ldots,t_n|$;
- si ϕ est dans $\Delta(2,1)$, alors il existe une constante c telle que $|\phi(t_1,\cdots t_n)| \leq 2^{c|t_1,\cdots,t_n|}$.

La preuve se trouve en filigrane dans l'article de Cichon et Lescanne [CL92]. Elle repose essentiellement sur le lemme suivant :

Lemme 3.3.4. Étant donné une algèbre $\mathbb{A}$ d'un système de réécriture $(\mathcal{R},\mathbb{A},\mathbb{F},f,\texttt{inp},\texttt{out},[])$, il existe une constante $d \in \mathbb{N}$ telle que pour tout terme $t \in \mathcal{T}(\texttt{out}(\mathcal{A}))$,

- $2^{d|t|} \leq [t]$ si le système est $\Delta(1,1)$;
- $2^{d|t|} \leq [t]$ si le système est $\Delta(2,1)$;
- $2^{2^{d|t|}} \leq [t]$ si le système est $\Delta(2,2)$.

La preuve est très similaire à celle de la proposition 3.2.6 ; il suffit de changer le sens des inégalités et max en min.

Preuve de la proposition 3.3.3. Pour simplifier, nous supposons que le symbole de calcul f du système de réécriture est unaire. Le lecteur adaptera facilement la preuve au cas général. Nous supposons donc que nous diposons d'une dérivation $f(t)\xrightarrow{!}_{\mathcal{R}} t'$ où $t \in \texttt{inp}(\mathcal{A})$ et $t' \in \texttt{out}(\mathcal{A})$.

1. Pour un système dans $\Delta(1,1)$, d'après le lemme précédent et la proposition 3.2.6, nous avons les inégalités : $2^{d|t'|} \leq [t'] \leq [t] \leq 2^{c|t|}$. En prenant le logarithme, il vient : $|t'| \leq \frac{c}{d}|t|$;

2. pour un système dans $\Delta(2,2)$, nous procédons de même : $2^{2^{d|t'|}} \leq [t'] \leq [t] \leq 2^{2^{c|t|}}$. On obtient alors : $|t'| \leq \frac{c}{d}|t|$;
3. pour un système dans $\Delta(2,1)$, $2^{d|t'|} \leq [t'] \leq [t] \leq 2^{2^{c|t|}}$. D'où l'on conclut : $|t'| \leq \frac{2^{c|t|}}{d}$

□

Nous pourrions alors imaginer utiliser ces bornes en espace pour caractériser les systèmes de réécritures. Par exemple, nous allons voir que les systèmes $\Delta(1,1)$ calculent les fonctions de LINSPACE. Toutefois, nous ne pouvons pas espérer de caractérisation des modèles en termes d'algorithmes comme dans la section précédente mais seulement des caractérisations extentionnelles des fonctions. L'exemple suivant montre le cas d'un système dans $\Delta(1,1)$ pour lequel il est nécessaire de passer par certains termes de taille exponentielle au cours de la normalisation. En conséquence de quoi, nous ne pouvons simuler directement le système par une machine dans LINSPACE.

Exemple 3.3.5. Prenons $\mathbb{A} = \mathtt{Nat}$ et $\mathbb{F} = \{f,g,h,a,b\}$ avec $\mathbf{ar}(f) = \mathbf{ar}(g) = \mathbf{ar}(h) = 1$ et $\mathbf{ar}(a) = \mathbf{ar}(b) = 2$. Les règles sont :

$$\begin{array}{rcl@{\qquad}rcl} f(x) & \to & h(g(x)) & & & \\ g(s(x)) & \to & a(g(x),g(x)) & g(0) & \to & b(0,0) \\ h(b(x,x)) & \to & 0 & a(b(x,x),b(x,x)) & \to & b(b(x,x),b(x,x)) \end{array}$$

pour lesquelles nous disposons de l'interprétation polynomiale dans $\Delta(1,1)$:

$$\begin{array}{rcl@{\qquad}rcl} [0] & = & 1 & [g](x) & = & x+2 \\ [s](x) & = & 2x+6 & [a](x,y) & = & x+y+1 \\ [h](x) & = & x+1 & [b](x,y) & = & x+y \\ [f](x) & = & [h(g(x))]+1 & & & \end{array}$$

Le système calcule la fonction $\underline{n} \mapsto \underline{0}$, fonction qui est dans LINSPACE. Pourtant, nous ne pouvons pas simuler le système par une machine de Turing dans LINSPACE car la dérivation de tout terme $f(\underline{n})$ passe

(pour chacune des branches de la dérivation) par le terme $h(t_n)$ de taille exponentielle où t_n est défini par récurrence :

$$\begin{aligned} t_0 &= 0 \\ t_{n+1} &= b(t_n,t_n) \end{aligned}$$

Théorème 3.3.5.1.
Nous avons les inclusions suivantes :

- LINSPACE $\subseteq \Pi(1,1)$,
- LINSPACE $\subseteq \Pi(2,2)$,
- ESPACE $\subseteq \Pi(2,1)$.

De manière analogue, pour les systèmes non déterministes :

- NLINSPACE $\subseteq \Delta(1,1)$,
- NLINSPACE $\subseteq \Delta(2,2)$,
- NESPACE $\subseteq \Delta(2,1)$.

3.3.1 Simulation des machines de Turing

Comme dans la section précédente, pour simuler une machine de Turing, nous procédons en deux étapes. Nous construisons d'abord un système de réécriture qui simule les transitions de la machine relativement à deux paramètres qui correspondent au temps et à l'espace. Ces deux compteurs vont décroître pendant la simulation, la deuxième tâche consiste donc à construire des compteurs suffisamment grands pour que la simulation puisse être achevée.

Nous supposons donc disposer d'une machine de Turing $\mathcal{M} = (L,Q,q_0,q_f,\delta : L \times Q \times L \times Q \times \{-,0,+\})$; nous supposons en outre qu'elle travaille sur un ruban unilatère infini. (Cette restriction ne pose pas de problème eu égard au fait que nous ne considérons que des classes de complexité plus grandes que LINSPACE cf. [HU79]). Nous ne supposons pas la machine déterministe ; le lecteur n'aura pas de

mal à adapter la preuve pour le cas des machines déterministes. Les configurations sont représentées par un terme du genre :

$$\triangleright x_1 x_2 \cdots x_n \# \# \cdots \# q s s \cdots s(\epsilon)$$

La partie $x_1 x_2 \cdots x_n$ représente le ruban, la partie $\# \# \cdots \#$ sert de *tampon* pour contrôler l'espace utile à la machine, q est l'état courant et la partie $ss \cdots s$ décompte les transitions, on appelle cette dernière partie, l'*horloge* de la machine.

La signature est $\Sigma \cup \overline{\Sigma} \cup Q \cup \hat{\Sigma_Q} \cup \check{\Sigma_Q} \cup \overline{\check{\Sigma_Q}} \cup \{\epsilon, s\}$. Les symboles sont tous unaires sauf ϵ qui est une constante. L'ensemble des symboles $\check{\Sigma_Q}$ est composé des lettres $\check{a_q}$ où $a \in \Sigma$ et $q \in Q$. Les autres ensembles $\overline{\Sigma}, \hat{\Sigma_Q}, \cdots$ sont définis de manière similaire.

Nous introduisons les règles qui simulent les transitions de la machine de Turing :

Déplace l'information :

$$\begin{array}{ll} a q s \to \hat{a_q} \hat{q} & \check{a_{q'}} b \to a \check{b_{q'}} \\ b \hat{a_q} \to \hat{b_q} a & \check{a_{q'}} q s \to a q' \end{array}$$

Opère la transition :

$$c \bar{b} \hat{a_q} \to \begin{cases} \bar{c} \bar{b'} \check{a_{q'}} & \text{si } (b,q,b',q',-) \in \delta \\ c \bar{b'} \check{a_{q'}} & \text{si } (b,q,b',q',0) \in \delta \\ c b' \bar{\hat{a}}_{q'} & \text{si } (b,q,b',q',+) \in \delta \end{cases}$$

où a, b, $c \in \Sigma$.

Termine les calculs :

$$\bar{b} \hat{a_{q_f}} \to b a \qquad \hat{q_f} x \to \epsilon$$

Chaque transition « consomme » deux symboles s. Si le nombre de symboles s de l'*horloge* est supérieur à deux fois le temps de l'exécution et que le tampon est de taille suffisante, nous sommes assurés que la forme normale du système est le résultat du calcul de la machine. Il ne reste maintenant qu'à construire les termes initiaux, avec un tampon et une horloge suffisamment longs. Pour LINSPACE, nous obtenons des preuves dans $\Delta(1,1)$ et $\Delta(2,2)$; pour ESPACE, nous disposons d'une preuve dans $\Delta(2,1)$.

Simulation de LINSPACE

Supposons que la machine à simuler soit dans LINSPACE. Il existe donc une constante c telle que le calcul utilise moins de cn cellules où n est la taille de l'entrée ; il existe également une constante k telle que le calcul prend moins de 2^{kn} étapes. Il ne reste donc qu'à construire le terme initial, avec un tampon de taille cn et une horloge de taille 2^{kn}. Nous rajoutons au système présenté ci-dessus les symboles $\Sigma \cup \Sigma' \cup \{f,g,E,H,\blacktriangle,\blacksquare\}$.

Un premier groupe de règles produit le terme :

$$\triangleright x_1 \cdots x_n \blacktriangle \underbrace{EE\cdots E}_{cn \text{ fois } E} \blacksquare \underbrace{HH\cdots H}_{kn \text{ fois } H} f(\epsilon)$$

$$\begin{array}{lll} g \to \triangleright\blacktriangle\blacksquare f & Ha' \to a'H & Ea' \to a'E \\ f\mathbf{a} \to a'E^cH^kf & HE \to EH & \\ \blacksquare E \to E\blacksquare & \blacksquare a' \to a'\blacksquare & \blacktriangle a' \to a\blacktriangle \end{array}$$

Le deuxième groupe transforme les lettres E et H pour le tampon et l'horloge :

$$\begin{array}{llll} f\epsilon \to s\epsilon & Hs \to ssH & E\blacksquare s \to \#s & \blacktriangle\# \to \#\blacktriangle \\ & H\epsilon \to \epsilon & E\# \to \#\# & \blacktriangle s \to q_0 s \end{array}$$

En se rappelant l'exemple 1.4.7, on constate que le système précédent produit le terme :

$$\triangleright x_1x_2\cdots x_n \underbrace{\#\#\cdots\#}_{cn \text{ fois } \#} q \underbrace{ss\cdots s}_{2^{kn} \text{ fois } s}(\epsilon)$$

Pour une interprétation dans $\Delta(1,1)$, nous prenons :

$[\epsilon] = 1$ $[a](x) = 2(x+2)$ $[\overline{a}](x) = 2(x+2)$

$[s](x) = x+2$ $[\check{a_{q'}}](x) = 2(x+1)$ $[\overline{\check{a}_{q'}}](x) = 2(x+1)$

$[q](x) = x+1$ $[\hat{a_q}](x) = 2(x+3)$ $[a'](x) = 3(x+1)$

$[f](x) = x+3$ $[\mathbf{a}](x) = [a']([E^c H^k f](x))$

$[g](x) = [\triangleright\blacktriangle\blacksquare f](x)+1$ $[H](x) = 3x$ $[E](x) = 3x+2$

$[\blacktriangle](x) = (x+1)^2$ $[\blacksquare](x) = 3x+1$

Pour une interprétation dans $\Delta(2,2)$, nous prenons :

$[\epsilon] = 1$ $[a](x) = (x+2)^2$ $[\overline{a}](x) = (x+2)^2$

$[s](x) = x+2$ $[\check{a_{q'}}](x) = (x+1)^2$ $[\overline{\check{a}_{q'}}](x) = (x+1)^2$

$[q](x) = x+1$ $[\hat{a_q}](x) = (x+3)^2$ $[a'](x) = (x+4)^2$

$[f](x) = x+3$ $[\mathbf{a}](x) = [a']([E^c H^k f](x))$

$[g](x) = [\triangleright\blacktriangle\blacksquare f](x)+1$ $[H](x) = (x+1)^2$ $[E](x) = (x+3)^2$

$[\blacktriangle](x) = (x+1)^2$ $[\blacksquare](x) = (x+2)^2$

Simulation de ESPACE

Pour la simulation dans ESPACE, nous pouvons supposer sans perte de généralité que la machine utilise au plus 2^{cn} cellules et $2^{2^{kn}}$ étapes. Nous reprenons alors le premier groupe de règles définies pour la simulation dans LINSPACE et nous modifions les règles du deuxième groupe comme suit :

$f(\epsilon) \to ss\epsilon$ $F(\epsilon) \to \epsilon$ $\epsilon + y \to y$

$F(sx) \to s((x+x)+Fx)$ $sx + y \to s(x+y)$

$Es \to s$ $E\blacksquare s \to \#s$ $\blacktriangle\# \to \#\blacktriangle$

$E\# \to \#\#E$ $\blacktriangle s \to q_0 s$

En se rappelant l'exemple 3.1.6, nous pouvons observer que le système produit le terme voulu :

$$\triangleright x_1 x_2 \cdots x_n \underbrace{\#\# \cdots \#}_{2^{cn} \text{ fois } \#} q \ \underbrace{ss \cdots s}_{2^{2^{kn}} \text{ fois } s} (\epsilon)$$

Il ne nous reste plus qu'à donner une interprétation polynomiale dans $\Delta(2,1)$:

$[\epsilon] = 1$ $\quad [a](x) = 2(x+2)$ $\quad [\overline{a}](x) = 2(x+2)$

$[s](x) = x+2$ $\quad [\tilde{a_{q'}}](x) = 2(x+1)$ $\quad [\tilde{\overline{a}_{q'}}](x) = 2(x+1)]]$

$[q](x) = x+1$ $\quad [\hat{a_q}](x) = 2(x+3)$ $\quad [a'](x) = (x+4)^3$

$[f](x) = x+5$ $\quad [\mathbf{a}](x) = [a']([E^c H^k f](x))$

$[g](x) = [\triangleright \blacktriangle \blacksquare f](x) + 1$ $\quad [+](x,y) = 2x+y$ $\quad [H](x) = (x+1)^3$

$[E](x) = (x+3)^3$ $\quad [\blacktriangle](x) = 2(x+1)$ $\quad [\blacksquare](x) = (x+2)^3$

3.3.2 Les inclusions sont-elles réciproques ?

Problème ouvert 3.3.5.1. *Nous avons montré les inclusions dans un sens en simulant les machines par des systèmes de réécriture. La réciproque reste un problème ouvert.*

Remarque 3.3.6. Néanmoins, voici une piste de travail pour montrer les inclusions de $\Pi(1,1)$ et $\Pi(2,2)$ dans LINSPACE. Supposons qu'une fonction ϕ soit calculée par un système dans $\Pi(1,1)$ (resp. $\Pi(2,2)$). Nous disposons de deux informations sur la fonction : elle est calculée dans ETIME (resp. dans E_2TIME) et elle est bornée polynomialement (sa taille étant bornée linéairement). Elle est donc récursive primitive et bornée par une fonction dans $\mathcal{E}_2$.

Or, pour $n > 2$, nous savons que si f est une fonction récursive primitive et que si $f < g$ où $g \in \mathcal{E}_n$, alors $f \in \mathcal{E}_n$. Pour $n = 2$, le problème reste ouvert. Si la propriété est vraie même dans ce cas, nous avons alors l'inclusion inverse ($\Pi(1,1) \subset$ LINSPACE).

Remarque 3.3.7. Voici un deuxième point de vue. Restreignons-nous aux problèmes de décision. Si $\Pi(1,1) \subset$ LINSPACE (la remarque est également valable pour $\Pi(2,2)$), nous avons alors l'égalité D(LINSPACE) = D(ETIME) (nous rajoutons un « D » pour indiquer que nous ne considérons que des problèmes de décision). En effet, prenons un problème de décision dans D(ETIME), d'après le théorème 3.2.4.1, il peut être calculé par un système de réécriture dans $\Pi(1)$. Mais comme il s'agit d'un problème de décision, les formes normales sont des constantes. Par conséquent, le système est dans $\Pi(1,1)$, c'est-à-dire dans LINSPACE.

3.4 Caractérisations en espace

Pour obtenir des caractérisations en espace des systèmes de réécriture, il nous paraît beaucoup plus fructueux de ne pas considérer les interprétations polynomiales comme des preuves de terminaison (qui concerne plus le temps que l'espace) mais comme un moyen de contrôler la taille des termes.

Dans cette section, nous développons cette idée, ce qui nous permet de définir une hiérarchie de fonctions parallèle à celle du théorème 3.2.4.1. Par ailleurs, nous obtenons une caractérisation de l'espace linéaire.

Définition 3.4.1. Nous disons d'un système de réécriture $(\mathcal{R},\Sigma)$ qu'il admet une *pseudo-preuve de terminaison par interprétation polynomiale* s'il existe pour tout symbole $g \in \Sigma$ un polynome $[g]$ qui vérifie les trois premières hypothèses de la définition 3.1.1 et la quatrième en remplaçant l'inégalité stricte par une inégalité large. Pour toute règle $l \to r$ et toute substitution close σ :

$$[l\sigma] \geq [r\sigma]$$

Par la suite, nous employons l'expression « *pseudo-interprétation polynomiale* » (ou plus simplement « pseudo-interprétation ») pour désigner l'assignation de tels polynômes à un système de réécriture. Dans cette section, $(\mathcal{R},\Sigma,[])$ dénote un système de réécriture muni d'une pseudo-preuve de terminaison par interprétation polynomiale.

Exemple 3.4.2. Bien évidemment, une pseudo-preuve de terminaison n'assure pas la terminaison du système. Il suffit de prendre $\Sigma = \{\bullet,\circ\}$ et les deux règles de réécriture :

$$\bullet \rightarrow \circ$$
$$\circ \rightarrow \bullet$$

pour lequelles nous avons l'interprétation $[\bullet] = [\circ] = 1$.

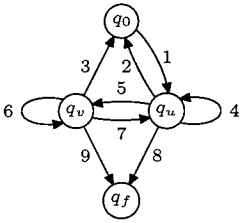
Nous disposons d'un résultat plus précis :

Théorème 3.4.2.1.
Le problème de la terminaison des systèmes de réécriture reste indécidable si l'on se restreint aux systèmes qui admettent une pseudo-preuve de terminaison par interprétation polynomiale.

Démonstration. Nous nous ramenons à un problème d'arrêt. Pour cela, nous nous appuyons sur le lemme suivant :

Lemme 3.4.3. La terminaison des machines de Turing déterministes qui travaillent en espace polynomial (mais qui ne s'arrêtent pas nécessairement) est indécidable.

Preuve du lemme. Nous procédons par réduction du problème de correspondance de Post. Étant donné deux suites de mots $u_1, \cdots u_n$ et $v_1, \cdots, v_n$, nous construisons une machine $\mathcal{M}$ non déterministe dont l'une des branches s'arrête uniquement si le problème de Post a une solution. La considération d'une telle machine est suffisante, en effet, du fait du théorème de Savich, cette machine peut être simulée par une machine qui travaille dans l'espace polynomial, qui est déterministe et qui s'arrête uniquement si le problème de Post a une solution. Nous reprenons maintenant la description de la machine non déterministe. La machine dispose de deux rubans. Le premier contient les mots $u_1, \cdots, u_n, v_1, \cdots, v_n$, le deuxième contient la solution courante. Comme le contenu du premier ruban ne change pas, nous utilisons le mot ruban pour désigner le deuxième ruban. Nous notons u le contenu du ruban dans l'état q_u et v dans l'état q_v.



(1) au départ, la machine choisit un mot u_i et le stocke sur le ruban ;
(2) s'il n'y a pas de mots v_i tels que v_i est un préfixe de u ou u est un préfixe de v_i ;
(3) s'il n'y a pas de mots u_i tels que u_i est un préfixe de v ou v est un préfixe de u_i ;
(4) choisit un mot v_i tel que v_i est un préfixe propre de u, supprime à u son préfixe v_i et stocke le résultat ;
(5) choisit un mot v_i tel que u est un préfixe propre de v_i, supprime à v_i son préfixe u et stocke le résultat ;
(6) choisit un mot u_i tel que u_i est un préfixe propre de v, supprime à v son préfixe u_i et stocke sur ;
(7) choisit un mot u_i tel que v est un préfixe propre de u_i, supprime à u_i son préfixe v et stocke sur ;
(8) choisit un mot v_i tel que $u = v_i$, passe à l'état succès ;
(9) choisit un mot u_i tel que $v = u_i$ et passe à l'état succès.

□

La suite de l'exposé montre comment simuler une machine de Turing qui travaille en espace polynomial à l'aide d'un système de réécriture qui admet une pseudo-preuve de terminaison. Nous pouvons donc réduire le problème de correspondance aux systèmes de réécriture. □

Pour la proposition suivante, nous avons besoin de certaines inégalités établies dans les sections 3.1.2 et 3.2.1 qui restent valables pour les pseudo-interprétations. Nous ne donnons pas les démonstrations vu qu'elles sont très similaires à celles données précedemment.

Lemme 3.4.4. L'inégalité $|t| \leq [t]$ reste valable pour les pseudo-interprétations.

Lemme 3.4.5. L'inégalité $[t] \leq 2^{2^{k|t|}}$ (pour un certain k donné dans la proposition 3.2.6) reste valable pour les pseudo-interprétations polynomiales. Pour les pseudo-interprétations exponentielles, l'inégalité est $[t] \leq \exp_\infty(|t| + k)$.

Proposition 3.4.6. Même si le problème de la terminaison est indécidable en général (théorème 3.4.2.1), nous disposons d'un critère simple qui permet de déterminer la terminaison d'un système de réécriture *sur les mots* (étant donné une pseudo-preuve de terminaison). Il suffit que pour toute règle $l \to r$ du système de réécriture, on ait $[l] \neq [r]$.

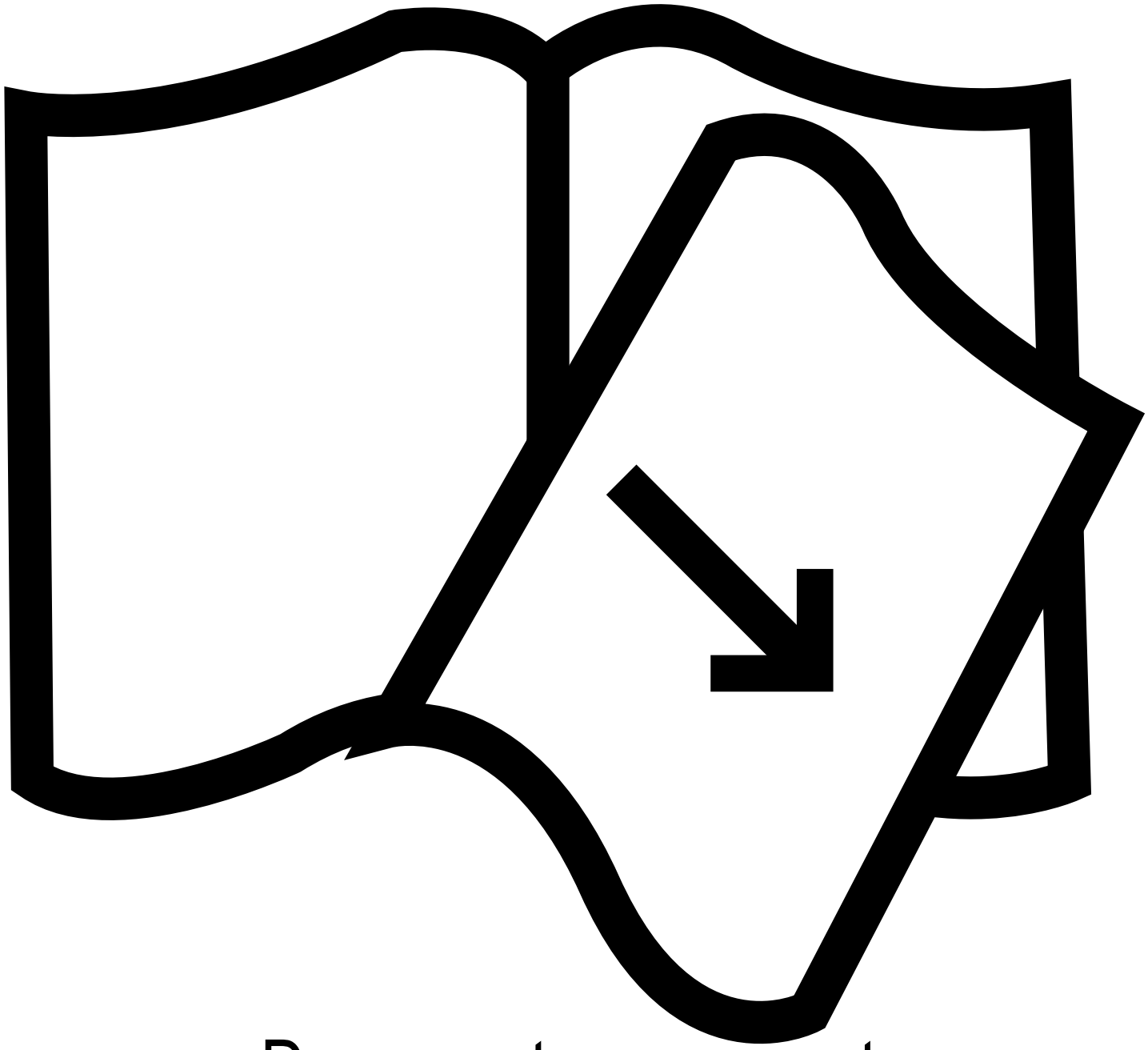
Démonstration. Le principe de la démonstration est de montrer que l'on peut trouver un entier M tel que si le système ne termine pas, alors il existe un terme de taille plus petit que M tel que le système ne termine pas pour ce terme. Pour savoir si le système termine ou non, il suffit alors de vérifier pour tous les termes de taille plus petits que M si le système termine ou non.

Tout d'abord, il nous faut observer que si le système ne termine pas, il existe une dérivation (dite *cyclique*) $t_1 \to t_2 \to \cdots t_n \to t_1$. Nous nous appuyons alors sur le lemme suivant.

Lemme 3.4.7. Il existe (et nous le construisons) un entier M tel que si $[t] > M$, alors si $t \xrightarrow{*}_{\mathcal{R}} t'$ nous avons $[t] > [t']$.

Preuve du lemme. Soit $\mathcal{R} = \{l_1 \to r_1, \cdots, l_k \to r_k\}$ le système de réécriture. Pour $m \leq k$, nous écrivons le polynôme $([l_m] - [r_m])(x) = \sum_{i=0}^{d_m} a_i^m x^i$.

Remarquons que comme $([l_m] - [r_m])(x) \geq 0$, nous sommes assurés que le coefficient de plus haut degré du polynôme $[l_m] - [r_m]$ est positif. En notant $A = \max_{\{m \leq k; i \leq d_m\}}(a_i^m)$, nous pouvons faire les majorations :



Documents manquants
(pages, cahiers .....)

2. l'interprétation de tout symbole de $\mathbb{A}$ est de genre au plus i.

Un système de genre $\Gamma(i)$ qui est confluent est dit de type $\Theta(i)$. Par extension, une fonction est dite de genre $\Gamma(i)$ (resp. $\Theta(i)$) si elle est calculée par un système de genre $\Gamma(i)$ (resp. $\Theta(i)$).

Nous avons un théorème équivalent au théorème 3.2.4.1 mais pour des classes de complexité en espace.

Théorème 3.4.8.1 (Hiérarchie des pseudo-preuves de terminaison). *Les classes de systèmes de réécriture disposant d'une pseudo-interprétation constituent une hiérarchie en termes de complexité échelonnée comme suit :*

Classes de complexité	*Systèmes de réécriture*
PSPACE	$\Theta(0)$, $\Gamma(0)$
ESPACE	$\Theta(1)$, $\Gamma(1)$
E_2SPACE	$\Theta(2)$, $\Gamma(2)$
SPACE$(\exp_\infty(n+k))$	$\Theta(3)$, $\Gamma(3)$

Remarque 3.4.9. En raison du théorème de Savitch, il n'y a pas lieu pour le théorème de faire de distinction entre machines déterministes ou non.

Les deux sous-sections suivantes sont consacrées à la preuve du théorème. Nous reprenons la méthode vue dans la section 3.2

3.4.2 Simulation des systèmes de réécriture

Les propositions suivantes qui ont été établies à la section 3.2 restent valable pour les pseudo-preuves de terminaison.

Proposition 3.4.10. Étant donné une classe de fonction $\mathbb{C}$ qui capture les polynômes et une fonction $\phi \in \mathbb{C}$, supposons que pour tout terme $t \in \mathbb{A}$ d'un système de réécriture $(\mathcal{R},\mathbb{A},\mathbb{F},f,\mathtt{inp},\mathtt{out},[])$ nous ayons $[t] \leq \phi(|t|)$. Il existe alors une fonction $\phi' \in \mathbb{C}$ telle que pour tout n-uplet $(t_1,\cdots,t_n) \in \mathbb{A}$:

(i) $[f(t_1,t_2,\cdots,t_n)] \leq \phi'|t_1,\ldots,t_n|)$.
(ii) si $f(t_1,t_2,\cdots,t_n) \xrightarrow{*}_{\mathcal{R}} v$, alors $|v| \leq \phi'|t_1,\ldots,t_n|)$.

Proposition 3.4.11. Les inégalités établies à la proposition 3.2.6 restent valables.

1. $[t] \leq c.|t|$ si le système est $\Gamma(0)$ (où $\Theta(0)$) ;
2. $[t] \leq 2^{c.|t|}$ si le système est $\Gamma(1)$ (où $\Theta(1)$);
3. $[t] \leq 2^{2^{c.|t|}}$ si le système est $\Gamma(2)$ (où $\Theta(2)$).

Proposition 3.4.12. Étant donné une classe de fonctions $\mathbb{C}$ qui capture les polynômes et une fonction ϕ de cette classe, si un système de réécriture vérifie les deux hypothèses :

- $\mathcal{R}$ termine,
- la taille des termes obtenus dans la dérivation est bornée par ϕ,

alors, il existe une fonction ϕ' dans $\mathbb{C}$ et une machine de Turing qui travaille en espace borné par ϕ' et qui simule le système de réécriture.

Démonstration. Il suffit de reprendre la construction de la proposition 3.2.8 et de remarquer que chaque étape de réécriture se fait en espace linéaire. □

Corollaire 3.4.13. D'après les propositions 3.4.11 et 3.4.12, les systèmes de réécriture $\Theta(i)$ et $\Gamma(i)$ peuvent être simulés dans les classes correspondantes de complexité (présentées au théorème 3.4.8.1).

3.4.3 Simulation des machines de Turing

Dans cette section, nous supposons que nous disposons d'une machine de Turing $\mathcal{M} = (L,Q,q_0,q_f,\delta : L \times Q \times L \times Q \times \{-,0,+\})$ qui utilise $\phi(n)$ cellules pour une donnée de taille n.

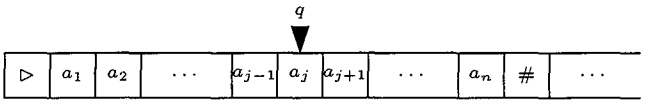
Sans perte de généralité, nous supposons que la machine travaille sur un ruban unilatère. Pour simplifier l'exposé, nous supposons également que la machine calcule une fonction à un seul argument (le lecteur n'aura aucune difficulté à adapter la démonstration aux cas où le nombre d'arguments est supérieur). Enfin, par soucis de concision, nous ne présentons que la simulation de la machine non-déterministe (la transposition au cas déterministe est transparente).

Proposition 3.4.14. Il existe un système de réécriture $(\mathcal{R},L,\mathbb{F},f,\texttt{inp},\texttt{out},[])$ tel que pour tout mot w, la forme normale du terme

$$f(\underbrace{\#(\#\cdots\#}_{\phi(|w|)\text{ fois }\#}(0)\cdots),w)$$

est exactement le résultat de la machine de Turing pour l'entrée w.

Démonstration. Le système de réécriture est composé de la signature $\mathbb{A} = L \cup \{\epsilon\}$ et de $\mathbb{F} = L_Q \cup \{f,\bullet\}$ (tous les symboles sont unaires sauf ϵ qui est une constante). Une configuration de la machine de Turing:



est représentée par le terme:

$$\triangleright a_1 \cdots, a_{j-1} a_{j_q} a_{j+1} \cdots a_m \# \# \cdots \#(\epsilon)$$

où le nombre de # est choisi suffisamment grand au départ pour que la simulation puisse être achevée.

Pour effectuer les transitions de la machine, nous introduisons les règles suivantes (les lettres $a,b,c,\cdots$ sont dans L).

$$ab_qc \rightarrow \begin{cases} a_{q'}b'c & \text{si } (b,q,b',q',-) \in \delta \\ ab'_{q'}c & \text{si } (b,q,b',q',0) \in \delta \\ ab'c_{q'} & \text{si } (b,q,b',q',+) \in \delta \end{cases}$$

Il nous faut ajouter les règles qui construisent le terme initial.

$$f(\#(x),y) \to f(x,\bullet(y)) \qquad \bullet a \to a\bullet$$
$$f(\epsilon,a(x)) \to \rhd a_{q_0}(x) \qquad \bullet(\epsilon) \to \#(\epsilon)$$

Ce système admet une pseudo-preuve de terminaison :

$$[\epsilon] = 1 \qquad [\bullet](x) = x+1 \qquad [f](x) = x+y+1$$
$$[a](x) = x+1 \qquad [a_q](x) = x+1$$

□

Proposition 3.4.15. La proposition 3.2.12 montre que

0. les polynômes sont dans $\Theta(0)$;
1. pour tout $c \in \mathbb{N}$, la fonction $\underline{n} \to \underline{2^{cn}}$ est dans $\Theta(1)$;
2. pour tout $c \in \mathbb{N}$, la fonction $\underline{n} \to \underline{2^{2^{cn}}}$ est dans $\Theta(2)$;
3. pour tout $c \in \mathbb{N}$, la fonction $\underline{n} \to \underline{\exp_\infty(n+c)}$ est dans $\Theta(3)$.

Corollaire 3.4.16. Les classes de complexité définies au théorème 3.4.8.1 peuvent être simulées par des systèmes de réécriture dans les classes correspondantes.

Démonstration. Pour simuler la machine de Turing $\mathcal{M}$, nous utilisons le système $(\mathcal{R},L,\mathbb{F},f,\mathtt{inp},\mathtt{out},[])$ défini à la proposition 3.4.14, auquel nous adjoignons le système défini à la proposition 3.4.15 et la règle :

$$g(w) \to f(k(w),w)$$

où g est un nouveau symbole et k est le symbole qui calcule $\phi(|w|)$.

Ce système admet une pseudo-preuve de terminaison. Il suffit de prendre $[g](x) = [f]([k](x),x)$. □

3.4.4 Caractérisation de LINSPACE

Définition 3.4.17. Un système de réécriture $(\mathcal{R},L,\mathbb{F},f,\mathtt{inp},\mathtt{out},[])$ dans $\Gamma(0)$ est dit linéaire si tout symbole $f \in \mathbb{F}$ est de genre 1.

Exemple 3.4.18. La recherche du plus long sous-mot décrite à l'exemple 1.4.18 est linéaire.

Théorème 3.4.18.1.
La restriction de $\Theta(0)$ *(respectivement* $\Gamma(0)$*) aux systèmes linéaires est équivalente à* LINSPACE *(resp.* NLINSPACE*).*

Remarque 3.4.19. On peut ainsi conclure que la recherche du plus long sous-mot est non seulement dans NPTIME mais qu'en plus elle est dans NLINSPACE.

Le reste de la section est consacré à la preuve du théorème.

Proposition 3.4.20. Les machines de Turing dans LINSPACE peuvent être simulées par des systèmes linéaires.

Démonstration. La raison pour laquelle nous ne réutilisons pas directement le système construit au corollaire 3.4.16 est que le symbole $[g]$ n'est pas de genre 0. Mais dans notre cas, nous pouvons avoir un traitement particulier du tampon. Pour simplifier, nous allons même (sans perte de généralité) nous restreindre à simuler les machines qui travaillent exactement dans l'espace de leurs données. Étant donné une telle machine de Turing $\mathcal{M}$, nous construisons le système de réécriture $(\mathcal{R},L,\mathbb{F},f,\mathtt{inp},\mathtt{out},[])$. $\mathbb{F} = \{f\}$. Les règles sont :

Initie le calcul :

$$f(a(w)) \to \triangleright(a_{q_0}(w))$$

Simule les transitions :

$$ab_qc \to \begin{cases} a_{q'}b'c & \text{si } (b,q,b',q',-) \in \delta \\ ab'_{q'}c & \text{si } (b,q,b',q',0) \in \delta \\ ab'c_{q'} & \text{si } (b,q,b',q',+) \in \delta \end{cases}$$

Termine le calcul :

$$a_{q_f} \to a$$

Ce système admet une pseudo-preuve de terminaison linéaire. Il suffit de prendre pour tous les symboles g de la signature, l'interprétation $[g](x) = x + 1$. □

Proposition 3.4.21. Soit un système de réécriture $(\mathcal{R},\Sigma)$ linéaire. Il existe une constante c telle que pour tout terme t, on a : si $t \to t'$, alors $|t'| < c|t|$.

Démonstration. C'est une conséquence triviale de la proposition 3.2.6. □

Corollaire 3.4.22. Les systèmes de réécriture linéaires de $\Gamma(0)$ (resp. $\Theta(0)$) peuvent être simulés dans NLINSPACE (resp. LINSPACE).

Démonstration. La construction proposée au corollaire 3.4.13 s'applique ici. □

Remarque 3.4.23. Nous n'avons pas de caractérisation équivalente pour le cas des interprétations polynomiales usuelles. Ceci est dû au fait que le modèle de calcul des machines de Turing n'est plus adapté pour le « temps linéaire ». Pour les caractérisations que nous avons vues précédemment, nous pouvions faire une réduction d'un modèle de machine à l'autre.

Chapitre 4

Caractérisations pour l'ordre de Knuth-Bendix

Nous entamons ici un court chapitre qui traite des liens entre les preuves de terminaison pour l'ordre de Knuth-Bendix et les classes de complexité. Nos motivations sont les mêmes que pour le chapitre précédent et les moyens utilisés s'en rapprochent.

L'ordre KBO est un ordre syntaxique introduit par Knuth et Bendix [KB70] à la fin des années 1960. Il consiste en un contrôle sur la taille des termes à l'aide d'une fonction de poids et d'un ordre lexicographique pour différencier les termes de même poids.

Cet ordre est utilisé dans des systèmes de prouveurs de théorèmes comme Vampire [RV99] ou bien SPASS [Wei99].

Définition 4.0.24. Nous supposons que la signature Σ est munie d'un poids, c'est-à-dire d'une fonction $w : \Sigma \to \mathbb{N}$. Cette fonction s'étend canoniquement sur les termes $\mathcal{T}(\Sigma)$:

$$w(f(t_1,t_2,\ldots,t_k)) = w(f) + \sum_{i=1}^{k} w(t_i)$$

Le poids d'un terme est une mesure de sa taille qui omet les symboles de poids nul. Nous verrons par la suite que la restriction à des

poids strictement positifs donne lieu à une caractérisation en espace du système de réécriture.

D'autre part, nous supposons que la signature Σ est équipée d'une précédence, i.e. d'un pré-ordre noté $\preceq$. Nous en déduisons la relation d'équivalence sur les symboles, $\simeq$ donnée par : $f \simeq g$ ssi $f \preceq g$ et $g \preceq f$. Nous introduisons également l'ordre strict induit $f \prec g$ ssi $f \preceq g$ et $f \not\simeq g$. $\succ$ est son complémentaire strict. La relation $\simeq$ est étendue sur les termes par :

$$\begin{cases} c \simeq c' & \text{pour } c \text{ et } c' \text{ des constantes avec } c \simeq c' \\ f(t_1, \cdots, t_n) \simeq g(s_1, \cdots, s_n) & \text{pour } f \simeq g \text{ et } \forall i,\ t_i \simeq s_i \end{cases}$$

Définition 4.0.25. Étant donné une signature Σ munie d'un poids w et d'une précédence $\prec$, l'ordre KBO (relatif à ce poids et à cette précédence) est le plus petit ordre sur $\mathcal{T}(\Sigma)$ qui satisfait :

1. si $w(t) > w(s)$, alors $t >_{kbo} s$,
2. si $w(f(t_1, \ldots, t_n)) = w(g(s_1, \ldots, s_n))$ et
   a) si $f \succ g$, alors $f(t_1, \cdots, t_n) >_{kbo} g(s_1, \cdots, s_m)$,
   b) si $f \simeq g$ et $\exists k \leq n : t_k >_{kbo} s_k$ et $\forall i < k : t_i \simeq s_i$, alors $f(t_1, \cdots, t_n) >_{kbo} g(s_1, \cdots, s_m)$.

Nous notons $\geq_{kbo}$ l'ordre définit par $t \geq_{kbo} s$ ssi $t >_{kbo} s$ ou $t \simeq s$.

Définition 4.0.26. Supposons que la signature Σ soit munie d'un poids et d'une précédence. Un système de réécriture $(\mathcal{R},\Sigma)$ respecte la propriété de KBO si pour toute règle $l \to r$ de $\mathcal{R}$ et toute substitution close σ, nous avons : $l\sigma >_{kbo} r\sigma$. Nous notons $(\Sigma,\mathcal{R},w, \prec)$ un tel système.

Exemple 4.0.27. Néanmoins, ainsi défini, l'ordre KBO sur les termes n'est pas bien fondé, comme le montrent les deux exemples suivants (d'après Hofbauer [Hof92a]) :

- Soit $\Sigma = \{a,\bullet\}$, où a est un symbole unaire et $\bullet$ une constante, $\mathcal{R} = \{\bullet \to a(\bullet)\}$, un poids $w(a) = w(\bullet) = 0$, et, finalement, une

précédence $a \prec \bullet$. Le système de réécriture respecte l'ordre KBO mais il admet une dérivation infinie :

$$\bullet \to a(\bullet) \to a(a(\bullet)) \to \cdots$$

- Soit $\Sigma = \{f, \bullet\}$, où f est un symbole binaire et $\bullet$ une constante. Prenons $\mathcal{R} = \{\bullet \to f(\bullet, \bullet)\}$, $w(f) = w(\bullet) = 0$, et finalement, $f \prec \bullet$. Le système de réécriture respecte l'ordre KBO, néanmoins, la dérivation suivante est infinie :

$$\bullet \to f(\bullet, \bullet) \to f(f(\bullet, \bullet), \bullet) \to \cdots$$

Proposition 4.0.28. L'ordre KBO est bien fondé sous les deux hypothèses suivantes :

1. Si $c \in \Sigma$ est une constante, alors, $w(c) \neq 0$,
2. Si $f \in \Sigma$ est un symbole unaire et $w(f) = 0$, alors, $f \succ g$ pour tout $g \in \Sigma$.

En ce cas, l'ordre est dit *gardé*.

Chez de nombreux auteurs, ces deux hypothèses font partie de façon implicite ou explicite de la définition de KBO. Mais pour les cas qui vont nous intéresser par la suite, elles sont inutiles. C'est pourquoi, nous faisons la distinction entre système gardé et non gardé.

Démonstration. Sous les deux hypothèses, l'ordre KBO est un ordre de simplification. Prenons le cas d'un terme n-aire, avec $n > 2$. Nous avons $w(f(t_1, \ldots, t_n)) = w(f) + \sum_{i=1}^{n} w(t_i)$. Comme le poids des constantes est positif, cela assure que $w(t_i) > 0$ pour tout $i \leq n$. Par conséquent, $f(t_1, \ldots, t_n) >_{kbo} t_k$ pour tout $k \leq n$.

Pour les symboles unaires, de deux choses l'une, soit $w(f) > 0$, auquel cas, $w(f(t)) > w(t)$ qui assure $f(t) >_{kbo} t$, soit $w(f) = 0$, auquel cas $w(f(t)) = w(t)$, mais comme f est le plus grand élément pour la précédence, il est en particulier plus grand que la tête de t. De ce fait, $f(t) >_{kbo} t$. □

Propriété 4.0.29. Étant donné un système de réécriture $(\Sigma,\mathcal{R})$, le problème de trouver un poids et une précédence à la signature telle que le système respecte l'ordre KBO est décidable. Des algorithmes ont été suggérés par Lankford [Lan79] et Martin [Mar87] et implantés par Altendorf [Alt89]. Cette propriété de KBO est évidemment cruciale dès lors que l'on veut, par exemple, automatiser les preuves de terminaison d'une spécification.

Proposition 4.0.30. La longueur de dérivation de tout système de réécriture qui respecte l'ordre KBO gardé est au plus multiplement récursif. D. Hofbauer a donné un premier exemple dont la longueur est multiplement récursive (de rang 2 dans la hiérarchie de Péter, c'est-à-dire de l'ordre de la fonction d'Ackermann) dans [HL89]. Par la suite, il donne dans sa thèse [Hof92a] une borne supérieure multiplement récursive de rang 4 dans la hiérarchie de Péter [Pét67]. Par ailleurs, en s'appuyant sur un argument combinatoire, H. Touzet propose une autre preuve[20] que la borne est au plus multiplement récursive [Tou97].

4.1 KBO et LINSPACE

Nous nous intéressons dans cette section à une restriction de l'ordre KBO qui donne lieu à une caractérisation de l'espace linéaire.

Définition 4.1.1. Nous appelons système de réécriture *à poids strictement positifs* tout système de réécriture $(\mathcal{R},\Sigma,w,\prec)$ tel que tout symbole unaire ou constant a un poids strictement positif.

Lemme 4.1.2. Supposons que $(\mathcal{R},\Sigma,w,\prec)$ soit à poids strictement positifs. Alors, nous pouvons borner la taille de tout terme d'une dérivation en fonction de la taille du premier terme de la dérivation.

Démonstration. Nous montrons par deux inductions successives que :

(i) $|t| \leq 2w(t)$,

(ii) $w(t) \leq M.|t|$, où $M = \max_{f\in\Sigma}(w(f))$.

20. preuve qualifiée d'« élégante » par D. Hofbauer.

Si $t >_{kbo} u$, nous avons en particulier $w(t) \geq w(u)$. La conclusion découle alors de (i) et (ii). □

Proposition 4.1.3 (Hofbauer [HL89]). La longueur de dérivation d'un système de réécriture à poids strictement positifs est au plus exponentielle.

Démonstration. L'argument est le suivant: comme tout symbole a un poids strictement positif, l'ordre est bien fondé d'après la proposition 4.0.28. Supposons maintenant que nous avons une dérivation $t_0 \to t_1 \to \cdots \to t_n$. À l'aide du lemme précédent, nous observons de suite que pour tout $i \leq n$, $|t_i| \leq 2M.|t_0|$. En conséquence de quoi, n est borné par le nombre de termes de taille inférieure à $2M|t_0|$, c'est-à-dire $n \leq |\Sigma|^{2M|t_1|}$ □

En fait, la borne est fine, ainsi que le montre l'exemple suivant:

Exemple 4.1.4. Définissons la signature $\Sigma = \{1',0',0,1,\bullet\}$; tous les symboles sont unaires sauf $\bullet$ qui est une constante.

$$\mathcal{R} = \left\{ \begin{array}{lll} 1(\bullet) \to 0'(\bullet) & 1'(\bullet) \to 1(\bullet) & 0(\bullet) \to 0'(\bullet) \\ 1(0'(x)) \to 0(1'(x)) & 0(0'(x)) \to 0'(0'(x)) & 1'(0'(x)) \to 1(1'(x)) \end{array} \right\}$$

Interprétons les termes de l'algèbre comme une notation binaire pour les entiers. À 0 et 0', nous faisons correspondre le chiffre 0, et à 1 et 1' le chiffre 1. Le système de réécriture s'interprète alors comme l'algorithme suivant sur les entiers:

```
input ([[x]])
begin
while ([[x]] > 0)
    begin
    [[x]] := [[x]] - 1
    end
end.
```

De ce fait, si nous prenons au départ la chaîne $\underbrace{1(1(\cdots 1}_{n \text{ fois } 1}(\bullet)))$, la longueur de dérivation pour cette chaîne est de: $[[\underbrace{1(1(\cdots 1}_{n \text{ fois } 1}(\bullet)))]] = 2^n - 1$.

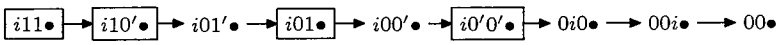


FIG. 4.1 – *Un exemple de dérivation de taille exponentielle*

Corollaire 4.1.5 (Simulation de la réécriture). La forme normale de tout terme d'un système de réécriture à poids strictement positifs peut être calculée dans LINSPACE. Ceci résulte du lemme 4.1.2 (les termes au cours de la dérivation sont de taille bornée linéairement en les entrées) et de la proposition 4.1.3 (la système de réécriture termine). Pour une description plus précise de la machine, nous reportons le lecteur à la description que nous avons faite au chapitre précédent.

4.2 KBO et ESPACE

La deuxième restriction syntaxique que nous étudions a été proposée par Hofbauer [Hof92a], elle concerne l'arité de la signature.

Proposition 4.2.1 (Hofbauer [HL89]). Si un système de réécriture $(\mathcal{R},\Sigma,w,\prec)$, gardé, n'admet que des symboles unaires et des constantes, alors, la longueur de dérivation du système est bornée de façon exponentielle.

Démonstration. La preuve est faite par interprétation des mots par des entiers. En notant $\mathtt{lev}_f = \mathtt{Card}(\{g \in \Sigma \mid g \prec f\})$, si l'on prend le soin de choisir $c > \max_{l\to r\in\mathcal{R}}(|r|) \times |\Sigma|$, l'interprétation d'un terme $B(t) = \sum_{i=1}^{n} \mathtt{lev}_{t[i]} c^{w(t[i..n])}$ décroît avec KBO. □

Cette fois-ci, la borne en taille est exponentielle en la taille des entrées. Néanmoins, nous ne pouvons pas utiliser de tels systèmes pour simuler les machines qui fonctionnent en temps exponentiel. La raison est que seul le symbole spécial peut être produit en un nombre de fois la taille des entrées plus que linéaire. De ce fait, les calculs se font sur un alphabet « presque unaire ». Pour pouvoir considérer des classes de machines qui travaillent sur un alphabet binaire, nous étendons la notion de système de mots de la manière suivante.

Conjecture 4.2.2. La classe des fonctions calculées par les systèmes de réécriture de mots tels que nous venons de les voir est exactement la classe $\text{ETIME} \cap \text{SPACE}(n^2)$.

Piste de démonstration. Étant donné une dérivation $t_0 \to t_1 \to \cdots \to t_n$, on peut écrire le terme $t_n = a_0 s^{i_0} a_1 s^{i_1} \cdots a_k s^{i_k}(c)$ où c est une constante et s le symbole spécial. Comme le système termine en temps exponentiel, chacun des i_j est borné par une exponentielle. D'autre part, comme le poids de t_n est borné par celui de t_0, on peut borner de manière linéaire le nombre de symboles a_i. De ce fait, les calculs sont faits sur une liste de $k|t_0|$ nombres unaires de taille simplement exponentielle. Si on code ces nombres unaires en base 2, on aboutit à des calculs qui se font dans l'espace $|t_0|^2$. □

Définition 4.2.3. Un système de réécriture $(\mathcal{R},\Sigma,w,\prec)$ est dit système de mots enrichi sur KBO si :

1. les symboles sont soit des constantes, soit unaires,
2. les symboles spéciaux sont plus grands pour la précédence que les autres symboles,
3. si l'on identifie les symboles spéciaux, nous avons $\overline{t} \geq_{kbo} \overline{t'}$ où $\overline{t}$ est le terme identifiant les symboles spéciaux de t. Formellement, nous définissons $\overline{\Sigma} = (\Sigma \backslash \{f \mid f \text{ est spécial}\}) \cup \{s\}$. Dans $\overline{\Sigma}$, le poids de s est 0, les autres éléments gardent le poids qu'ils avaient dans Σ. Pour la précédence, $s \succeq f$ pour tout élément $f \in \Sigma$, les autres éléments sont rangés comme dans Σ. Nous définissons maintenant

$$\overline{(-)} : \begin{array}{ll} g \mapsto s & \text{si } g \text{ est spécial} \\ g \mapsto g & \text{dans le cas contraire} \end{array}$$

Proposition 4.2.4. Si $(\mathcal{R},\Sigma,w,\prec)$ est un système de mots sur KBO, alors, non seulement il termine, mais il termine en temps doublement exponentiel. D'autre part, la taille des termes est bornée de façon exponentielle. Il s'ensuit le corollaire :

Corollaire 4.2.5. Le calcul de la forme normale d'un système de mots sur KBO peut être effectué dans ESPACE pour le cas des systèmes confluents et dans NESPACE pour les systèmes non confluents.

Preuve de la proposition 4.2.4. Soit $t_1 \to t_2 \to \cdots \to t_n$ une dérivation dans $(\mathcal{R},\Sigma,w,\prec)$. Nous disons que la transition $t_i \to t_{i+1}$ est *bleue* si $\overline{t_i} \simeq \overline{t_{i+1}}$, *rouge* dans le cas contraire. La proposition 4.2.1 nous permet de conclure qu'il y a au plus $2^{O(n)}$ dérivations rouges (pour $n = |t_1|$).

De plus, nous pouvons observer que la taille des termes ne peut augmenter que pour les transitions rouges. Comme les duplications ne sont pas possibles sur les mots, nous en concluons que la taille des termes est au plus exponentielle : $|t_i| \leq 2^{kn}$ avec $k = \max_{l\to r}(||r|-|l||)$.

Maintenant, combien peut-il y avoir de transitions bleues consécutives? Si la transition est bleue, on peut observer que le poids des termes n'intervient pas dans l'ordre. Par conséquent, pour ces transitions, le système respecte KBO si nous choisissons un poids strictement positif pour tous les symboles. D'après la proposition 4.1.3, la borne est exponentielle en la taille du premier terme de la dérivation ; d'où la borne doublement exponentielle. □

4.3 Simulation de TM à l'aide de systèmes de réécriture

Nous nous consacrons maintenant à la démonstration de la réciproque des deux inclusions montrées dans les sections précédentes. En d'autres termes, nous avons les théorèmes :

Théorème 4.3.0.1.
Les systèmes de réécriture à poids strictement positifs calculent exactement les fonctions de NLINSPACE. *Les systèmes (à poids strictement positifs) confluents calculent exactement les fonctions de* LINSPACE.

Théorème 4.3.0.2.
Les systèmes de mot sur KBO *calculent les fonctions de* NESPACE= ESPACE. *Si l'on se restreint aux systèmes confluents, nous obtenons la classe* ESPACE.

Nous construisons ici un système de réécriture qui simule un certain nombre de transitions d'une machine de Turing. Ce nombre de transitions est un paramètre, ainsi que la taille de l'espace disponible. Nous utiliserons par la suite ce système pour le cas des machines travaillant en espace linéaire et celui des machines travaillant en espace exponentiel. Notons que nous utilisons un modèle avec un ruban unilatère, modèle suffisamment général eu égard au fait que nous ne considérons que des classes de complexité en espace plus grande que LINSPACE ([HU79]).

Soit $\mathcal{M} = (L,Q,q_0,q_f,\delta : L \times Q \to L \times Q \times \{-,0,+\})$, une machine de Turing. Nous construisons le système de réécriture $(\Sigma_{\mathcal{M}},\mathcal{R}_{\mathcal{M}})$ défini de la manière suivante :

$$\Sigma_{\mathcal{M}} = Q \cup L \cup \overline{L} \cup M_Q \cup M_{QL} \cup N_{QL} \cup N_{Q\overline{L}} \cup \{0,1,u,0',\mathrm{E},\bullet\}$$

Les symboles de $\Sigma_{\mathcal{M}}$ sont tous unaires sauf $\bullet$ qui est une constante. Nous utilisons une copie de l'ensemble L : $\overline{L}$. Les lettres de $\overline{L}$ sont surlignées pour les différencier de celles de L. De plus, l'ensemble M_{QL} est l'ensemble des symboles m_{qx} où $q \in Q$ et $x \in L$. Sont définis de manière similaire, M_{QL}, N_{QL} et $N_{Q\overline{L}}$.

Une configuration de la machine est donnée par un terme de la forme :

$$i_1 i_2 \cdots i_k \; q \rhd x_{i_1} x_{i_2} \cdots \overline{x_{i_j}} \cdots x_{i_n} \# \# \cdots \#(\bullet)$$

où pour tout $j \leq k$, $i_j \in \{0,1\}$ et pour tout $j \leq n$, $x_{i_j} \in L$.

La première partie $(i_1 i_2 \cdots i_k)$ est utilisée pour compter les transitions de la machine. On parle par la suite de l'*horloge* du système. L'état de la machine est indiqué par q. Enfin, la dernière partie $(\rhd x_{i_1} x_{i_2} \cdots \overline{x_{i_j}} \cdots x_{i_n} \# \cdots \#)$ représente le ruban de la machine, où le symbole surligné représente le symbole lu par la tête de lecture.

L'idée de la simulation est de faire décroître l'horloge d'une unité pour chaque transition de la machine de Turing (nous identifions la représentation binaire de l'horloge à l'entier correspondant) ; ainsi, le système s'arrête si l'horloge a atteint la valeur nulle ou si le calcul est terminé. La décroissance de l'horloge nous sert par la suite pour disposer d'une preuve de terminaison à l'aide de KBO.

Décrivons maintenant le système de réécriture qui simule la machine (pour la lisibilité, nous avons omis les variables et les parenthèses) :

Gère l'horloge :

$$00' \to 0'0' \qquad 1'0' \to 11' \qquad 0q \to 0'q$$
$$10' \to 01' \qquad\qquad\qquad 1'q \to 1q$$

Lance la transition :

$$1q\rhd \to 0m_q m_{q\rhd}$$

Cherche le symbole courant :

$$m_{qx}y \to xm_{qy}$$

Opère la transition sur le ruban :

$$m_{qx}\overline{y}0' \to \begin{cases} n_{q'\overline{x}}y'0' & \text{pour } \delta(q,y) = (q',y',-) \\ n_{q'x}\overline{y'}0' & \text{pour } \delta(q,y) = (q',y',0) \\ n_{q'x}y'\overline{0'} & \text{pour } \delta(q,y) = (q',y',+) \end{cases}$$

Actualise l'état courant :

$$(chg)\begin{cases} yn_{q'x} \to n_{q'y}x & x \in L + \overline{L} \\ m_q n_{q'x} \to q'x \end{cases}$$

Cas de transition sur la première lettre :

- si $\delta(q,\rhd) = (q', \rhd, 0)$:

$$1q\overline{\rhd} \to 0q'\overline{\rhd}$$

- si $\delta(q,\rhd) = (q', \rhd, +)$:

$$1q\overline{\rhd}x \to 0q' \rhd \overline{x}$$

Pour terminer le calcul, nous introduisons cinq règles supplémentaires :

$$q_f\rhd \to \rhd\mathrm{E} \qquad \mathrm{E}x \to x\mathrm{E} \qquad f\rhd \to \rhd$$
$$q_f\overline{\rhd} \to \rhd \qquad \mathrm{E}\overline{x} \to x$$

où f est à choisir parmi $0,1,0',1'$.

Proposition 4.3.1. Étant donné deux nombres m, e et une configuration de la machine (w,q,h), la représentation de la configuration à l'étape m avec un tampon de taille e est donnée par le terme :

$$t(m,w,q,h,e) = i_1 i_2 \cdots i_k q \rhd x_{i_1} x_{i_2} \cdots x_{i_{j-1}} \overline{x_{i_j}} x_{i_{j+1}} \cdots x_{i_n} \underbrace{\# \cdots \#}_{e \text{ fois}\#}(\bullet)$$

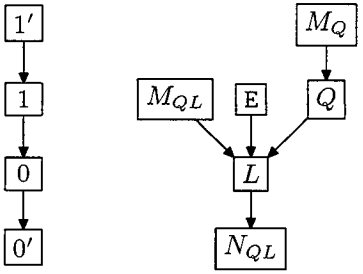
où $i_1 i_2 \cdots i_k$ est la représentation binaire de m et $w = x_{i_1} x_{i_2} \cdots x_{i_n}$.

Nous pouvons observer que si nous avons deux configurations telles que $(w,q,h) \vdash_{\mathcal{M}} (w',q',h')$, alors pour tout $m > 0$, nous avons

$$t(m,w,q,s) \xrightarrow{+}_{\mathcal{R}} t(m-1,w',q',s')$$

où s et s' contrôlent le nombre de cellules utilisées par la machine. Par conséquent, si nous prenons m et s suffisamment grands, il vient $t(m,w,q_0,1,s) \xrightarrow{+}_{\mathcal{R}} w'$ si $w \vdash_{\mathcal{M}} w'$. Le problème est maintenant de construire ces termes initiaux.

Proposition 4.3.2. Supposons que l'ensemble des symboles de $\Sigma_{\mathcal{M}}$ aient le même poids. Alors, le système a une preuve de terminaison pour la précédence suivante :



4.3.1 Simulation de LINSPACE

Supposons maintenant que la machine $\mathcal{M}$ soit dans LINSPACE. Nous rajoutons au système précédent les règles nécessaires pour construire le terme initial. Sans perte de généralité, nous pouvons supposer que la machine travaille en temps borné par 2^{kn} où k est une

constante et qu'elle utilise $A.n$ cellules pour une entrée de taille n. Nous construisons alors le système de réécriture $(\mathcal{R},\Sigma)$ en étendant le système $(\Sigma_{\mathcal{M}},\mathcal{R}_{\mathcal{M}})$:

$$\Sigma = \Sigma_{\mathcal{M}} \cup \mathbf{L} \cup \{f,g,\blacksquare,\square\}$$

Et nous rajoutons les règles :
Lis l'entrée :

$$f \to q_0 g \qquad g\mathbf{x} \to 1^k x \square^A g \qquad g(\bullet) \to \blacksquare(\bullet)$$

Réarrange l'entrée :

$$x\blacksquare \to \blacksquare x \qquad \square 1 \to 1\square \qquad \square\blacksquare \to \blacksquare\square \qquad \square x \to x\square$$
$$x1 \to 1x \qquad 1q_0\blacksquare \to 0q_0\triangleright \qquad q_0 1 \to 1q_0 \qquad \square(\bullet) \to \#(\bullet)$$

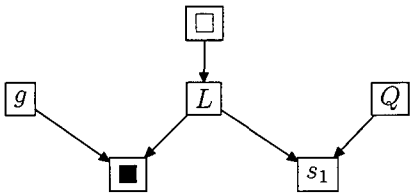
Proposition 4.3.3. Nous pouvons alors observer qu'étant donné un mot $w = x_{i_1}x_{i_2}\cdots x_{i_n} \in L^*$, nous avons :

$$f(\mathbf{x_{i_1}}\mathbf{x_{i_2}}\cdots\mathbf{x_{i_n}}) \xrightarrow{+}_{\mathcal{R}} \underbrace{11\cdots1}_{k|w|} q_0 \overline{\triangleright} w \underbrace{\#\#\cdots\#}_{A|w|}(\bullet)$$

Démonstration. D'après la proposition 4.3.1, et comme l'horloge permet de simuler 2^{kn} transitions, nous sommes assurés que $fw \xrightarrow{+}_{\mathcal{R}} w'$ dès lors que $w \vdash_{\mathcal{M}} w'$. □

Proposition 4.3.4. Le système de réécriture admet une preuve pour l'ordre KBO.

Démonstration. Nous proposons une précédence et un poids qui respectent les hypothèses de la proposition 4.3.2. Pour le poids, nous prenons $w(f) = 3$, $w(\mathbf{x}) = k + A + 2$ et pour tout autre symbole g, nous prenons $w(g) = 1$. La précédence est donnée par le diagramme de Hasse (compatible avec le précédent) :



□

Proposition 4.3.5. De plus, il est aisé de voir que cette construction permet la composition des fonctions calculées par des machines dans LINSPACE. En effet, il suffit de juxtaposer les deux systèmes de réécriture et de multiplier par $2 + k_1 + A_1$ le poids de l'ensemble des symboles du premier système.

4.3.2 Simulation de ESPACE

Nous prenons maintenant le cas où la machine de Turing est une machine qui travaille dans l'espace exponentiel. Nous pouvons supposer sans perte de généralité qu'elle utilise au plus $2^{2^{kn}}$ étapes de calcul (où n est la longueur de l'entrée) et $2^{k.n}$ cases de la machine. En ce cas, nous rajoutons les règles suivantes :

$f \to Fq_0g$	$g\mathbf{x} \to \sigma_1^k xg$	$g(\bullet) \to \blacksquare(\bullet)$
$x\sigma_1 \to \sigma_1 x$	$q_0\ \sigma_1 \to \sigma_1 q_0$	$x\blacksquare \to \blacksquare x$
$\sigma_1\zeta \to \sigma_0\upsilon$	$\sigma_0\zeta \to \zeta\zeta$	$\sigma_0 1 \to \zeta 1$
$\upsilon\zeta \to \sigma_1\upsilon$		$\upsilon 1 \to \sigma_1 1$
$\sigma_1 q_0\blacksquare \to \sigma_0 1 q_0\square$	$\sigma_1 1 \to \sigma_0 1^2\square^k$	
$\square q_0 \to q_0\square$	$\square 1 \to 1\square$	$\square x \to x\square$
$F\sigma_0 \to F$	$F1 \to 1F$	$Fq_0 \to q_0\overline{\triangleright}$
$\square(\bullet) \to \#(\bullet)$		

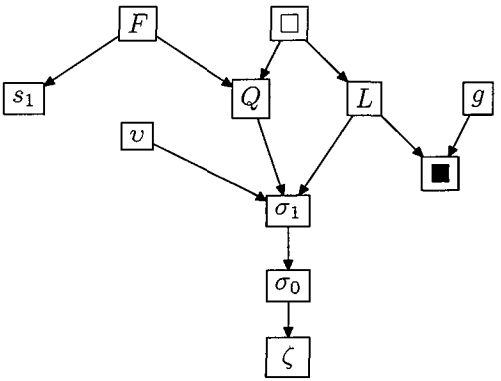
L'idée de la construction est de construire l'horloge et le tampon du ruban à l'aide d'une première horloge *grecque*. Cette horloge fonctionnant en temps exponentiel, nous obtenons alors une horloge (*latine*) et un tampon tous deux de taille exponentielle.

Proposition 4.3.6. Nous pouvons alors observer qu'étant donné un mot $w \in L^*$, nous avons

$$fw \xrightarrow{+}_{\mathcal{R}} \underbrace{11\cdots1}_{2^{k|w|}\text{fois}}\, q_0 \overline{\triangleright} w \underbrace{\#\#\cdots\#}_{2^{k|w|}} (\bullet)$$

Comme précédemment, nous concluons d'après la proposition 4.3.1 que $fw \xrightarrow{+}_{\mathcal{R}} w'$ dès lors que $w \vdash_{\mathcal{M}} w'$.

Proposition 4.3.7. Le système de réécriture termine pour le poids et la précédence suivants. Nous prenons $w(\mathbf{x}) = k+1$, $w(\sigma_0) = w(\sigma_1) = w(\zeta) = w(\upsilon) = w(f) = 1$ et $w(h) = 0$ pour les autres symboles. La précédence est donnée par le diagramme :



Chapitre 5

Sémantique des notations ordinales

Nous nous intéressons plus particulièrement dans ce chapitre à la notion de hiérarchie de fonctions telle que nous l'avons vue à la section 1.3.

En offrant tout un florilège de fonctions sur les entiers, une application naturelle des hiérarchies de fonctions est la construction d'interprétations pour prouver la terminaison de systèmes de réécriture. Plus généralement, les hiérarchies sont employées pour donner des bornes de complexité. Un bel exemple de cette méthode se trouve dans Cichon-Tahhan Bittar [CTB98] qui traite le théorème de Higman sur l'ordre de plongement des mots.

Le point crucial de la construction réside dans l'assignation de suites fondamentales pour les ordinaux limites. Celles-ci permettent de faire des sauts (les *jumps* de Simmons) dans la construction de fonctions dont le taux de croissance est de plus en plus grand. La structure d'ordre sur les ordinaux fournit alors un outil permettant de mesurer la complexité d'une fonction.

Nous sommes pourtant confrontés au problème du choix des suites fondamentales. En effet, la hiérarchie dépend du choix de la suite que l'on fait pour chaque ordinal limite. Dès lors, ce ne sont pas les ordinaux que l'on doit considérer mais les ordinaux équipés de suites

fondamentales d'ordinaux eux-mêmes équipés de suites fondamentales d'ordinaux, etc. Ce concept a été développé initialement par Dennis-Jones et Wainer [DJW83] sous le nom de *terme ordinal*. Dans [CT96], Cichon et Touzet ont poursuivi cette approche. Nous mentionnons également le travail de Simmons [Sim00] qui propose une autre notion d'arbre ordinal (*canonical ordinal notation*).

Ces systèmes s'accordent tous pour introduire une notion d'élément minimal, une notion de successeur et une construction équivalente aux limites ordinales. Nous appelons de telles structures des *progressions* et proposons une nouvelle notion sémantique (les *arborescences*) qui permet d'unifier les démarches décrites précédemment. Nous classifions ces progressions suivant deux critères (inflationnarité et stagnation) et dégageons pour chacuns des cas la sous-structure des arborescences qui est universelle (ou initiale).

Il nous semble toutefois que les points de vue précédents ne prennent pas suffisamment en compte la notion d'ordre comme faisant partie de la construction. Reprenons la définition des termes ordinaux. C'est le plus petit ensemble vérifiant les règles :

$$\frac{}{0 \in \mathcal{O}} \qquad \frac{x \in \mathcal{O}}{sx \in \mathcal{O}} \qquad \frac{f : \mathbb{N} \to \mathcal{O}}{\Delta f \in \mathcal{O}}$$

Après avoir observé que l'on peut inclure Ω (le premier ordinal indénombrable) dans l'ensemble $\mathcal{O}$, Cichon (et par ailleurs Simmons) « colle » un ordre sur $\mathcal{O}$ qui est compatible avec Ω quel que soit le choix de l'inclusion.

Notre approche est différente pour la raison suivante. Rappelons-nous que les termes ordinaux trouvent une de leurs applications dans les hiérarchies de fonctions. L'ordre sur les termes est ainsi défini que pour $\alpha < \beta$, la fonction g_α (resp. h_α, resp. f_α) est une fonction dont le taux de croissance est plus petit que celui de g_β (resp. h_β, resp. f_β) ; en ce sens l'ordre sur les termes ordinaux mesure la complexité des fonctions sous-récursives correspondantes.

Maintenant, si l'on considère la suite fondamentale d'un ordinal limite, on attend de cette suite qu'elle soit croissante, et, de fait, elle « tend » vers l'ordinal limite. Il devrait en être de même pour les hiérarchies de fonctions : la fonction diagonalisée Δf doit être plus complexe

que chacune des fonctions de la famille $(f_n)_{n\in\mathbb{N}}$. Si l'on restreint l'utilisation de la diagonalisation à des familles croissantes de fonctions, nous pouvons affirmer qu'il en est ainsi.

Nous avons choisi une démarche suffisamment générale pour pouvoir prendre en compte plusieurs notions de croissance. Ainsi, dans la suite, nous considérons les deux cas particuliers suivants : celui où les suites ne sont pas monotones (nous retrouvons alors les systèmes cités précédemment) et celui où les suites sont les ω-chaînes ordinaires.

Par ailleurs, notre démarche a été orientée par l'idée (due a Simmons) de construire les hiérarchies de fonctions à l'aide d'un λ-calcul typé. Le calcul de Simmons (qu'il appelle **λH**) est une extension du système T de Gödel. Il contient à la fois un type $\mathcal{N}$ pour les entiers naturels et un type pour les termes ordinaux. En outre, il contient les deux règles d'élimination correspondant à ces types. Elles sont toutes deux fondées sur la propriété d'induction des sémantiques (même si celle des termes ordinaux n'est pas très bien définie) attribuées à ces deux types.

L'un des objectifs principaux que nous aurons sera donc de montrer que notre sémantique bénéficie d'une propriété universelle. Ceci nous donne l'outil théorique nécessaire pour prouver la correction du λ-calcul que l'on construira dans le chapitre 7.

5.1 Les arborescences

Dans cette section, nous définissons le concept fondamental de notre sémantique : les *arborescences*. Informellement, une arborescence est un ordinal équipé d'une suite fondamentale, chaque élément de la suite étant lui-même équipé d'une suite fondamentale, etc. Dans un premier temps, nous définissons une notion de *matrice* qui s'accorde avec la notion de suite de suite de suite etc.

Définition 5.1.1. La *matrice* $\mathbb{D}_{(\mathcal{X},\sqsubseteq)}$ d'un ordre $(\mathcal{X}, \sqsubseteq)$ est l'ensemble des mots sur $\mathcal{X}$. Nous adoptons la convention suivante : nous utilisons les lettres u, v, w, $\cdots$ pour les mots et d'autre part, le mot vide est désigné par la lettre ϵ. Comme par la suite nous ne considérons que des suites dénombrables, nous fixons $\mathcal{X} = \mathbb{N}$; de ce fait, nous abrégerons $\mathbb{D}_{(\mathcal{X},\sqsubseteq)}$ par $\mathbb{D}_{\sqsubseteq}$ (voire par $\mathbb{D}$ si le contexte montre

clairement quel est l'ordre $\sqsubseteq$ en question).

Avant de passer aux notions d'ordres, explicitons un peu notre convention sur les mots. Dans la suite du texte, nous associons aux mots un sens de lecture de gauche à droite ; une lettre à droite d'une autre est « dépendante » de cette dernière. Techniquement, cette dépendance est reflétée par l'ordre $\sqsubseteq^*$ sur les mots. Un autre effet de la dépendance est que nous ne faisons des concaténations que par la gauche.

Pour souligner cette vision, nous ajoutons le mot vide à la fin des mots. Par exemple, si $n \in \mathbb{N}$, le mot $nn\epsilon$ est le mot composé de deux fois la lettre n.

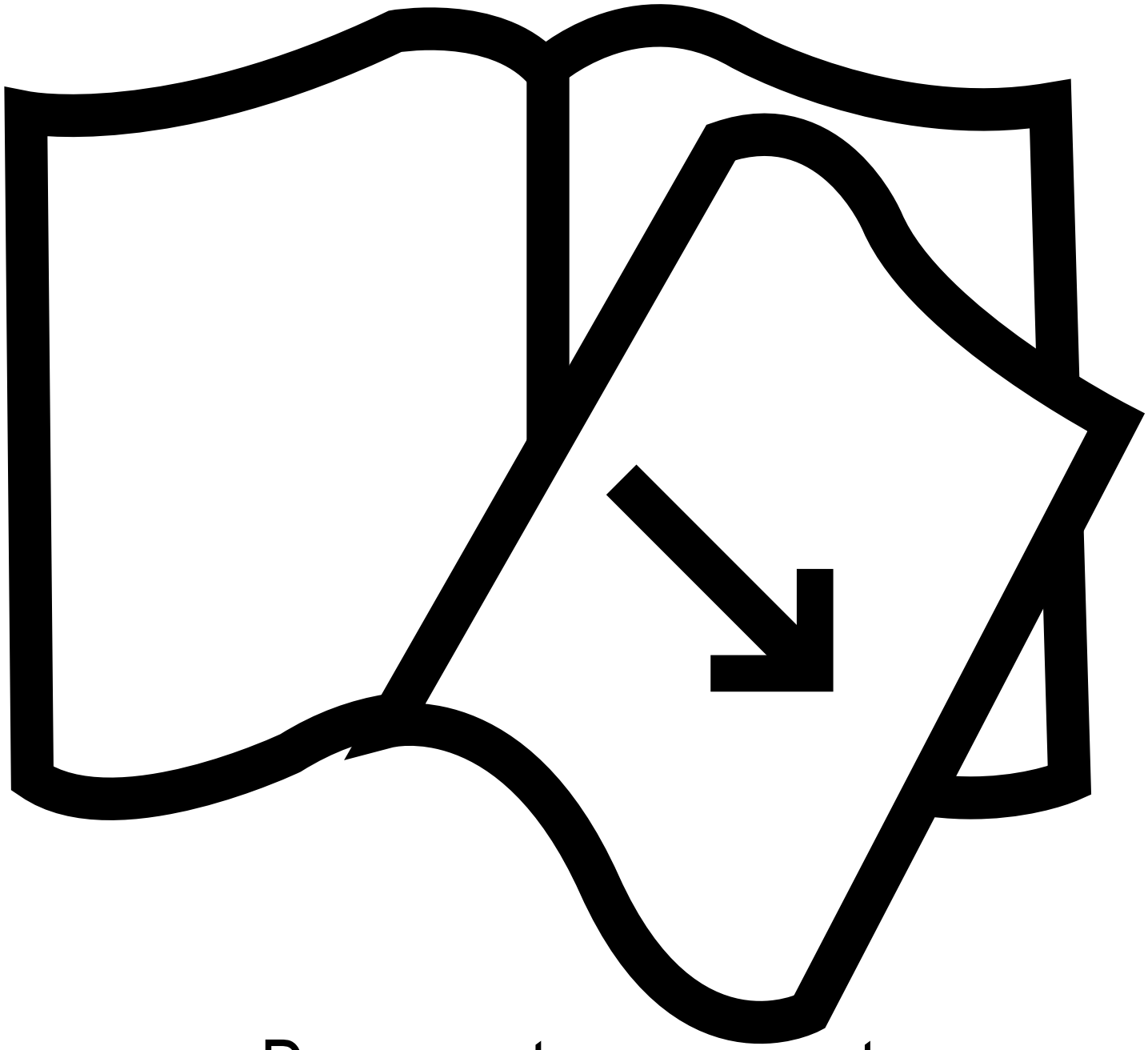
Définition 5.1.2. Nous introduisons maintenant deux ordres sur la matrice $\mathbb{D}$.

- $\trianglelefteq$ est le plus petit ordre qui contient $u.v \trianglelefteq u$ pour tous mots $u, v \in \mathbb{D}$. (On notera que cet ordre est indépendant de $\sqsubseteq$).
- $\sqsubseteq^*$ est le plus petit ordre qui contient à la fois $\trianglelefteq$ et les mots $a_0a_1 \cdots a_k\epsilon \sqsubseteq^* b_0b_1 \cdots b_k\epsilon$ où, pour tout $i \leq k$, a_i et b_i sont des entiers tels que $a_i \sqsubseteq b_i$. On notera que $\trianglelefteq$ est identique à $=^*$ où $=$ est l'ordre discret. En résumé, la notation $\sqsubseteq^*$ est une notation générale dont les deux exemples principaux sont $\trianglelefteq{=}=^*$ et $\leq^*$ où $\leq$ est l'ordre habituel sur les entiers.

Nous notons $\triangleleft$, $<^*$, $\sqsubset^*$ les ordres stricts correspondants. Afin d'aider le lecteur à se faire une intuition des ordres, nous proposons des diagrammes de Hasse (forcément partiels !) des ordres $\trianglelefteq$ et $\leq^*$. Nous n'avons représenté dans ces diagrammes que des mots dont les lettres sont réduites à un chiffre. Ainsi, dans les diagrammes, 12ϵ désigne le mot composé de deux lettres : 1 suivi de 2 (et non le mot composé de la seule lettre 12).

Remarque 5.1.3.

(i) La signification intuitive de $x \trianglelefteq y$ est que x est nécessaire à la construction de y. Plus précisément, la construction de la suite



Documents manquants
(pages, cahiers .....)

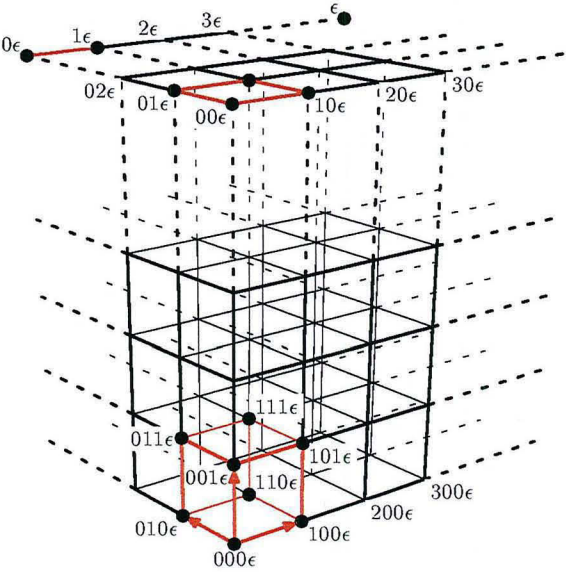


FIG. 5.2 – *Diagramme de Hasse pour* $\leq^*$

Intuitivement, la suite fondamentale de l'ordinal $[a]_w$ est donnée par la suite $([a]_{wi})_{i\in(\mathbb{N},\sqsubseteq)}$. L'ordre $\sqsubseteq$ spécifie la notion de monotonie en ce sens que la suite fondamentale $([a]_{wi})_{i\in(\mathbb{N},\sqsubseteq)}$ doit respecter l'ordre donné par $\sqsubseteq$ sur $\mathbb{N}$. Cet ordre codifie la façon dont une suite fondamentale doit croître. Dans le cas de l'ordre, $=^*$, aucune contrainte n'est donnée, et dans le cs de l'ordre $\leq^*$, on se restreint aux ω-chaînes. La monotonie vis-à-vis de l'ordre $\trianglelefteq$ (qui est un sous-ordre de $\sqsubseteq^*$) correspond au fait que l'ordinal décrit par la suite fondamentale est plus grand que chacun des ordinaux de la famille. La monotonie vis-à-vis des paires $a_0a_1\cdots a_k\epsilon \sqsubseteq^* b_0b_1\cdots b_k\epsilon$ correspond à la monotonie à l'intérieur des suites fondamentales.

Nous nous intéresserons bien sûr plus particulièrement au cas où l'on ne donne aucune contrainte sur la suite ($\sqsubseteq$ réduit à $=$) ainsi qu'au

cas où l'ordre est l'ordre habituel $\leq$ auquel cas les suites sont les ω-chaînes ordinaires.

Exemple 5.1.6. Pour tout $\alpha \in \Omega$, nous définissons l'arborescence $\langle\alpha\rangle$:

$$\langle\alpha\rangle : \begin{cases} [\langle\alpha\rangle]_\epsilon = \alpha \\ [\langle\alpha\rangle]_w = 0 \quad \text{si } w \neq \epsilon \end{cases}$$

Le lecteur vérifiera aisément qu'il s'agit d'une $\sqsubseteq^*$-arborescence pour tout ordre $\sqsubseteq^*$.

Exemple 5.1.7. Un autre exemple important est celui de « l'arborescence constante » (α) :

$$(\alpha) : \forall w \in \mathbb{D}, [(\alpha)]_w = \alpha$$

Remarque 5.1.8.

1. L'arborescence $\langle\alpha\rangle$ est problématique : si α est un ordinal limite non nul, la suite $([\langle\alpha\rangle]_i)_{i\in(\mathbb{N},\sqsubseteq)}$ ne décrit pas correctement l'ordinal α. En effet, nous n'avons pas $\alpha = \lim_{i\in(\mathbb{N},\sqsubseteq)}([\langle\alpha\rangle]_i)$;
2. L'arborescence (α) n'est pas non plus très intéressante : chaque ordinal est décrit par lui-même ; ce processus stérile est sans fin, ce qui pose problème du point de vue de l'induction.

La suite de l'exposé (sections 5.2 et 5.3) formalise ces deux remarques, et montre comment y remédier.

Définition 5.1.9. L'ensemble $\mathbb{A}_{\sqsubseteq^*}$ des $\sqsubseteq^*$-arborescences est muni de la structure d'ordre point à point. Nous notons l'ordre $\longrightarrow$: $a \longrightarrow b$ si pour tout mot w, $[a]_w \leq [b]_w$. Par la suite, nous supprimerons l'indice $\sqsubseteq$ lorsque l'ordre n'intervient pas dans notre propos ou lorsque le contexte montre clairement de quel ordre $\sqsubseteq$ il est question. En particulier, la remarque suivante est indépendante de l'ordre $\sqsubseteq$.

Remarque 5.1.10. $\langle 0\rangle$ est le plus petit élément de $\mathbb{A}$.

5.2 Construction et propriétés élémentaires des arborescences

De la même manière qu'il y a un successeur et une notion de limite pour les ordinaux, il y a un opérateur unaire $\mathbf{s}$ et un opérateur $(\mathbb{N}, \sqsubseteq)$-aire $\mathbf{L}$ sur les arborescences. Par la suite, nous disposerons même d'une propriété universelle relativement à ces deux opérateurs. Cette propriété universelle nous fournira alors l'outil théorique pour construire un λ-calcul qui permettra de calculer avec les arborescences.

5.2.1 Opération successeur, opération limite

Définition 5.2.1. L'opération monotone $\mathbf{s}$ est définie sur les arborescences comme suit :

$$\begin{cases} [\mathbf{s}a]_\epsilon &= [a]_\epsilon + 1 \\ [\mathbf{s}a]_w &= [a]_w \qquad \text{pour } w \neq \epsilon. \end{cases}$$

C'est un exercice simple de montrer que l'arborescence $\mathbf{s}a$ respecte la monotonie de $\mathbb{D}$. Comme c'est le premier du genre, nous donnons une solution. Soit $u \sqsubseteq^* v$. Si $v \neq \epsilon$, nous avons l'inégalité : $[\mathbf{s}a]_u = [a]_u \leq [a]_v = [\mathbf{s}a]_v$. Si $u \neq \epsilon$ et $v = \epsilon$, nous avons les inégalités $[\mathbf{s}a]_u = [a]_u \leq [a]_\epsilon \leq [a]_\epsilon + 1 = [\mathbf{s}a]_\epsilon$. L'inégalité est triviale pour $u = v = \epsilon$.

Par ailleurs, l'opération $\mathbf{s}$ est elle-même monotone sur $\mathbb{A} \longrightarrow \mathbb{A}$. Soit $a \longrightarrow b$. Si $u \neq \epsilon$, nous avons $[\mathbf{s}a]_u = [a]_u \leq [b]_u = [\mathbf{s}b]_u$. Sinon, nous avons $[a]_\epsilon \leq [b]_\epsilon$, d'où : $[\mathbf{s}a]_\epsilon = [a]_\epsilon + 1 \leq [b]_\epsilon + 1 = [\mathbf{s}b]_\epsilon$.

Nous appelons successeur l'opération $\mathbf{s}$ étant donné qu'il/elle joue le même rôle que le successeur des ordinaux. Par extension, toute arborescence $\mathbf{s}a$ est qualifiée de successeur.

Exemple 5.2.2. Pour tout $n \in \omega$, nous avons $\langle n \rangle = \underbrace{\mathbf{s}(\mathbf{s}(\cdots \mathbf{s}}_{n \text{ fois } \mathbf{s}}(\langle 0 \rangle) \cdots))$.
En plus de cela, nous avons pour tout $n \leq m < \omega$ l'inégalité $\langle n \rangle \longrightarrow \langle m \rangle$.

Définition 5.2.3. Soit l'opération $\mathbf{L}$[21] définie sur $\mathbb{A}^{(\mathbb{N},\sqsubseteq)}_{\sqsubseteq^*}$ qui associe

21. $\mathbf{L}$ pour limite.

à toute suite $(a_i)_{i\in(\mathbb{N},\sqsubseteq)}$ l'arborescence $\mathbf{L}((a_i)_{i\in(\mathbb{N},\sqsubseteq)})$ définie comme suit :

$$\begin{array}{lcl}[\mathbf{L}(a_i)_i]_{nw} & = & [a_n]_w \\ [\mathbf{L}(a_i)_i]_\epsilon & = & \lim_{i\in(\mathbb{N},\sqsubseteq)}([a_i]_\epsilon)\end{array}$$

Cette opération respecte la monotonie ; elle est également monotone.

Exemple 5.2.4. L'ordinal ω peut être représenté à l'aide du successeur et de l'opérateur $\mathbf{L}$ dans $\mathbb{A}_{\trianglelefteq}$. Il suffit de choisir :

$$\omega_{\trianglelefteq} \equiv \mathbf{L}(\langle n\rangle)_{n\in\mathbb{N}}$$

Cette représentation peut être transposée dans $\mathbb{A}_{\leq^*}$:

$$\omega_{\leq} \equiv \mathbf{L}(\langle n\rangle)_{n\in\omega}$$

la seule différence étant que l'indice est une fois $\mathbb{N}$ et l'autre fois ω. La représentation pour ω contient beaucoup plus d'informations que celle pour $\mathbb{N}$; en effet, il est nécessaire que pour tout $n \leq m < \omega$, nous ayons $\langle n\rangle \longrightarrow \langle m\rangle$.

Définition 5.2.5. Enfin, il existe une opération « inverse » de la précédente. Pour tout $n \in \mathbb{N}$, la fonction $(-)_n$ opère comme suit :

$$\forall w \in \mathbb{D}\ [(a)_n]_w = [a]_{nw}$$

Il s'agit également d'une opération monotone. L'opération peut être étendue sur les mots $w \in \mathbb{D}$. Nous attirons toutefois l'attention du lecteur sur l'ordre des opérations : $(-)_\epsilon = 1_{\mathbb{A}}$ et $(-)_{iw} = (-)_w \circ (-)_i$. On notera que pour pour toute famille $(a_i)_{i\in(\mathbb{N},\sqsubseteq)}$ et tout $k \in \mathbb{N}$, $(\mathbf{L}(a_i)_i)_k = a_k$.

Remarque 5.2.6. Si l'on considère $\mathbb{A}$ comme une catégorie, nous observons que l'opération

$$\begin{array}{rccl}(-)_{i\in(\mathbb{N},\sqsubseteq)} : & \mathbb{A} & \to & \mathbb{A}^{(\mathbb{N},\sqsubseteq)} \\ & a & \mapsto & (a_0, a_1, \cdots)\end{array}$$

est l'adjointe à droite de $\mathbf{L}$. Nous avons en effet l'équivalence :

$$\mathbf{L}(b_i)_i \longrightarrow a \Longleftrightarrow (\forall i, b_i \longrightarrow a_i)$$

Proposition 5.2.7. Étant donné une arborescence a et deux entiers $n \sqsubseteq m$, nous disposons de l'inégalité suivante : $a_n \longrightarrow a_m$. En effet, pour tout mot w, nous avons en vertu de la remarque 5.1.3, $nw \sqsubseteq^* mw$. Dès lors, $[a_n]_w = [a]_{nw} \leq [a]_{mw} = [a_m]_w$.

5.2.2 Progressions

Définition 5.2.8. Une $\sqsubseteq$-*progression* (en abrégé une progression) est un quintuplet $(S, \leq, \mathbf{z}, \mathbf{s}, \mathbf{l})$ où

- $(S, \leq)$ est un ordre partiel ;
- $\mathbf{z} \in S$;
- $\mathbf{s}$ est une opération monotone sur $(S, \leq)$;
- $\mathbf{l}$ est une opération monotone sur $(S, \leq)^{(\mathbb{N}, \sqsubseteq)}$.

Nous étudierons certains types de progressions. Étant donné l'une d'entre elles $(S, \leq, \mathbf{z}, \mathbf{s}, \mathbf{l})$, nous disons qu'elle est :

- $\mathbf{z}$-stagnante si $\mathbf{l}(\mathbf{z})_i = \mathbf{z}$ où $(\mathbf{z})_i$ est la suite constante égale à $\mathbf{z}$ (en abrégé stagnante) ;
- inflationnaire si pour tout $x \in S$, $x \leq \mathbf{s}(x)$.

Il y a de très nombreux exemples de progressions dans la nature, mais dans notre esprit, il s'agit surtout du concept qui chapeaute les concepts d'ordinaux (et plus particulièrement l'ordinal Ω), de hiérarchies de fonctions, et, mais c'est justement l'objet de l'exposé, d'arborescences.

Les deux critères permettent d'envisager quatre sortes de progressions (progressions simples, inflationnaires, stagnantes et inflationnaires-stagnantes). Pour chacun des cas, nous montrons quelle est la sous-structure des arborescences qui est universelle.

Exemple 5.2.9. $(\Omega, 0, s, \sup)$ est une progression inflationnaire et 0-stagnante.

Exemple 5.2.10. $(\omega^\omega,\lambda x.x+1,\mathbf{s},\Delta)$ est défini comme suit : ω^ω est l'ensemble des fonctions monotones $\mathbb{N} \to \mathbb{N}$ ordonné à l'aide de l'ordre point à point, $\mathbf{s}$ est la fonctionelle $\mathbf{s}(f)(x) = f(x+1)$ et $\Delta(f_i)_i(x) = f_x(x)$ est l'opérateur de diagonalisation classique. Cette progression est la structure sous-jacente à la construction de la hiérarchie de Hardy. Elle est stagnante et inflationnaire.

Nous aurions tout aussi bien pu choisir l'opérateur $\mathbf{s}(f)(x) = \llcorner f(x)/2 \lrcorner$, la progression $(\omega^\omega,0,\mathbf{s},\Delta)$ eût alors été stagnante mais pas inflationnaire. Dans le cas où nous aurions choisi l'opérateur $\Delta'(f_i)_i(x) = f_x(x)+1$ à la place de Δ, la progression n'eût point été stagnante.

Une autre progression de ω^ω qui nous intéresse est celle qui est sous-jacente à la hiérarchie à croissance lente. Elle est décrite comme suit : $\mathbf{z} = \lambda x.0$ (la fonction constante égale à 0) et $\mathbf{s}(f)(x) = f(x)+1$, et enfin, $\mathbf{l} = \Delta$. Cette progression est également inflationnaire et stagnante.

Enfin, il y a la progression de la hiérarchie à croissance rapide. Dans ce cas, l'opération successeur est définie par

$$\mathbf{s}(f)(x) = f^x(x+1)$$

où $f^x = \underbrace{f \circ f \cdots \circ f}_{x \text{ fois } f}$.

La structure est : $(\omega^\omega,\lambda x.x+1,\mathbf{s},\Delta)$.

Exemple 5.2.11. Enfin, le quintuplet $(\mathbb{A}, \longrightarrow, \langle 0 \rangle, \mathbf{s}, \mathbf{L})$ est une progression inflationnaire et stagnante. Notons à ce propos que les seules arborescences qui vérifient l'égalité $x = \mathbf{L}(x)_{i \in (\mathbb{N},\sqsubseteq)}$ sont les arborescences constantes.

Remarque 5.2.12. Il existe une équation plus générale que celle de la z-stagnation, il s'agit de $\mathbf{L}(x)_i = x$ pour toute famille constante $(x)_{i \in (\mathbb{N},\sqsubseteq)}$. Elle est intéressante vu que les opérations sup dans Ω et la diagonalisation Δ de ω^ω vérifient l'équation. Pourtant, elle pose un problème évident du point de vue de l'induction.

Remarque 5.2.13. Quel est le pouvoir d'expression de $\mathbb{A}$, c'est-à-dire, quel ordinal peut être représenté par une arborescence $a \in \mathbb{A}$? L'exemple 5.1.6 montre que c'est Ω. Seulement, cette représentation

ne nous intéresse pas : elle ne peut être décrite à partir des opérations $\langle 0 \rangle$, $\mathbf{s}$, $\mathbf{L}$ que nous nous sommes données. Par la suite, nous chercherons justement à nous restreindre aux arborescences que l'on peut construire à partir de $\langle 0 \rangle$, $\mathbf{s}$ et $\mathbf{L}$ tout en conservant le pouvoir d'expression de $\mathbb{A}$.

5.3 Propriété universelle des arborescences

5.3.1 Fibrations discrètes

Comme nous l'avons vu à la remarque 5.2.13, l'ordre partiel $\mathbb{A}$ contient trop d'éléments. Nous cherchons ici à nous débarrasser des arborescences qui sont « mal construites » et à garder toutes celles qui peuvent être définies à l'aide des opérateurs $\langle 0 \rangle$, $\mathbf{s}$, $\mathbf{L}$.

Définition 5.3.1. Nous utilisons ici la construction de Grothendieck discrète. Dans notre cas, la construction pour une arborescence a est un ordre partiel Σa appelé le *tronc* de l'arborescence, associé à une fonction monotone π :

$$\begin{array}{ccc} \Sigma a & \xrightarrow{\pi} & (\mathbb{D}, \sqsubseteq^*) \\ (w,x) & \longmapsto & w \end{array}$$

Un élément de Σa est une paire (w,x) où $w \in \mathbb{D}$ et $x \in [a]_w$. Nous appelons une telle paire (w,x) un *germe* et nous le notons plus succintement $x_{\downarrow w}$. L'ensemble $[a]_w$ des germes au-dessus du mot w s'appelle la *fibre* au-dessus de w.

Définition 5.3.2. L'ordre sur le tronc, noté $\sqsubseteq^*$ est défini par : $x_{\downarrow v} \sqsubseteq^* y_{\downarrow w}$ si $x = y$ et $v \sqsubseteq^* w$. De manière analogue, $\trianglelefteq$ induit l'ordre $x_{\downarrow v} \trianglelefteq y_{\downarrow w}$ si $x = y$ et $v \trianglelefteq w$. On dit d'un germe que c'est un *point générique*[22] s'il est minimal pour l'ordre $\trianglelefteq$.

22. Le terme générique a été utilisé dans la théorie des domaines stables (cf Berry [Ber78] ou Girard [Gir86]). L'idée est ici la même, bien que les ordres avec lesquels nous travaillons soient différents de ceux de la théorie des domaines.

Proposition 5.3.3. Une fibre d'une arborescence est composée d'un segment initial de points non-génériques surmonté d'un ensemble de points génériques.

Démonstration. Soient a une arborescence et $w \in \mathbb{D}$. Supposons que le germe $x_{\downarrow w}$ soit générique. Supposons alors qu'il existe $y \in [a]_w$ tel que $x < y$. Comme $x_{\downarrow w}$ est générique, pour tout mot $v \neq w$ tel que $v \sqsubseteq^* w$, nous avons $x \notin [a]_v$. Ce qui signifie $[a]_v \leq x < y$. Par conséquent, $y_{\downarrow w}$ est un point générique. □

5.3.2 Décomposition des arborescences

Proposition 5.3.4. L'opérateur s ajoute simplement un point générique au sommet de la fibre au-dessus de ϵ. L'inverse est également vrai : si une arborescence a a un point générique au sommet de la fibre au-dessus de ϵ, il existe une arborescence b telle que $a = \mathrm{s}b$ (dans ce cas, on dit que a est un successeur).

Démonstration. Il suffit de définir $[b]_w = [a]_w$ pour tout $w \neq \epsilon$ et comme $[a]_\epsilon$ admet un plus grand élément, c'est un ordinal de la forme $\alpha + 1$. On définit $[b]_\epsilon = \alpha$. Pour voir que la définition est correcte, il suffit de vérifier que pour tout w, $[b]_w \leq [b]_\epsilon$. Le point $\alpha_{\downarrow\epsilon}$ au sommet de la fibre de Σb est générique. En conséquence de quoi, nous avons pour tout $w \neq \epsilon$, $\alpha \notin [a]_w$. D'où l'on conclut : $[b]_w = [a]_w \leq \alpha = [b]_\epsilon$. □

Proposition 5.3.5. Il y a également une propriété de décomposition si la fibre au-dessus de ϵ ne contient pas de points génériques. En ce cas, $a = \mathbf{L}([a_i]_i)$. Une telle arborescence est appelée *limite.*

Démonstration. Premièrement, nous avons pour tout $i \sqsubseteq j$, $[a]_i \longrightarrow [a]_j$ en raison de la proposition 5.2.7. La famille est donc correctement construite. Deuxièmement, pour tout mot w et tout entier n (c'est-à-dire tout mot $nw \neq \epsilon$), nous avons $[a]_{nw} = [a_n]_w = [\mathbf{L}(a_i)_i]_{nw}$. Il reste le cas de ϵ. Soit $x_{\downarrow\epsilon}$ un point de sa fibre. Comme il n'est pas générique, il existe un mot $iw \lhd \epsilon$ tel que $x \in [a]_{iw}$. De ce fait, $x \in \sup_{i\in\mathbb{N}}([a_i]_\epsilon) = [\mathbf{L}(a_i)_i]_\epsilon$. □

5.3.3 Principe d'induction pour les arborescences

Définition 5.3.6. Étant donné une arborescence, nous disons qu'elle est

- *délimitée* si pour tout suite $\lhd$-décroissante de mots $w_0 \rhd w_1 \rhd \cdots$, il existe n tel que $[a]_{w_n} = \langle 0 \rangle$,
- *localement finie* si les fibres contiennent au plus un nombre fini de points génériques.

Nous montrons par la suite que les arborescences délimitées et localement finies donnent une propriété universelle pour les progressions. Nous qualifions ces arborescences de *plates*.

En particulier, les arborescences des exemples 5.1.6 et 5.1.7 ne sont pas des arborescences plates : les arborescences $\langle \alpha \rangle$ ne sont pas localement finies pour $\alpha \geq \omega$ et les arborescences (α) ne sont pas délimitées (sauf dans le cas particulier de l'arborescence (0)). En revanche, l'arborescence $\omega_{\trianglelefteq}$ est plate (même remarque pour l'arborescence $\omega_{\leq}$, mais dans $\mathbb{A}_{\leq^*}$ au lieu de $\mathbb{A}_{\lhd}$).

Définition 5.3.7. Nous notons $\mathbb{P}$ la restriction de $\mathbb{A}$ aux arborescences plates.

Proposition 5.3.8. Les opérations $\mathbf{s}$, $\mathbf{L}$ et $(-)_n$ peuvent être restreintes à $\mathbb{P}$.

Démonstration. Nous avons vu que $\mathbf{s}$ ajoutait simplement un point générique au-dessus de ϵ. Il respecte donc la propriété de finitude locale, et comme il ne change que la valeur de la fibre au-dessus de ϵ, il respecte également la propriété de délimitation.

En ce qui concerne l'arborescence $\mathbf{L}(a_i)_i$, nous pouvons noter que la fibre au-dessus de ϵ ne contient pas de point générique puisqu'elle est définie à partir d'une limite. Pour les mots iw, nous avons $\mathbf{L}(a_i)_i = [a_i]_w$. Par conséquent, l'arborescence $\mathbf{L}(a_i)_i$ respecte la finitude locale. Pour ce qui est de la délimitation, nous pouvons remarquer que pour toute suite $w_0 \rhd w_1 \rhd \cdots$, il existe un entier k tel que $w_j = kw'_j$ (avec $w'_1 \rhd w'_2 \rhd \cdots$) pour tout $j > 0$. Dès lors, il existe un rang n tel que $[\mathbf{L}(a_i)_i]_{w_n} = [a_k]_{w'_n} = \langle 0 \rangle$.

Enfin, l'opération $(-)_n$ respecte la finitude locale puisque toute fibre de l'arborescence a_n est une fibre de l'arborescence a. De même,

si l'on suppose que l'on dispose d'une suite $w_1 \rhd w_2 \rhd \cdots$, il existe un entier k tel que $[a]_{nw_k} = \langle 0 \rangle$, c'est-à-dire un entier k tel que $[a_n]_{w_k} = \langle 0 \rangle$. □

Remarque 5.3.9. Si l'on suppose que pour tout n l'ensemble $n_\downarrow = \{k \mid k \sqsubseteq n\}$ est fini, alors le pouvoir d'expression de $\mathbb{P}$ est le même que celui de $\mathbb{A}$.

Démonstration. Par induction sur $\alpha \in \Omega$, nous construisons une arborescence $\hat{\alpha}$ telle que $[\hat{\alpha}]_\epsilon = \alpha$ et telle que si $\gamma \leq \alpha$, alors $\hat{\gamma} \longrightarrow \hat{\alpha}$.

- $\hat{0} = \langle 0 \rangle \in \mathbb{P}$. De plus, nous avons trivialement pour tout $\gamma \leq 0$, $\hat{\gamma} \leq \hat{0}$,
- si $\alpha = s\beta$, alors nous définissons $\hat{\alpha} = \mathbf{s}(\hat{\beta})$ où $\hat{\beta}$ provient de l'induction (observons que d'après la proposition 5.3.8, $\mathbf{s}(\hat{\beta}) \in \mathbb{P}$). Nous pouvons alors remarquer que si $\gamma < \alpha$, nous avons $\gamma \leq \beta$; ce qui conduit à $\hat{\gamma} \longrightarrow \hat{\beta} \longrightarrow \hat{\alpha}$.
- si α est un ordinal limite ; comme il est dénombrable, il en existe une énumération $(\alpha_i)_{i \in \mathbb{N}}$. De là, nous construisons la suite monotone (pour l'ordre $\sqsubseteq$) $(\alpha'_n)_{n \in (\mathbb{N}, \sqsubseteq)} = (\sup_{k \sqsubseteq n}(\alpha_k))_{n \in (\mathbb{N}, \sqsubseteq)}$. Comme chaque ensemble $\{k \sqsubseteq n\}$ est fini, nous avons $\alpha'_i < \alpha$. Par induction, nous avons également que pour tout $i \sqsubseteq j$, $\hat{\alpha'_i} \longrightarrow \hat{\alpha'_j}$. De là, nous construisons $\hat{\alpha} = \mathbf{L}(\hat{\alpha'_i})_i$ (qui appartient à $\mathbb{P}$ au vu de la proposition 5.3.8).

□

Théorème 5.3.9.1.
Étant donné une arborescence plate $a \in \mathbb{P}$, il y a trois cas mutuellement exclusifs :

- $a = \langle 0 \rangle$;
- $a = \mathbf{s}(b)$ *avec* $b \in \mathbb{P}$;
- $a \neq \langle 0 \rangle$ *et* $a = \mathbf{L}(a_i)_i$ *avec* $a_i \in \mathbb{P}$ *et pour un certain* i, $a_i \neq \langle 0 \rangle$.

Démonstration. Nous commençons par deux lemmes. Le premier est une conséquence de la propriété de délimitation tandis que le second provient de la finitude locale.

Lemme 5.3.10. Pour tout germe $x_{\downarrow w}$, il existe un point générique $x_{\downarrow v} \sqsubseteq^* x_{\downarrow w}$.

Lemme 5.3.11. Si la fibre $[a]_\epsilon$ contient un point générique, alors, elle contient un point générique au sommet de la fibre.

Preuve du lemme. Comme a est localement finie, il existe un plus grand point générique dans la fibre. Ce point est nécessairement au sommet de la fibre d'après la proposition 5.3.3. □

Nous démontrons maintenant le théorème. Supposons que l'arborescence a ne contienne aucun point générique. Les fibres de a ne contiennent aucuns germes d'après le lemme 5.3.10. En d'autres termes, $a = \langle 0 \rangle$. Si maintenant a contient des points génériques, deux cas se présentent :

1. Si elle possède un point générique au-dessus de ϵ, alors (selon le lemme 5.3.11) il existe un point générique au sommet de la fibre. Nous construisons b comme dans la proposition 5.3.4. Nous observons alors que b est plate.
2. Dans le cas contraire, nous construisons la suite $(a_i)_i$ comme dans la proposition 5.3.5. D'après la proposition 5.3.8, pour tout i, a_i est plate.

□

Le théorème nous permet de transporter le principe d'induction des ordinaux aux arborescences plates. Nous avons besoin pour cela de quelques définitions supplémentaires.

Définition 5.3.12. Étant donné une arborescence a, on appelle *noyau* de l'arborescence l'ensemble

$$\Gamma a = \{w \in \mathbb{D} \mid [a]_w \neq 0\} \cup \{\epsilon\}$$

Définition 5.3.13. Un ensemble de mot W est dit *étalé* si

(i) W n'est pas vide ;

(ii) W ne contient pas de suite infinie décroissante pour $\lhd$;

(iii) W est clos par le haut pour l'ordre $\sqsubseteq^*$.

Nous notons $\mathcal{W}$ l'ensemble des ensembles étalés de mots.

Lemme 5.3.14.

(i) Sur $\mathcal{W}$, nous considérons l'ordre strict suivant : $<$ est le plus petit ordre qui contient les paires $W < W'$ pour lesquelles il existe un entier k avec $kW \subset W'$. Cet ordre est bien fondé d'après la clause (ii) de la définition.

(ii) Le corps d'une arborescence est étalé. Pour toute arborescence a, nous avons $\Gamma a = \Gamma \mathrm{s} a$, et pour toute famille d'arborescences $(a_i)_i$ et tout $n \in \mathbb{N}$, $\Gamma a_n < \Gamma \mathbf{L}(a_i)_i$.

Lemme 5.3.15 (d'induction). Le procédé de décomposition décrit au théorème 5.3.9.1 s'arrête. En d'autres termes, si une propriété P concernant les arborescences vérifie :

(i) $\mathrm{P}(\langle 0 \rangle)$

(ii) $\mathrm{P}(a) \Rightarrow \mathrm{P}(\mathrm{s} a)$

(iii) $(\forall i \in \mathbb{N}, \mathrm{P}(a_i)) \;\&\; (\exists j (a_j \neq \langle 0 \rangle)) \Rightarrow \mathrm{P}(\mathbf{L}(a_i)_{i \in \omega})$.

alors, elle est vraie pour toutes les arborescences de $\mathbb{P}$.

Démonstration. On considère les paires (α, W) où $\alpha \in \Omega$ et W est étalé. Ces paires sont ordonnées lexicographiquement (nous notons $<_{lex}$ cet ordre) et cet ordre est bien fondé. À chaque arborescence a, nous associons la paire $([a]_\epsilon, \Gamma a)$, la fibre au-dessus de ϵ et le noyau de l'arborescence. Par extension, nous disons que $a <_{lex} b$ si $([a]_\epsilon, \Gamma a) <_{lex} ([b]_\epsilon, \Gamma b)$. Observons alors que $\langle 0 \rangle$ est le minimum pour l'ordre $<_{lex}$. L'induction repose sur le fait que $<_{lex}$ est bien fondé.

Le théorème 5.3.9.1 montre qu'il y a trois cas à considérer :

- si $a = \langle 0 \rangle$, nous avons par l'hypothèse (i) que $\mathrm{P}(a)$;
- si $a = \mathrm{s} b$, nous avons $b <_{lex} a$. A l'aide de $\mathrm{P}(b)$ qui vient de l'induction et de l'hypothèse (ii), nous tirons $\mathrm{P}(a)$.
- si $a = \mathbf{L}(a_i)_i$ avec $a_k \neq \langle 0 \rangle$. Alors, pour tout $j \in \mathbb{N}$, nous avons $\Gamma a_j < \Gamma a$. En conséquence de quoi, $a_j <_{lex} a$. D'où $\mathrm{P}(a_j)$ par induction. On en conclut alors $\mathrm{P}(a)$.

□

Remarque 5.3.16. Le lemme précédent nous offre une autre caractérisation des éléments de $\mathbb{P}$. En revanche, il ne nous donne aucune information sur l'ordre de $\mathbb{P}$. Celui-ci est trop riche pour pouvoir être décomposé à l'aide du même principe d'induction que celui dont on dispose pour les éléments.

Nous disposons néanmoins du théorème suivant :

Théorème 5.3.16.1.
La progression $(\mathbb{P}_{\trianglelefteq}, =, \langle 0\rangle, \mathbf{s}, \mathbf{L})$ *est la* $\trianglelefteq$*-progression stagnante universelle.*

Démonstration. Notons tout d'abord que la progression est effectivement stagnante. Maintenant, étant donné une autre progression stagnante $(S, \leq, \mathbf{z}, \mathbf{s}, \mathbf{l})$, nous construisons la plus petite[23] fonction ϕ telle que :

$$\phi(\langle 0\rangle) = \mathbf{z} \tag{5.1}$$
$$\phi(sa) = \mathbf{s}(\phi(a)) \tag{5.2}$$
$$\phi(\mathbf{L}(a_i)_i) = \mathbf{l}(\phi(a_i))_i \tag{5.3}$$

Le lemme d'induction nous permet de montrer que la fonction ϕ est totale. Son unicité provient du fait qu'elle est entièrement spécifiée par les équations 5.1, 5.2, 5.3. En outre, elle est trivialement monotone. □

5.3.4 Arborescences bien fondées

Les arborescences bien fondées que nous introduisons dans cette section sont un sous-ordre des arborescences plates. Pour pouvoir obtenir une propriété universelle des arborescences, nous faisons coller l'ordre au principe d'induction du lemme 5.3.15.

Définition 5.3.17. Nous considérons la restriction de l'ordre $\sqsubseteq^*$ aux mots de la même longueur. $v \sqsubseteq^*_{||} w$ si $v \sqsubseteq^* w$ et $|v| = |w|$.

23. Au sens où son domaine de définition est le plus petit.

Définition 5.3.18. Étant donné une arborescence plate a, nous notons n^a_w le nombre d'éléments génériques au-dessus d'une fibre w ; ces éléments sont situés au-dessus de la fibre d'après la proposition 5.3.3.

Définition 5.3.19. Une arborescence est dite bien fondée si pour tous mots $v \sqsubseteq^*_{||} w$, nous avons $n^a_v \leq n^a_w$. Nous notons $\mathbb{V}$ l'ensemble des arborescences bien fondées ordonné de la manière suivante. $a \xrightarrow{!} b$ si $a \longrightarrow b$ et pour tout mot w, $n^a_w \leq n^b_w$. En d'autres termes, le nombre de points génériques respecte l'ordre point à point.

Proposition 5.3.20. Les opérations $\mathbf{s}$, $\mathbf{L}$ et $(-)_n$ sont stables dans $\mathbb{V}$.

Démonstration. La propriété est immédiate en ce qui concerne les opérations $\mathbf{s}$ et $(-)_n$.

Pour $\mathbf{L}$, considérons une famille $(a_i)_{i\in(\mathbb{N},\sqsubseteq)} \in \mathbb{V}^{(\mathbb{N},\sqsubseteq^*)}$. Prenons deux mots $v \sqsubseteq^*_{||} w$, soit $v = w = \epsilon$, auquel cas $n_v^{\mathbf{L}(a_i)_i} = n_w^{\mathbf{L}(a_i)_i} = 0$. Soit $v = iv'$ et $w = jw'$ avec $v' \sqsubseteq^* w'$ et alors $n_v^{\mathbf{L}(a_i)_i} = n^{a_i}_{v'} \leq n^{a_i}_{w'} \leq n^{a_j}_{w'} = n_w^{\mathbf{L}(a_i)_i}$. □

Proposition 5.3.21. La progression $((\mathbb{V}_\sqsubseteq, \xrightarrow{!}), \langle 0\rangle, \mathbf{s}, \mathbf{L})$ est une $\sqsubseteq$-progression stagnante et inflationnaire.

Remarque 5.3.22. Quel est le pouvoir d'expression de $\mathbb{V}$? C'est le même que celui de $\mathbb{P}$ si l'on fait la même hypothèse sur l'ordre $\sqsubseteq$ qu'à la remarque 5.3.9, c'est-à-dire que les ensembles $\{v \mid v \sqsubseteq^*_{||} w\}$ sont finis.

Démonstration. Nous reprenons la construction $\hat{\alpha}$ (pour tout $\alpha \in \Omega$) de la remarque 5.3.9 et nous transformons ces arborescences en des arborescences $\check{\alpha}$ de $\mathbb{V}$.

Soit une arborescence $\hat{\alpha}$, nous décomposons toute fibre au-dessus de w : $[\hat{\alpha}]_w = \lambda^{\hat{\alpha}}_w + n^{\hat{\alpha}}_w$ en le segment initial des germes non génériques $(\lambda^{\hat{\alpha}}_w)$ surmonté de l'ensemble des points génériques (selon la proposition 5.3.3). En observant la construction de $\hat{\alpha}$, nous pouvons remarquer que pour tout mot w, nous avons l'égalité $\lambda^{\hat{\alpha}}_w = \sup_{i\in(\mathbb{N},\sqsubseteq)}([\hat{\alpha}]_{wi})$.

En notant par ailleurs que pour tout mot w, l'ensemble $\{v \mid v \sqsubseteq^*_{||} w\}$ est fini, nous définissons $[\check{\alpha}]_w = \lambda^{\hat{\alpha}}_w + \sup_{\{v \sqsubseteq^*_{||} w\}}(n^{\hat{\alpha}}_v)$.

On vérifie alors aisément que l'arborescence $\check{\alpha}$ est bien fondée et que $[\hat{\alpha}]_\epsilon = [\check{\alpha}]_\epsilon$. □

Théorème 5.3.22.1.
La progression $(\mathbb{V}_{\sqsubseteq^*}, \langle 0\rangle, \mathbf{s}, \mathbf{L})$ *est la* $\sqsubseteq$*-progression stagnante et inflationnaire universelle.*

Le reste de la section est consacré à la preuve du théorème. Nous commençons par quelques lemmes qui concernent l'ordre sur $\mathbb{V}$.

Lemme 5.3.23. Si $\mathbf{L}(a_i)_i \xrightarrow{!} b$ et $a_j \neq \langle 0\rangle$ pour un certain $j \in \mathbb{N}$,

(i) $b \neq \langle 0\rangle$,

(ii) si $b = \mathbf{s}b'$, alors $\mathbf{L}(a_i)_i \xrightarrow{!} b'$,

Démonstration.

(i) si $a_j \neq \langle 0\rangle$, $[a_j]_\epsilon \neq 0$. Par conséquent, $[b]_\epsilon \geq [\mathbf{L}(a_i)_i]_\epsilon \geq [a_j]_\epsilon > 0$. Donc $b \neq \langle 0\rangle$.

(ii) si $b = \mathbf{s}b'$, nous pouvons remarquer que pour tout mot $w \neq \epsilon$, $[\mathbf{L}(a_i)_i]_w \leq [b]_w = [b']_{iw}$. De plus, $n^{\mathbf{L}(a_i)_i}_w \leq n^b_w \leq n^{b'}_w$. Concernant le mot ϵ, il suffit de noter que $[\mathbf{L}(a_i)_i]_\epsilon$ est un ordinal limite plus petit que l'ordinal successeur $[b]_\epsilon$. Par conséquent, $[\mathbf{L}(a_i)_i]_\epsilon \leq [b']_\epsilon$. D'autre part, la fibre de l'arborescence $\mathbf{L}(a_i)_i$ au-dessus de ϵ ne contient pas de points génériques. D'où $n^{\mathbf{L}(a_i)_i}_\epsilon \leq n^{b'}_\epsilon$.

□

Lemme 5.3.24. Si $a \xrightarrow{!} \mathbf{L}(a_i)_i$, alors a n'est pas un successeur.

Démonstration. $\mathbf{L}(a_i)_i$ n'a pas de points génériques au-dessus de ϵ ; a (qui doit avoir moins de points génériques dans chacune des fibres) n'en a pas. a n'est donc pas un successeur. □

Lemme 5.3.25. Si $\mathbf{s}a' \xrightarrow{!} b$, alors $b = \mathbf{s}b'$. De plus, $a' \xrightarrow{!} b'$.

Démonstration. $\mathbf{s}a'$ a au moins un point générique au-dessus de ϵ. b a donc au moins un point générique au-dessus de ϵ. D'après la proposition 5.3.4, $b = \mathbf{s}b'$. On vérifie alors que $a' \xrightarrow{!} b'$. □

Lemme 5.3.26. Étant donné une progression inflationnaire et stagnante $(S, \leq, \mathrm{z}, \mathrm{s}, \mathrm{l})$, z est le minimum de l'image de la progression libre.

Démonstration. Par induction. □

Nous sommes maintenant en mesure de montrer le théorème 5.3.22.1. Étant donné une progression inflationnaire et stagnante $(S, \leq, \mathrm{z}, \mathrm{s}, \mathrm{l})$, nous construisons la plus petite fonction partielle qui vérifie les équations :

- $F(\langle 0 \rangle) = \mathrm{z}$,
- $F(\mathrm{s}a) = \mathrm{s}F(a)$, si $F(a)$ est défini,
- pour tout $i \in \mathbb{N}$, $F(a_i)$ est défini et pour tout $i \sqsubseteq j$, $F(a_i) \leq F(a_j)$, alors $F(\mathbf{L}(a_i)_{i \in (\mathbb{N}, \sqsubseteq)}) = \mathrm{l}(F(a_i))_{i \in (\mathbb{N}, \sqsubseteq)}$.

La définition est correcte eu égard au théorème 5.3.9.1. De plus, on peut noter que si la fonction précédente est totale, elle est unique (étant déterminée entièrement par les équations).

Nous montrons maintenant que la fonction est totale et qu'elle est monotone. Nous travaillons par induction en utilisant l'ordre $<_{lex}$ que nous avons introduit au lemme 5.3.15. Notre hypothèse d'induction est

(1) Fa est défini,

(2) pour tout $x \xrightarrow{!} a$, Fx est défini,

(3) pour tout $x \xrightarrow{!} y \xrightarrow{!} a$, nous avons $Fx \leq Fy$.

Le théorème 5.3.9.1 montre qu'il suffit de considérer les trois cas suivants :

- $a = \langle 0 \rangle$. En ce cas, nous avons :
  (1) $F(\langle 0 \rangle) = \mathrm{z}$ est défini,
  (2) si $x \xrightarrow{!} a$, alors $x = \langle 0 \rangle$. $F(x)$ est donc défini,
  (3) si $x \xrightarrow{!} y \xrightarrow{!} a$, nous avons $x = y = \langle 0 \rangle$, d'où $Fx \leq Fy$;
- Si $a = \mathrm{s}(a')$.
  (1) comme $a' <_{lex} a$, Fa' est défini. Par conséquent, $F(\mathrm{s}a) = \mathrm{s}F(a)$ est défini,

(2) supposons que $x \xrightarrow{!} a$. Il y a trois cas :
- si $x = \langle 0 \rangle$, $F(x)$ est bien défini,
- si $x = \mathrm{s}x'$, alors d'après le lemme 5.3.25, nous avons $x' \xrightarrow{!} a'$. Comme $a' <_{lex} a$, Fx' est bien défini. Ce qui permet de conclure que Fx est bien défini,
- si $x = \mathbf{L}(x_i)$, le lemme 5.3.23 montre que $x \xrightarrow{!} a'$. Comme dans le cas précédent, nous concluons que Fx est bien défini ;

(3) supposons que $x \xrightarrow{!} y \xrightarrow{!} a$. Nous prouvons $Fx \leq Fy$ en fonction des trois cas concernant x :
- si $x = \langle 0 \rangle$, d'après le lemme 5.3.26, (et comme Fy est bien défini) $Fx \leq Fy$,
- si $x = \mathrm{s}x'$. Par le lemme 5.3.25 nous avons $y = \mathrm{s}y'$ et $x' \xrightarrow{!} y'$. En fait, nous avons $x' \xrightarrow{!} y' \xrightarrow{!} a'$. Par conséquent, $Fx' \leq Fy'$. Ce qui implique immédiatement $Fx \leq Fy$,
- si $x = \mathbf{L}(x_i)$, l'un des x_i étant non nul. Il y a trois cas dépendants de y :
  - y ne peut être $\langle 0 \rangle$ en raison du lemme 5.3.23 (i),
  - supposons $y = \mathrm{s}y'$. En ce cas, le lemme 5.3.23 (ii) montre que $x \xrightarrow{!} y' \xrightarrow{!} a'$. Nous tirons $Fx \leq Fy' \leq Fy$,
  - si $y = \mathbf{L}(y_i)_i$. Le lemme 5.3.24 montre que $x \xrightarrow{!} y \xrightarrow{!} a'$. Par induction, $Fx \leq Fy$;

- Le dernier cas donne $a = \mathbf{L}(a_i)_i$.

(1) Notons tout d'abord que $a_i <_{lex} a$ pour tout $i \in \mathbb{N}$. Par conséquent, pour tout i, Fa_i est bien défini. Ensuite, pour tout $i \sqsubseteq j$, nous avons $a_i \xrightarrow{!} a_j \xrightarrow{!} a_j$. Par induction, $Fa_i \leq Fa_j$. Par conséquent, Fa est bien défini,

(2) supposons que $x \xrightarrow{!} a$. Du fait du lemme 5.3.24, x n'est pas un successeur. Il y a donc deux cas. Soit $x = \langle 0 \rangle$ auquel cas Fx est défini. Soit $x = \mathbf{L}(x_i)_i$. Nous avons pour tout i, $x_i \xrightarrow{!} a_i$, ce qui permet de dire que Fx_i est défini pour

tout $i \in \mathbb{N}$. Supposons maintenant $i \sqsubseteq j$, nous avons $x_i \xrightarrow{!} x_j \xrightarrow{!} a_j$. Dès lors, $Fx_i \leq Fx_j$, ce qui signifie que Fx est bien défini,

(3) supposons que $x \xrightarrow{!} y \xrightarrow{!} a$. D'après le lemme 5.3.24, il y a deux cas à considérer :

- si $x = \langle 0 \rangle$, nous avons $Fx \leq Fy$ (lemme 5.3.26),
- si $y = \mathbf{L}(y_i)_i$. Le lemme 5.3.24 montre que $x_i \xrightarrow{!} y_i \xrightarrow{!} a_i$ pour tout $i \in \mathbb{N}$. Par induction, nous tirons alors $Fx \leq Fy$.

5.3.5 Un deuxième point de vue sur $\mathbb{V}$

L'ordre $\mathbb{V}$ peut être vu de manière différente en s'appuyant de manière plus radicale sur la notion de point générique : on associe à chaque mot w le nombre de points génériques que contient la fibre au-dessus de w. Nous arrivons ainsi à la définition suivante.

Définition 5.3.27. Nous appelons les fonctions monotones $\mathbf{a} : (\mathbb{D}, \sqsubseteq_{||}^*) \to \omega$ des arborescences *stratifiées*[24]. Une arborescence stratifiée est limitée si l'ensemble $\{w \in \mathbb{D} \mid \mathbf{a}(w)\}$ ne contient pas de suite $\lhd$-décroissante infinie.

Définition 5.3.28. L'ensemble des arborescences stratifiées limitées ordonné point à point est noté $\mathcal{V}$. Nous allons voir que cette structure est isomorphe à $\mathbb{V}$.

Exemple 5.3.29. Nous notons 0 l'arborescence stratifiée qui a tout mot w associe 0. C'est l'élément minimum de $\mathcal{V}$.

Définition 5.3.30. L'opération monotone $\mathbf{s}$ agit comme suit :

$$\begin{array}{rccl} \mathbf{s} : & \mathcal{V} \to & \mathcal{V} & \\ & \mathbf{a} \mapsto & \left\{ \begin{array}{lll} \epsilon \mapsto & \mathbf{a}(w) + 1 & \\ w \mapsto & \mathbf{a}(w) & \text{si } w \neq \epsilon \end{array} \right. & \end{array}$$

24. Le concept n'a rien à voir a priori avec les arborescences telles que nous les avons vues jusque là. La suite montrera le contraire, mais n'anticipons pas sur l'exposé.

Définition 5.3.31. De même, nous définissons l'opération $\mathbf{l} : \mathcal{V}^{(\mathbb{N},\sqsubseteq)} \to \mathcal{V}$:

$$(\mathbf{a}_i)_i \mapsto \left\{ \begin{array}{ll} \epsilon \mapsto & \mathbf{a}(w) + 1 \\ iw \mapsto & \mathbf{a}_i(w) \end{array} \right.$$

Définition 5.3.32. Enfin, pour tout entier n, nous définissons l'opération $(-)_n$ par :

$$\mathbf{a} \mapsto \{ \ w \mapsto \ \mathbf{a}(nw) \ \}$$

Théorème 5.3.32.1.
($\mathcal{V}$,0,s,l) *est la progression stagnante inflationnaire initiale.*

Démonstration. Il suffit de montrer qu'il existe un morphisme de progression injectif de $\phi : \mathcal{V} \to \mathbb{V}$. En effet, $\mathbb{V}$ étant une progression universelle, nous sommes assurés de l'existence d'un morphisme $h : \mathbb{V} \to \mathcal{V}$. Comme $\mathbb{V}$ est universel, nous avons l'égalité : $\phi \circ h = 1_{\mathbb{V}}$. De ce fait, nous sommes assurés que le morphisme ϕ est surjectif. Il ne reste plus qu'à montrer qu'il est injectif.

Pour construire le morphismes de progression de $\mathcal{V}$ vers $\mathbb{V}$, nous avons besoin de deux lemmes.

Lemme 5.3.33. Étant donné une arborescence stratifiée $\mathbf{a}$, il y a trois cas mutuellement exclusifs :

(i) $\mathbf{a} = 0$,

(ii) $\mathbf{a}(\epsilon) \neq 0$, auquel cas il existe $\mathbf{b}$ telle que $\mathbf{a} = \mathbf{s}(\mathbf{b})$,

(iii) $\mathbf{a} \neq 0$ et $\mathbf{a}(\epsilon) = 0$ auquel cas $\mathbf{a} = \mathbf{l}(\mathbf{a}_i)_i$, l'un des a_i étant différent de 0.

Lemme 5.3.34. Étant donné une arborescence stratifiée $\mathbf{a}$, nous considérons l'ensemble de mots $\Gamma a = \{\epsilon\} \cup \{w \mid \exists v, \mathbf{a}(v) \neq 0 \text{ et } v \trianglelefteq w\}$. Cet ensemble est étalé. Nous disposons alors sur les arborescences stratifiées d'un ordre analogue à l'ordre $<_{lex}$ (bien fondé) que nous avons vu au lemme 5.3.14 : à chaque arborescence stratifiée $\mathbf{a}$, nous associons la paire $(\mathbf{a}(\epsilon),\Gamma\mathbf{a})$. Ces paires sont ordonnées par $<_{lex}$. Par extension, nous écrivons $\mathbf{a} <_{lex} \mathbf{b}$ si $(\mathbf{a}(\epsilon),\Gamma\mathbf{a}) <_{lex} (\mathbf{b}(\epsilon),\Gamma\mathbf{b})$. Nous pouvons alors noter que $\mathbf{a} <_{lex} \mathbf{sa}$ et $\mathbf{a}_i <_{lex} \mathbf{a}$ si $\mathbf{a} \neq 0$.

Nous construisons maintenant la fonction $\phi : \mathcal{V} \to \mathbb{V}$,

$$\begin{aligned} \phi(0) &= \langle 0 \rangle \\ \phi(\mathbf{s}(\mathbf{b})) &= \mathbf{s}(\phi(\mathbf{b})) \\ \phi(\mathbf{l}(\mathbf{a}_i)_i) &= \mathbf{L}(\phi(\mathbf{a}_i))_i \end{aligned}$$

Nous montrons tout d'abord que la fonction est bien définie. Pour cela, nous faisons une induction en utilisant l'ordre $<_{lex}$. L'hypothèse d'induction est comme précédemment :

(1) $\phi(\mathbf{a})$ est défini,
(2) pour tout $\mathbf{x} \leq \mathbf{a}$, $\phi(\mathbf{x})$ est défini,
(3) pour tout $\mathbf{x} \leq \mathbf{y} \leq \mathbf{a}$ nous avons $\phi(\mathbf{x}) \xrightarrow{!} \phi(\mathbf{y})$.

D'après le lemme 5.3.33, il y a trois cas à étudier :

- $\mathbf{a} = 0$. En ce cas, les points (1), (2) et (3) sont trivialement vérifiés ;
- $\mathbf{a} = \mathbf{s}(\mathbf{b})$;
  1. Comme $\mathbf{b} <_{lex} \mathbf{a}$, $\phi(\mathbf{b})$ est défini ; ce qui implique que $\phi(\mathbf{a}) = \mathbf{s}(\phi(\mathbf{b}))$ est défini,
  2. Si $\mathbf{x} \leq \mathbf{s}(\mathbf{b})$, il y a deux cas. Soit $\mathbf{x}(\epsilon) < \mathbf{s}(\mathbf{b})(\epsilon)$ auquel cas, $\mathbf{x} \leq \mathbf{b}$ et par induction : $\phi(\mathbf{x}) \xrightarrow{!} \phi(\mathbf{b}) \xrightarrow{!} \phi(\mathbf{a})$. Soit $\mathbf{x}(\epsilon) = \mathbf{s}(\mathbf{b})(\epsilon)$, auquel cas, $\mathbf{x} = \mathbf{s}(\mathbf{x}')$. Comme $\mathbf{x}' \leq \mathbf{b}$, nous avons $\phi(\mathbf{x}') \xrightarrow{!} \phi(\mathbf{b})$, d'où l'on tire la conclusion,
  3. En procédant de manière similaire, on montre que si l'on a $\mathbf{x} \leq \mathbf{y} \leq \mathbf{a}$, alors $\phi(\mathbf{x}) \xrightarrow{!} \phi(\mathbf{y})$;
- $\mathbf{a} = \mathbf{l}(\mathbf{a}_i)_i$;
  1. Comme $\mathbf{a}_i <_{lex} \mathbf{a}$, $\phi(\mathbf{a}_i)$ est bien défini. Par ailleurs, comme pour tout $i \sqsubseteq j$, nous avons $\mathbf{a}_i \leq \mathbf{a}_j \leq \mathbf{a}_j$, par conséquent $\phi(\mathbf{a}_i) \xrightarrow{!} \phi(\mathbf{a}_j)$. De ce fait, ϕ est bien définie en $\mathbf{a}$,
  2. Si $\mathbf{x} \leq \mathbf{a}$, on a en particulier, $\mathbf{x} = \mathbf{l}(\mathbf{x}_i)_i$. Comme $\mathbf{x}_i \leq \mathbf{a}_i$, $\phi(\mathbf{x_i})$ est bien défini. Comme par ailleurs, nous avons pour tout $i \sqsubseteq j : \mathbf{x}_i \leq \mathbf{a}_j \leq \mathbf{a}_j$; nous déduisons $\phi(\mathbf{x}_i) \xrightarrow{!} \phi(\mathbf{x}_j)$, ce qui assure le résultat,

3. Une étude similaire à (2) montre que nous avons également (3).

Ensuite, il convient de vérifier que la fonction ϕ est injective. On procède par induction sur les éléments $\phi(x)$ dans $\mathbb{V}$. □

Application aux hiérarchies de fonctions

Les hiérarchies de fonctions sont, comme nous l'avons vu, associées à une progression inflationnaire stagnante, la structure $H = (\omega^\omega, \lambda x.x+1, \mathrm{s}(f)(x) = f(x+1), \Delta)$ pour la hiérarchie de Hardy et la structure $G = (\omega^\omega, \lambda x.0, \mathrm{s}(f)(x) = f(x)+1, \Delta)$ pour la hiérarchie à croissance lente et enfin, la structure $F = (\omega^\omega, \lambda x.0, \mathrm{s}(f)(x) = f^x(x+1), \Delta)$ pour la hiérarchie à croissance rapide. D'après le théorème 5.3.22.1, nous avons donc une fonction monotone :

$$\begin{array}{rccc} h: & \mathbb{V} & \to & \omega^\omega \\ & a & \mapsto & h_a \end{array}$$

qui construit les fonctions sous-récursives de la hiérarchie de Hardy, et une fonction monotone :

$$\begin{array}{rccc} g: & \mathbb{V} & \to & \omega^\omega \\ & a & \mapsto & g_a \end{array}$$

qui construit celles de la hiérarchie à croissance lente, et enfin, une fonction monotone :

$$\begin{array}{rccc} f: & \mathbb{V} & \to & \omega^\omega \\ & a & \mapsto & f_a \end{array}$$

qui définit la hiérarchie à croissance rapide.

Le théorème montre que si l'on a deux arborescences $a \xrightarrow{!} b$, nous avons pour tout $n \in \mathbb{N}$,

$$\begin{array}{rcl} h_a(n) & \leq & h_b(n) \\ g_a(n) & \leq & g_b(n) \\ f_a(n) & \leq & f_b(n) \end{array}$$

5.4 Progressions non nécessairement stagnantes

Nous envisageons maintenant les progressions inflationnaires *générales*, c'est-à-dire les progressions inflationnaires mais pas nécessairement stagnantes. Pour justifier cette approche, nous rappelons que nous nous servons de ces structures universelles comme principe d'induction pour un λ-calcul. Or une équation comme celle de la stagnation est incompatible avec l'induction.

Nous établissons dans cette section la notion d'arborescences croissantes qui est la progression inflationnaire générale. Pour cela, nous faisons l'hypothèse que l'ordre $\sqsubseteq$ est connexe. Plus précisément, la relation de l'ordre est un graphe connexe. Ceci nous empêche en particulier de considérer le cas des $\trianglelefteq$-arborescences. En revanche, l'ordre habituel sur les entiers est connexe.

L'idée de la construction est la suivante : nous avons remarqué que $\langle 0\rangle$ est la seule arborescence pour laquelle $\mathbf{L}(a)_i = a$. Qu'à cela ne tienne, supprimons $\langle 0\rangle$! Il reste alors à éliminer suffisamment d'éléments de $\mathbb{V}$ pour que les opérations $\mathbf{s}$ et $\mathbf{L}$ soient stables. En utilisant un sous-ordre de $\mathbb{V}$, nous conservons la propriété d'inflationnarité, en supprimant $\langle 0\rangle$, on évite l'équation de la stagnation.

Définition 5.4.1. Une arborescence $a \in \mathbb{V}$ est dite *croissante* si :

- $a \neq \langle 0\rangle$;
- si $[a]_{wi} \neq 0$ pour un certain mot de la forme wi, alors pour tout entier $j \sqsubseteq i$, nous avons $[a]_{wj} \neq 0$.

Nous notons $\mathbb{W}$ l'ensemble des arborescences croissantes ; $\mathbb{W}$ est ordonné par l'ordre induit par $\mathbb{V}$.

Exemple 5.4.2. L'arborescence $\langle 1\rangle$ est croissante. C'est même la plus petite arborescence croissante. En effet, toute arborescence croissante a étant différente de $\langle 0\rangle$, $[a]_\epsilon \geq 1$.

Proposition 5.4.3. Les opérateurs $\mathbf{s}$ et $\mathbf{L}$ peuvent être restreints à $\mathbb{W}$.

Démonstration. La proposition est triviale concernant l'opération $\mathbf{s}$.

Pour l'opération $\mathbf{L}$, premièrement, on observe que $\mathbf{L}(a_i)_i \neq \langle 0\rangle$ car pour au moins un i (en fait tous), $a_i \neq \langle 0\rangle$. Pour la deuxième

hypothèse, il y a deux cas :

1. Pour les mots de la forme $n(\epsilon)$, il suffit de remarquer que $[\mathbf{L}(a_i)_i]_{n(\epsilon)} = [a_n]_\epsilon \neq 0$.
2. Pour les mots de la forme nwi, nous pouvons observer que $[\mathbf{L}(a_i)_i]_{nwi} = [a_n]_{wi}$. Par conséquent, si $[\mathbf{L}(a_i)_i]_{nwi} \neq 0$, nous avons $[a_n]_{wj} \neq 0$ pour tout $j \sqsubseteq i$. On en conclut alors $[\mathbf{L}(a_i)_i]_{nwj} \neq 0$.

□

Remarque 5.4.4. En revanche, l'opération $(-)_n$ n'est pas stable sur $\mathbb{W}$. Nous avons en effet $\langle 1\rangle_0 = \langle 0\rangle$. Nous verrons néanmoins que l'on peut l'appliquer sur une arborescence croissante qui n'a pas de points génériques au-dessus de ϵ.

Théorème 5.4.4.1.
Soit a une arborescence croissante. Trois cas mutuellement exclusifs se présentent :

(i) $a = \langle 1\rangle$,
(ii) il existe une arborescence croissante b telle que $a = \mathbf{s}(b)$,
(iii) $a = \mathbf{L}(a_i)_i$.

Démonstration. Que les cas soient mutuellement exclusifs est trivial. Maintenant, il nous suffit de montrer que les décompositions sont correctes. Pour cela, nous nous appuyons sur un lemme :

Lemme 5.4.5. Soit une arborescence croissante a qui n'a pas de points génériques au-dessus de ϵ. Nous avons alors l'égalité :

$$a = \mathbf{L}(a_i)_i$$

Démonstration. L'égalité est vraie dans $\mathbb{V}$, il nous suffit donc de montrer que a_i est une arborescence croissante pour tout $i \in \mathbb{N}$. Il y a deux choses à vérifier.

1. A-t-on $a_n \neq \langle 0\rangle$? La réponse à la question dépend directement du fait que l'ordre $\sqsubseteq$ est connexe (et c'est pour cela qu'on l'a

supposé tel). Pour montrer que $a_n \neq \langle 0 \rangle$, nous montrons que $[a_n]_\epsilon \neq 0$.
Nous observons tout d'abord que comme $a \neq \langle 0 \rangle$ et que a n'a pas de points géneriques au-dessus de ϵ, il existe un mot iw tel que $[a]_{iw} \neq 0$. Par monotonie, $[a]_{i(\epsilon)} \neq 0$. Nous introduisons maintenant l'ensemble $D = \{m \in \mathbb{N} \mid [a]_{m(\epsilon)} \neq 0\}$. Notons au préalable que si $n \in D$, alors $a_n \neq \langle 0 \rangle$. Maintenant, nous pouvons observer que D vérifie les équations :

- $i \in D$,
- si $m \in D$, alors pour tout $k \sqsubseteq m$, $[a]_{k(\epsilon)} \neq 0$ en raison de la croissance de l'arborescence a. D'où $k \in D$,
- si $m \in D$, alors pour tout $m \sqsubseteq k$, $[a]_{k(\epsilon)} \neq 0$ car a est monotone vis-à-vis de $\sqsubseteq^*$.

Comme l'ordre est connexe, nous pouvons conclure que $D = \mathbb{N}$, cqfd.

2. Soit un mot wi tel que $[a_n]_{wi} \neq 0$. Nous avons alors $[a]_{nwi} = [a_n]_{wi} \neq 0$. Ce qui conduit immédiatement pour tout $j \sqsubseteq i$ à : $[a_n]_{wj} = [a]_{nwj} \neq 0$.

□

Maintenant, nous constatons que les deux premiers cas de la décomposition correspondent au fait que l'arborescence admet des points génériques au-dessus de ϵ. La décomposition $a = \mathbf{L}(a_i)_i$ découle alors directement du lemme. □

Théorème 5.4.5.1.
La structure $(\mathbb{W}_\sqsubseteq, \xrightarrow{!}, \langle 1 \rangle, \mathbf{s}, \mathbf{L})$ *est la* $\sqsubseteq$*-progression inflationnaire universelle.*

Démonstration. Premièrement, nous devons remarquer que la structure est bien inflationnaire. Cela provient en fait de la nature inflationnaire de la structure $(\mathbb{V}, \langle 0 \rangle, \mathbf{s}, \mathbf{L})$. En revanche, la structure $(\mathbb{W}, \langle 1 \rangle, \mathbf{s}, \mathbf{L})$ n'est pas stagnante contrairement à la précédente. Ceci est dû au fait que $\langle 0 \rangle$ est la seule arborescence a telle que $\mathbf{L}(a)_i = a$.

Supposons maintenant que nous disposons d'une progression inflationnaire $(S, \leq, \mathrm{y}, \mathbf{s}, \mathbf{l})$. Nous construisons la plus petite fonction ϕ (ainsi que nous l'avons fait précédemment) telle que :

(i) $\phi(\langle 1\rangle) = \mathrm{y}$,
(ii) $\phi(\mathbf{s}(b)) = \mathrm{s}(\phi(b))$,
(iii) $\phi(\mathbf{L}(a_i)_i) = \mathrm{l}(\phi(a_i))_i$

Comme nous l'avons déjà fait, nous procédons par induction en utilisant l'ordre $<_{lex}$. Rappelons que nous associons à toute arborescence a, la paire $([a]_\epsilon, \Gamma a)$ et que nous écrivons $a <_{lex} b$ pour $([a]_\epsilon, \Gamma a) <_{lex} ([b]_\epsilon, \Gamma b)$. Nous utilisons à nouveau l'hypothèse d'induction :

(1) $\phi(a)$ est défini,
(2) pour tout $x \xrightarrow{!} a$, $\phi(x)$ est défini,
(3) pour tout $x \xrightarrow{!} y \xrightarrow{!} a$, nous avons $\phi(x) \leq \phi(y)$.

Comme la démonstration est très proche de celle que nous avons faite pour le théorème 5.3.22.1, nous ne développons que les aspects qui en sont différents.

Trois cas se présentent :

- $a = \langle 1\rangle$. $\phi(\langle 1\rangle)$ est bien défini. Si $x \longrightarrow \langle 1\rangle$, $x = \langle 1\rangle$; par conséquent, $\phi(x)$ est défini. Si $x \longrightarrow y \longrightarrow \langle 1\rangle$, $x = y = \langle 1\rangle$. D'où, $\phi(x) \leq \phi(y)$.
- $a = \mathbf{s}(b)$. Ce cas est essentiellement analogue à celui du théorème 5.3.22.1.
- $a = \mathbf{L}(a_i)_i$. Nous procédons comme pour le théorème 5.3.22.1. Néanmoins, nous indiquons que la preuve est encore réduite du fait du lemme :

  Lemme 5.4.6. Dans $\mathbb{W}$, si $b \longrightarrow \mathbf{L}(a_i)_i$, nous avons $b = \mathbf{L}(b_i)_i$ et pour tout i, $b_i \longrightarrow a_i$.

On vérifie alors que la fonction ϕ est monotone. On dispose en fait du résultat en vertu du point (3) de l'induction précédente. □

5.4.1 Progression générale

La construction que nous avons faite plus haut peut s'appliquer pour obtenir la progression universelle la plus générale (ce qui ne signifie pas qu'elle soit la plus intéressante). Nous reprenons l'ordre $\mathbb{P}_{\trianglelefteq}$ que nous avons montré être la progression stagnante universelle, et nous considérons :

Définition 5.4.7. Une arborescence $a \in \mathbb{P}_{\trianglelefteq}$ est *simple* si :

- $a \neq \langle 0 \rangle$,
- si $[a]_{wi} \neq \langle 0 \rangle$ alors $[a]_{wk} \neq \langle 0 \rangle$.

On note $\mathbb{S}_{\trianglelefteq}$ l'ensemble des arborescences plates.

Théorème 5.4.7.1.
L'ordre discret $(\mathbb{S}_{\trianglelefteq}, =, \langle 1 \rangle, \mathbf{s}, \mathbf{L})$ *est la* $\trianglelefteq$*-progression initiale.*

Démonstration. Pour montrer ce résultat, on procède en panachant la démonstration du théorème 5.3.16.1 et celle du théorème 5.4.5.1. □

5.5 Récapitulation

Nous avons défini quatre sortes de progressions, et pour chacune d'elle un modèle universel.

$\sqsubseteq$-progression universelle	stagnant	non nécesmt stagnant
inflationnaire	$\mathbb{V}$	$\mathbb{W}$ (1)
non nécesmt inflationnaire	$\mathbb{P}_{\trianglelefteq}$ (2)	$\mathbb{S}_{\trianglelefteq}$ (3)

1. Rappelons que la propriété n'est démontrée que pour les cas où $\sqsubseteq$ est connexe ;
2. Nous supposons ici que $\sqsubseteq$ est l'ordre discret $=^*$;
3. Nous faisons la même remarque que pour (2).

Chapitre 6

Construction d'ordres

Au chapitre 5, nous avons donné une sémantique à la notion d' « ordinal équipé d'une suite fondamentale » et nous avons dégagé une notion de structure croissante. Nous nous intéressons maintenant au problème de la syntaxe, c'est-à-dire à un calcul qui permette de construire de manière *effective* de telles arborescences. Pour cela, nous nous sommes orientés vers l'utilisation d'un λ-calcul typé comme le fait Simmons [Sim00], mais avec des types dépendants. Un tel calcul sera présenté au chapitre 7.

Pour commencer, nous présentons sous un angle historique le développement de notre travail. Ceci nous permet de justifier un certain nombre de choix que nous avons faits pour la construction de la syntaxe. D'autre part, ceci nous permet de présenter le travail que nous développons dans ce chapitre.

À l'origine, nous avons donc cherché à définir un λ-calcul typé dans lequel nous avons un type « à part entière » pour l'ordre $\mathbb{W}$, et, c'est le point principal, le principe d'induction de $\mathbb{W}$. En fait, nous avons développé une technique plus générale qui permet de définir des ordres issus d'une classe de théories algébriques, les Ψ-théories que nous décrivons par la suite.

Approche à la manière de Simmons

Revenons pour le moment au problème de la syntaxe pour ω. Nous nous sommes orientés dans un premier temps vers un calcul à la manière de Simmons [Sim00]. En voici les principales caractéristiques. Nous utilisons deux niveaux, le premier pour les λ-termes (typés) et le deuxième pour les jugements d'ordre sur ces λ-termes ; ces jugements d'ordre s'appliquent sur des termes t et t' de type ω (éventuellement dans un contexte Γ), nous écrivons ceci sous la forme $\Gamma \vdash t \leq t'$. Pour représenter l'ordre ω par exemple, on introduit les règles :

- Les termes définissant les éléments de ω :

$$\frac{}{\vdash 0 : \omega} \qquad \frac{}{\vdash \mathrm{s} : \omega \to \omega}$$

- Les jugements d'inégalités définissant l'ordre sur ω :

$$\frac{}{\vdash 0 \leq 0 : \omega} \qquad \frac{\Gamma \vdash x \leq y}{\Gamma \vdash \mathrm{s}x \leq \mathrm{s}y} \qquad \frac{\Gamma \vdash x \leq y}{\Gamma \vdash x \leq \mathrm{s}y}$$

- Les jugements qui définissent $\leq$ comme un ordre :

$$\frac{\Gamma \vdash t = t'}{\Gamma \vdash t \leq t'}{}^{25} \qquad \frac{\Gamma \vdash t \leq t' \qquad \Gamma \vdash t' \leq t''}{\Gamma \vdash t \leq t''}$$

Tel quel, le calcul est inopérant, en effet, nous ne sommes pas en mesure de déduire (par un arbre fini de déductions) des jugements aussi simples que

$$n : \omega \vdash 0 \leq n$$

Certes, au niveau de la meta-théorie, une induction sur $\mathbb{N}$ nous permet de déduire l'inégalité. Mais si ce principe suffit à Simmons qui montre les inégalités sur les termes ordinaux *a posteriori*, nous ne pouvons pas nous en contenter[26]. En effet, comme nous intégrons l'ordre

25. L'égalité ici est la fermeture réflexive transitive de la β-réduction. Cette règle contient le cas $0 \leq 0$ que nous avons néanmoins introduit pour ce qu'il est spécifique de ω (en particulier, il est indépendant de la β-réduction).

26. L'idée qui consiste à rajouter l'inégalité $n : \omega \vdash 0 \leq n$ comme axiome n'est pas concluante. En effet, l'ajout de cette seule inégalité $n : \omega \vdash 0 \leq n$ ne résout rien : on retrouve le problème pour $n : \omega \vdash n \leq n$ par exemple. Une

de manière intrinsèque à la structure de progression, nous devons être en mesure de représenter le raisonnement précédent à l'intérieur de la syntaxe.

Or, le calcul que nous cherchons à construire contient justement le principe d'induction de ω qui nous permet de déduire le jugement en question. Nous décidons donc d'unifier le point de vue : les jugements sont des types, ce qui nous amène à la théorie des types dépendants. Nous avons donc envisagé la suite dans ce cadre.

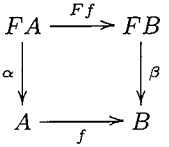
Induction et théorie algébrique

Reprenons maintenant le problème sous l'angle de la sémantique. Le lien entre la notion de principe d'induction et la notion de modèle initial d'algèbre universelle est connu depuis longtemps, au moins depuis Lawvere [Law64] qui donne une axiomatisation des entiers naturels dans ce cadre. De nombreuses structures mathématiques (nous en verrons des exemples) qui donnent lieu à un principe d'induction peuvent être vues comme la structure libre résultant d'un certain choix d'opérations sans équations additionnelles entre elles. Et en fait, de telles équations sont même dangereuses pour le principe d'induction : en identifiant des éléments, on risque de créer des circularités dans l'ordre définissant l'induction. Nous essaierons donc autant que possible d'éviter toute équation entre les opérateurs. De ce point de vue, le choix des types dépendants est judicieux, en ce sens qu'il permet d'intégrer au niveau logique certaines équations.

Revenons sur la notion d'algèbre. Étant donné une catégorie K et un endofoncteur $F : K \to K$, une F-algèbre est la donnée d'une paire (A,α) où A est un objet de K et α un morphisme $FA \xrightarrow{\alpha} A$. Les F-algèbres définissent les objets d'une catégorie pour laquelle un morphisme $f : (A,\alpha) \to (B,\beta)$ est un morphisme de $f : A \to B \in K$ tel que le diagramme commute :

---

meilleure idée consiste donc à définir un ensemble d'axiomes qui permette de déduire « suffisamment » d'inégalités. Outre le fait que le choix des inégalités est *ad hoc*, nous n'avons rien obtenu de probant de cette manière. Il manque toujours une inégalité (plus ou moins triviale par induction).

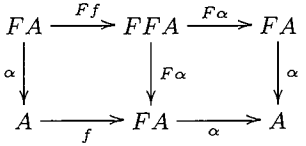


Les identités et la composition étant héritées de celles de K.

Maintenant, que peut-on dire d'un objet initial $FA \xrightarrow{\alpha} A$ dans cette catégorie?

Propriété 6.0.1 (Lemme de Lambek). α est un isomorphisme. Par extension de la notion classique, on qualifie A de point fixe de F.

Démonstration. Comme $FA \xrightarrow{\alpha} A$ est initial, il existe un morphisme f tel que le diagramme (de gauche) commute; comme le diagramme de droite commute également, le rectangle extérieur commute.



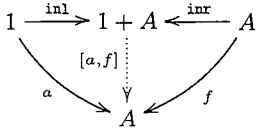
Par unicité, on conclut $\alpha \circ f = 1_A$. On note alors l'égalité $f \circ \alpha = F(\alpha \circ f) = 1_{FA}$. □

Propriété 6.0.2. D'autre part, pour tout autre F-algèbre $FB \xrightarrow{\beta} B$, nous disposons d'un morphisme $f : A \to B$ qui exprime le principe d'induction. A peut être vu comme le plus petit point fixe de F.

Prenons par exemple le cas des entiers naturels. Plaçons-nous dans la catégorie des ensembles, avec $1 = \{\star\}$ l'ensemble terminal et $+$ l'union disjointe sur les ensembles. Les entiers naturels sont obtenus comme étant l'algèbre initiale pour le foncteur :

$$N : X \mapsto 1 + X$$

En d'autres termes, eu égard à la propriété de la somme disjointe (qui est un coproduit), une N-algèbre est la donnée d'un objet A et de deux morphismes $1 \xrightarrow{a} A$ et $A \xrightarrow{f} A$:



On notera alors que les entiers naturels constituent l'algèbre initiale pour le foncteur N, plus précisément, l'algèbre initiale est donnée par $(\mathbb{N},[0,s])$. Pour toute autre N-algèbre $(A,[a,f])$, on dispose d'un morphisme $I_A : \mathbb{N} \to A$ tel que :

$$\begin{cases} I_A(0) = a \\ \forall n, \quad I_A(s(n)) = f(I_A(n)) \end{cases}$$

Ceci peut s'écrire informellement en théorie des types de la manière suivante :

$$\frac{\begin{array}{rcl} & \vdash & a : A \\ x : A & \vdash & f(x) : A \end{array}}{n : \mathbb{N} \quad \vdash \quad I_{Aa,f}n : A}$$

Les deux prémices traduisent le fait que A est une $\mathbb{N}$-algèbre et la conclusion, qui correspond précisément au principe d'induction sur $\mathbb{N}$, traduit le fait que $\mathbb{N}$ est la $\mathbb{N}$-algèbre initiale.

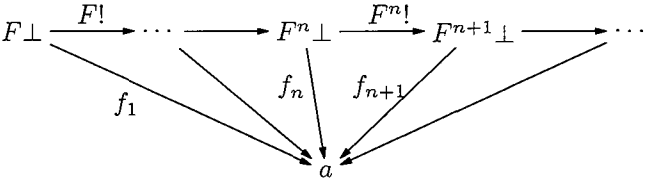
Définition 6.0.3. On dit qu'un foncteur $F : K \to K$ respecte les colimites de ω-chaînes si pour toute colimite $(a,(f_i)_{i\in\omega})$ d'une ω-chaîne $a_0 \xrightarrow{f_{0,1}} a_1 \xrightarrow{f_{1,2}} \cdots$, le diagramme $(Fa,F(f_i)_{i\in\omega})$ est une colimite de la ω-chaîne $Fa_0 \xrightarrow{Ff_{0,1}} Fa_1 \xrightarrow{Ff_{1,2}} \cdots$.

Propriété 6.0.4. Pour calculer l'objet initial dans la catégorie des F-algèbres, une technique usuelle est de calculer la colimite obtenue à partir de l'objet initial de K. Il s'agit de la généralisation en termes catégoriques de la construction d'un plus petit point fixe d'une fonction monotone sur les ensembles ordonnés à partir du plus petit élément : $\mu f = \bigsqcup_{i=0}^{\infty} f^n \bot$. Étant donné une catégorie K qui dispose d'un objet initial $\bot$ et qui admet les colimites de ω-chaînes, étant donné un

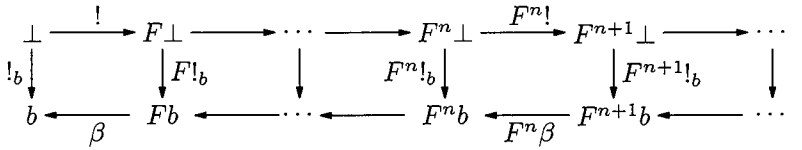
foncteur $F : K \to K$, si F respecte les colimites de ω-chaînes, alors la F-algèbre initiale est la colimite du diagramme :

$$\bot \xrightarrow{!} F\bot \xrightarrow{F!} F^2\bot \to \cdots$$

Démonstration. Observons premièrement que $\bot$ étant initial, il y a un (unique) morphisme $\bot \xrightarrow{!} F\bot$. Les morphismes de la ω-chaîne précédente sont donc plus précisément $f_{i,i+1} = F^i(!)$. On appelle $(a,(f_i)_{i\in\omega})$ la colimite de ce diagramme. Comme F respecte les colimites, $(Fa,(Ff_i)_{i\in\omega})$ est la colimite du diagramme $F\bot \xrightarrow{F!} F^2\bot \to \cdots$. On peut alors observer que a est également un cocône pour ce diagramme :



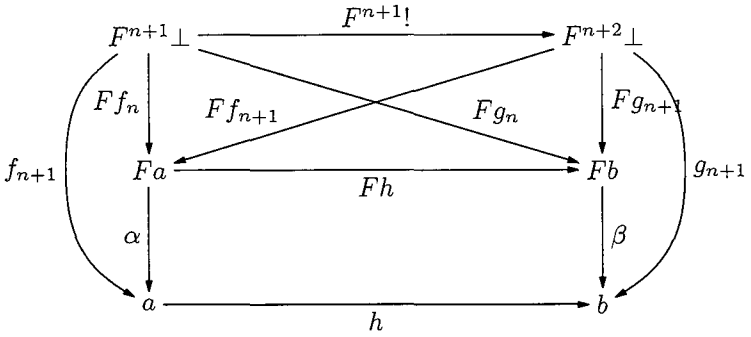
Par conséquent, on a un morphisme $\alpha : Fa \to a$. Supposons maintenant que $Fb \xrightarrow{\beta} b$ est une autre F-algèbre. Comme $\bot$ est initial, le carré le plus à gauche dans le diagramme suivant commute, de ce fait, les autres carrés commutent (par fonctorialité de F) :



Par conséquent, $(b,(g_i)_{i\in\omega})$ est un cocône du diagramme $\bot \xrightarrow{!} F\bot \xrightarrow{F!} F^2\bot \to \cdots$ où $g_0 = !_b$ et $g_{n+1} = \beta \circ Fg_n$. Il existe donc un morphisme $h : A \to B$. Montrons maintenant que h est un morphisme

d'algèbre, c'est-à-dire $\beta \circ Fh = h \circ \alpha$. Observons que l'on a deux épines :

$$(Fa, (Ff_i)_i) \underset{\beta \circ Fh}{\overset{h \circ \alpha}{\rightrightarrows}} (b, (g_{i+1})_i)$$



On a en effet les égalités :

$$\begin{aligned} h \circ \alpha \circ Ff_n &= h \circ f_{n+1} \text{ par définition de } \alpha \\ &= g_{n+1} \end{aligned}$$

et :

$$\begin{aligned} \beta \circ Fh \circ Ff_n &= \beta \circ F(h \circ f_n) \\ &= \beta \circ F(g_n) \\ &= g_{n+1} \end{aligned}$$

Par unicité de l'épine, on a $\beta \circ Fh = h \circ \alpha$. L'unicité de l'épine h se montre alors en utilisant le fait que la colimite est définie à un unique isomorphisme près. □

Exemple 6.0.5. Il y a un cas particulier de foncteurs qui respectent les ω-chaînes, il s'agit des foncteurs polynomiaux. On note X^n le foncteur $X \mapsto \underbrace{(\cdots(X \times X) \times \cdots) \times X}_{n \text{ fois}}$, avec $X^0 = 1$ et $X^1 = X$. Étant donné $n+1$ objets $A_0, \ldots, A_n$ de $\mathcal{C}$, on définit le foncteur polynomial $P : X \mapsto A_0 + A_1 \times X + \cdots + A_n \times X^n$, noté autrement $\sum_{i=0}^{n} A_i X^i$.

Dans les ensembles, étant donné un polynôme $P = \sum_{i=0}^{n} A_i X^i$, A_i étant fini pour chaque i, une P-algèbre $\alpha : PX \to X$ revient à spécifier pour chaque $i \leq n$ et chaque $j \in A_i$ une opération $\alpha_i^j : X^i \to X$. Ceci est dû en particulier au fait que $A_i \times X^i$ s'identifie au foncteur $\underbrace{X^i + \cdots + X^i}_{\mathtt{Card}(A_i) \text{ fois}}$. En d'autres termes, la donnée d'un foncteur polynomial P correspond à une signature $\alpha = (\alpha_i)_{i \leq n}$, et une P-algèbre à une interprétation de cette signature.

En fait, rien ne limite la taille des ensembles A_i, et on peut étendre la définition aux « séries formelles » : $\sum_{i=0}^{\infty} A_i X^i$.

Par rapport à cette situation « idyllique », nous avons plusieurs complications. Premièrement, nous avons un opérateur $\mathbf{L} : \mathbb{W}^\omega \to \mathbb{W}$ dont l'arité n'est pas finie, ensuite, cette même arité n'est pas un ensemble, mais un ordre.

Du point de vue de la sémantique, le passage à une arité infinie ne pose pas de problèmes si on se limite à un petit nombre (au sens d'univers ensembliste) d'opérateurs. De la même manière qu'une opération d'arité n sur X peut être vue comme une fonction $X^{\overline{n}} \to X$ où $\overline{n}$ est l'ensemble à n éléments et $X^{\overline{n}}$ le Hom-interne, on peut considérer n'importe quel ensemble D comme une arité. Une fonction $X^D \to X$ est alors une opération sur X d'arité D. La construction de l'algèbre libre s'obtient dans ce cas en termes de colimite.

Du point de vue de la syntaxe, on peut imiter cette situation en employant un type fonctionnel dans le λ-calcul ; informellement,

$$x : D \to X \vdash \alpha(x) : X$$

et n'importe quel type du calcul peut servir d'arité.

En fait, l'utilisation de Hom-foncteur interne pour définir les arités résout simultanément le problème des arités qui ne sont pas des

ensembles. Dans une catégorie cartésienne fermée, et plus généralement dans une catégorie monoïdale fermée, tout objet A peut être vu comme une arité, et un morphisme $(A \multimap X) \to X$ comme une opération de cette arité.

L'idée d'utiliser les structures monoïdales fermées dans ce contexte date de Lawvere [Law73]. Nous reportons le lecteur à Kelly et Power [Kel82, KP93] qui ont développé ces techniques. Enfin, terminons par mentionner le fait qu'il existe des notions abstraites de taille (pour les catégories localement présentables [GU71, KP93]) qui permettent de considérer une notion équivalente aux arités finies.

Dans ce contexte, de nombreuses études ont été menées. Un des points sur lesquels se sont développés des travaux a été la généralisation de la notion d'endofoncteur. Les foncteurs agissent à la fois sur les objets, mais plus généralement sur des « contextes ». De ce point de vue, la théorie des types de Martin-Löf formalise la notion d'ensemble de manière suffisamment générale pour qu'on puisse définir des notions de signatures très puissantes. Nous mentionnons ici les travaux de [Dyb97, PM93, PS89] et l'article de Dybjer et Setzer qui donne les principes généraux et une bibliographie [DS99]. Mendler donne dans [Men91] les constructions catégoriques correspondantes. Dans cette optique d'algèbre généralisée, l'idée est donc d'utiliser l'expressivité des types dépendants pour définir des signatures suffisamment riches pour définir les opérations dont on a besoin. L'intérêt du formalisme des types dépendants vient du fait que l'on peut éviter un certain nombre d'équations (vis-à-vis d'une logique d'ordre supérieure par exemple), or rappelons-nous, les équations ne font pas bon ménage avec l'induction.

Théorie algébrique et structure d'ordre

Ici, il pourrait paraître séduisant d'employer des T-algèbres libres sur la catégorie des ordres partiels (avec les fonctions monotones) : c'est une catégorie cartésienne fermée (et donc une structure monoïdale fermée). On peut y définir des arités finies (via les ordinaux finis par exemple) ou infinies comme ω.

Le problème est que l'on n'arrive pas à exprimer l'équation d'inflationnarité d'un opérateur en termes d'endo-foncteur. Prenons par

exemple le cas de ω. Essayons de le construire à la manière de ce que nous avons fait pour les entiers naturels. ω peut être obtenu comme l'algèbre initiale dans la catégorie des ordres avec les fonctions monotones pour le foncteur lift que nous avons vu au chapitre 2 :

$$T : X \mapsto \bot + X$$

où $\bot$ est un élément minimum que l'on rajoute à X.
Ceci nous donne bien ω :

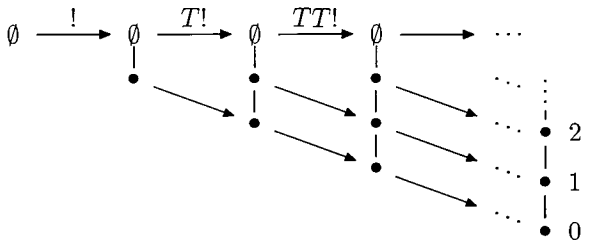


FIG. 6.1 – *Construction de ω*

Mais à quel principe d'induction cela correspond-il? Dans la syntaxe, la règle d'élimination de l'ordre précédent ressemble informellement à :

$$\frac{\begin{array}{rcl}\Gamma & \vdash & a : X \\ \Gamma,x : X & \vdash & fx : X \\ \Gamma,x : X & \vdash & t : a \leq fx\end{array}}{\Gamma,n : \omega \vdash I_{aft}n : X}$$

Cette règle fait appel à un principe plus fort que celui que nous voulons considérer. En effet, nous utilisons ici l'équation $\forall n,\ 0 \leq n$ alors que nous voudrions simplement utiliser l'inflationnarité $\forall n,n \leq sn$. En d'autres termes, nous voudrions une règle du type :

$$\frac{\begin{array}{rcl}\Gamma & \vdash & a : X \\ \Gamma,x : X & \vdash & fx : X \\ \Gamma,x : X & \vdash & t : x \leq fx\end{array}}{\Gamma,n : \omega \quad \vdash \quad I_{aft}n : X}$$

Ce principe d'induction ne se traduit pas facilement en terme d'endofoncteur, nous avons donc cherché à employer un type d'algèbre universelle plus général que celui obtenu au moyen d'endofoncteur. Il semble que ceci soit tout à fait nouveau dans la théorie des principes d'induction. Nous n'allons pas expliciter exactement quel type d'algèbre universelle nous employons, nous nous contentons de décrire certains modèles concrets.

Mais pour cela, nous devons changer d'univers de travail. Il y a au moins deux bonnes raisons à cela. Premièrement, la structure d'ordre est une relation « trop riche » : il est nécessaire de prendre en compte à la fois la réflexivité et (surtout) la transitivité. Ceci complique, voire rend impraticable l'écriture des schémas d'inductions. Il apparaît plus simple de construire des *relations binaires qui se révèlent être des ordres* a posteriori que des ordres directement.

Deuxièmement, même si ceci est moins important, l'utilisation des ordres est incompatible avec les types dépendants, les ordres ne sont pas en effet un modèle naturel des types dépendants. Ceci provient (sans doute) du fait que la catégorie des ordres partiels n'est pas localement fermée (ni même relativement localement cartésienne fermée [Pit89]). Nous nous sommes donc reportés sur la catégorie des relations binaires.

Principe d'inflationnarité pour les relations binaires

Le prix à payer pour utiliser des relations binaires (que nous appelons par la suite des graphes) est que nous devons utiliser un principe d'inflationnarité un peu plus fort que celui que nous avons vu jusque là. Prenons le cas de ω vu comme un simple graphe, ω est l'algèbre libre pour la structure d'algèbre suivante : un triplet $((X,R_X),\zeta,\sigma)$ est une ω_R-algèbre si :

- (X,R_X) est une relation binaire,

- ζ est un élément de X tel que $R_X(\zeta,\zeta)$,
- σ est une opération monotone sujette à une (in)équation d'inflationnarité qui est dans ce cas : $\forall x,y, R_X(x,y) \Rightarrow R_X(x,\sigma y)$.

En d'autres termes, nous avons une règle de la forme :

$$\frac{\begin{array}{rcl}\Gamma & \vdash & a : X \\ \Gamma,x : X & \vdash & fx : X \\ \Gamma,p : R_X(x,y) & \vdash & t(p) : R_X(x,fy)\end{array}}{\Gamma,n : \omega \vdash I_{aft}n : X}$$

Remarque 6.0.6. Nous aurions pu prendre le principe d'inflationnarité des relations binaires pour les ordres, l'inverse n'est pas vrai. Représentons les relations binaires obtenues pour chacun des deux principes (pour simplifier le schéma, nous n'avons pas représenté la réflexivité) :

FIG. 6.2 – *Avec l'hypothèse :* $\forall x, R_X(x,\sigma x)$

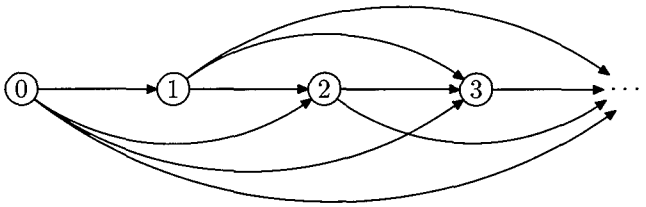


FIG. 6.3 – *Avec l'hypothèse :* $\forall x,y, R_X(x,y) \Rightarrow R_X(x,\sigma y)$

Remarque 6.0.7. Le fait d'utiliser les relations binaires plutôt que les ordres n'apporte rien au problème de la construction de ω en termes d'algèbre universelle au moyen d'endofoncteurs.

Relations binaires, relations binaires réflexives

En fait, quand nous avons abandonné l'idée d'utiliser les ordres pour faire de l'algèbre universelle, nous avons eu un choix, entre les relations binaires et les relations binaires réflexives. Nous commençons par analyser l'utilisation des relations binaires. En théorie des types, une relation binaire est la donnée de :

$$\begin{aligned} \Gamma &\vdash X \text{ type} \\ \Gamma, x, y : X &\vdash R_X(x,y) \text{ type} \end{aligned}$$

La catégorie des relations binaires, ainsi que nous l'avons vu à la section 2, est une catégorie cartésienne fermée, une structure monoïdale fermée par conséquent. Toutefois, cette structure n'est pas adaptée à nos besoins. La raison en est que l'ensemble des éléments de la relation binaire $(X \Rightarrow Y)$ n'est pas égal à $\mathrm{Hom}(X,Y)$, ce qui nous empêche de considérer, au moins naïvement, X comme une arité. Dans la catégorie des relation binaires/graphes, un morphisme $(\omega \Rightarrow X) \to X$ ne peut être vu comme une opération d'arité ω comme nous le souhaitons. En effet, l'ensemble des sommets du graphe $(\omega \Rightarrow X) = \mathbf{Ens}(\mathbb{N}, |X|)$ contient des éléments qui *ne sont pas des ω-chaînes.*

Qu'en est-il des relations binaires réflexives? Leur intérêt vient essentiellement du fait que l'exponentielle vérifie la propriété $\mathrm{Hom}(X,Y) = |(X \Rightarrow Y)|$ qui manque aux relations binaires. Ceci nous permet de considérer X comme une arité, et ainsi de faire de l'algèbre universelle comme nous le ferions naïvement dans les ensembles. Maintenant, le problème est que l'utilisation des relations binaires réflexives demande de gérer des équations (ce que nous cherchons à éviter).

Un morphisme de relation binaire réflexive doit en effet transporter la fonction de réflexivité, ce qui nécessite une équation supplémentaire.

En syntaxe, une relation binaire réflexive est la donnée de :

$$\begin{aligned} \Gamma &\vdash X \text{ type} \\ \Gamma,x,y : X &\vdash R_X \text{ type} \\ \Gamma,x : X &\vdash \mathtt{Ref}_X : R_X(x,x) \end{aligned}$$

Pour tout morphisme $f : X \to Y$, nous devons disposer de $f(\mathtt{Ref}_X(x)) = \mathtt{Ref}_Y(f(x))$. Certes, on pourrait inclure des constructeurs de types spéciaux analogues au « J » de Martin-Löf pour internaliser l'égalité ci-dessus, mais cela va à l'encontre de notre démarche qui cherche à se restreindre aux moyens les plus élémentaires possibles.

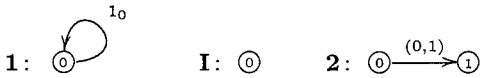
En théorie des types dépendants, la sémantique naturelle des relations binaires est la catégorie des multi-graphes. Un habitant du type $R_X(x,y)$ est une « preuve » de l'existence d'une arête entre x et y.

Mais il reste à résoudre le problème : doit-on utiliser les multi-graphes ou les multi-graphes réflexifs ? Nous venons de voir que les deux possibilités semblent inutilisables en théorie des types. La solution que nous avons retenue est d'employer un moyen terme : une structure monoïdale fermée sur les multi-graphes qui possède certaines qualités désirables des multi-graphes réflexifs.

6.1 Structure monoïdale fermée sur les multi-graphes

La structure monoïdale et les constructions que nous présentons dans cette section concernent la catégorie des multi-graphes. Elles se transposent néanmoins dans la sous-catégorie pleine des graphes. Les constructions sont les mêmes que pour les multi-graphes à ceci près qu'il faut « effacer » le nom des arêtes.

Exemple 6.1.1. Pour commencer, nous (re-)présentons quelques multi-graphes particuliers (qui sont également des graphes comme le lecteur pourra le remarquer) qui interviennent très souvent dans la suite de l'exposé.



Remarque 6.1.2. Nous identifions (classiquement) les sommets du graphe G avec les morphismes $\mathbf{I} \to G$. Pour tout $k \in |G|$, le morphisme $\mathbf{I} \xrightarrow{k} G$ désigne l'unique morphisme qui associe k à $\bullet$.

6.1.1 Une structure monoïdale fermée sur les multi-graphes

Nous avons vu que la catégorie des multi-graphes est cartésienne fermée (cf la section2.7) ; le problème est que l'ensemble des sommets de $X \Rightarrow Y$ ne s'identifie pas avec l'ensemble des morphismes $X \to Y$. Or nous avons besoin de cette propriété pour pouvoir considérer X comme une arité. L'idée de la construction est de rajouter des arêtes de « réflexivité» dans le graphe tenseur. Ces arêtes supplémentaires permettent de travailler jusqu'à un certain point dans la catégorie des multi-graphes comme si on travaillait dans la catégorie des multi-graphes réflexifs.

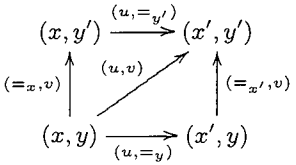
Définition 6.1.3. Étant donné deux multi-graphes X et Y, nous considérons le multi-graphe $X \otimes Y$:

- $|X \otimes Y| = |X| \times |Y|$,
- $R_{X\otimes Y} = R_X \times R_Y + |X| \times R_Y + R_X \times |Y|$. Nous écrivons les flèches de $|X| \times R_Y$ (respectivement celles de $R_X \times |Y|$) comme des paires $(=_x, v)$ où x est un élément de $|X|$ et $v \in R_Y$ (resp. $(u, =_y)$ où $u \in R_X$ et $y \in |Y|$). Nous supposons pour cela que les deux ensembles (de symboles) $\{=_x |\ x \in |X|\}$ et $\{=_y |\ y \in |Y|\}$ sont disjoints de R_X et R_Y.

  Les domaines et les codomaines des flèches sont définis comme suit :

$$\begin{array}{ll} (u,v) : (x,y) - (x',y') & \text{si } u : x - x' \text{ et } v : y - y' \\ (=_x, v) : (x,y) - (x,y') & \text{si } v : y - y' \\ (u, =_y) : (x,y) - (x',y) & \text{si } u : x - x' \end{array}$$

Ceci peut être résumé par un schéma (les arêtes sont représentées par des flèches) :



Proposition 6.1.4. En fait, l'opération $\otimes$ peut être étendue en un foncteur de la manière suivante. À tous morphismes de graphes $f : X \to Y$ et $g : Z \to T$, nous associons le morphisme : $f \otimes g : X \otimes Z \to Y \otimes T$ défini par :

- $(f \otimes g)(x,z) = (f(x),g(z))$,
- - $(f \otimes g)((p,q) : (x,z) - (x',z')) = (f(p),g(q))$,
  - $(f \otimes g)((=_x ,q) : (x,z) - (x,z')) = (=_{f(x)} ,g(q))$,
  - $(f \otimes g)((p, =_z) : (x,z) - (x',z)) = (f(p), =_{g(z)})$.

Proposition 6.1.5. Le tenseur $\otimes$ munit la catégorie **MGr** d'une structure symétrique monoïdale.

Proposition 6.1.6. Nous avons des isomorphismes :

(i) $\lambda_X : \mathbf{I} \otimes X \simeq X$,
(ii) $\rho_X : X \otimes \mathbf{I} \simeq X$,
(iii) $\gamma_{X,Y} : X \otimes Y \simeq Y \otimes X$,
(iv) $\alpha_{X,Y,Z} : X \otimes (Y \otimes Z) \simeq (X \otimes Y) \otimes Z$.

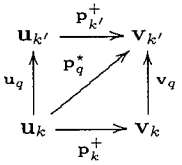
qui vérifient les équations de Mac Lane (cf section 2.5).

Démonstration. Ces isomorphismes sont engendrés par les isomorphismes dus à la structure de produit ordinaire sur les ensembles de sommets et d'arêtes sous-jacents ; on en déduit la commutativité des diagrammes de structure monoïdale. □

Définition 6.1.7. Étant donné deux multi-graphes X et Y, nous définissons le multi-graphe $X \multimap Y$:

- $|X \multimap Y| = \mathbf{MGr}(X,Y)$.

- Une arête entre les sommets $\mathbf{u}$ et $\mathbf{v}$ dans $X \multimap Y$ est une paire $\mathbf{p} = (\mathbf{p}^+,\mathbf{p}^*)$ de fonctions telle que pour tous $k,k' \in X$, $q : k - k' \in X$, nous avons $\mathbf{p}^+_k : \mathbf{u}_k - \mathbf{v}_k \in Y$ et $\mathbf{p}^\star_q : \mathbf{u}_k - \mathbf{v}_{k'}$. En résumé (les arêtes sont représentées par des flèches) :



Nous retrouvons le « diagramme » des graphes réflexifs (à comparer avec le diagrammes des multi-graphes (section 2.7) et celui des multi-graphes réflexifs (la section 2.7.1)).

Nous soulignerons en règle générale en gras les éléments de $X \multimap Y$ (comme des vecteurs). En ce cas, on note les arguments comme des indices.

Remarque 6.1.8. Nous attirons l'attention du lecteur sur le fait que l'ensemble des éléments de $|X \multimap Y|$ est $\mathbf{MGr}(X,Y)$ (contrairement à ce qui se passe pour la structure cartésienne fermée usuelle sur les graphes).

Proposition 6.1.9. La structure $(\mathbf{MGr},\otimes,\multimap)$ est une structure symétrique monoïdale fermée.

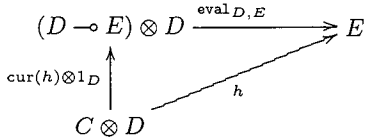
Démonstration. Étant donné un morphisme de graphe $h : (C \otimes D) \to E$, on définit le morphisme de graphe :

$$\begin{array}{rcll} \mathrm{cur}(h) : & C & \to & (D \multimap E) \\ & c & \mapsto & \left\{ \begin{array}{rcl} d & \mapsto & h(c,d) \\ v : d - d' & \mapsto & h(=_c,v) : h(c,d) - h(c,d') \end{array} \right. \\ u : c - c' & \mapsto & \left\{ \begin{array}{rcl} d & \mapsto & h(u,=_d) : h(c,d) - h(c',d) \\ v : d - d' & \mapsto & h(u,v) : h(c,d) - h(c',d') \end{array} \right. \end{array}$$

On définit $\mathtt{eval}_{D,E} : (D \multimap E) \otimes D \to E$ par :

$$\begin{array}{rrcl} \mathtt{eval}_{D,E} : & (D \multimap E) \otimes D & \to & E \\ & (\mathbf{u},d) & \mapsto & \mathbf{u}_d \\ & (\mathbf{p} : \mathbf{u} - \mathbf{v}, q : k - j) & \mapsto & \mathbf{p}^*_q : \mathbf{u}_k - \mathbf{v}_j \\ & (\mathbf{p} : \mathbf{u} - \mathbf{v}, =_k) & \mapsto & \mathbf{p}^+_k : \mathbf{u}_k - \mathbf{v}_k \\ & (=_{\mathbf{u}}, q : k - j) & \mapsto & \mathbf{u}_q \end{array}$$

On peut alors observer que le diagramme suivant commute :



Et en fait, le morphisme $\mathrm{cur}(h)$ est uniquement déterminé par h. Une des raisons à cela est que la currification que nous faisons ici est directement héritée de la currification des ensembles. □

Nous reprenons la construction du bifoncteur Hom-foncteur interne que nous avons vu à la section 2.5 dans le cas des multi-graphes réflexifs. Du point de vue du λ-calcul, il s'agit de l'interprétation de la substitution.

Définition 6.1.10. Étant donné un graphe D, le foncteur $D \multimap (-) : \mathbf{MGr} \to \mathbf{MGr}$ est défini par :

- $X \mapsto (D \multimap X)$,
- étant donné un morphisme de graphe $f : X \to Y$, le morphisme $D \multimap f : (D \multimap X) \to (D \multimap Y)$ est :

$$\mathbf{u} \mapsto f \circ \mathbf{u}$$

Il existe également un foncteur contravariant $(-) \multimap D : \mathbf{MGr}^{\mathrm{op}} \to \mathbf{MGr}$:

- $X \mapsto (X \multimap D)$,
- étant donné $f : X \to Y$, le morphisme $f \multimap D : (Y \multimap D) \to (X \multimap D)$ est :

$$\mathbf{u} \mapsto \mathbf{u} \circ f$$

6.1.2 La structure monoïdale des graphes

Les graphes squelettiques admettent une structure monoïdale similaire à celle que nous venons de voir pour les multi-graphes.

Définition 6.1.11. Le tenseur sur les graphes squelettiques est défini de la manière suivante. Étant donné deux graphes (X,R_X) et (Y,R_Y), le graphe $(X,R_X)\hat{\otimes}(Y,R_Y)$ est composé des sommets $(X\times Y)$, et $(x,y)-(x',y')$ si l'un des trois cas suivants est réalisé :

- $x-x'\in R_X$ et $y-y'\in R_Y$,
- $x=x'$ et $y-y\in R_Y$,
- $y=y'$ et $x-x'\in R_X$.

Note 6.1.1. On remarquera que $(X,R_X)\hat{\otimes}(Y,R_Y)=\widehat{(X,R_X)\otimes(Y,R_Y)}$, la construction $\widehat{(-)}$ étant le collapse du multi-graphe en un graphe de la proposition 2.7.1.

Définition 6.1.12. Le Hom-foncteur $(X,R_X)\hat{\multimap}(Y,R_Y)$ se définit alors :

- $|(X,R_X)\multimap(Y,R_Y)|=\mathbf{Gr}(X,Y)$,
- Étant donné deux morphismes de graphes, $\mathbf{u},\mathbf{v}\in\mathbf{Gr}(X,Y)$, il y a une arête entre les deux si pour tout $x\in X$ on a $\mathbf{u}_x-\mathbf{v}_x$ et pour tous $x-x'$, on a $\mathbf{u}_x-\mathbf{v}_{x'}$.

Note 6.1.2. On remarquera que $(X,R_X)\hat{\multimap}(Y,R_Y)\simeq(X,R_X)\multimap(Y,R_Y)$.

Note 6.1.3. Pour voir que la construction suivante est un Hom-foncteur, on observera que l'on dispose de l'équation

$$\mathrm{Hom}_{\mathbf{Gr}}((X,R_X)\hat{\otimes}(Y,R_Y),(Z,R_Z))\simeq\mathrm{Hom}_{\mathbf{MGr}}((X,R_X)\otimes(Y,R_Y),(Z,R_Z))$$

Ceci est dû au fait que $\hat{(-)}$ est adjoint à gauche de l'inclusion. Ceci nous permet d'écrire :

$$\begin{aligned}\mathrm{Hom}_{\mathbf{Gr}}((X,R_X)\hat{\otimes}(Y,R_Y),(Z,R_Z)) &\simeq \mathrm{Hom}_{\mathbf{MGr}}((X,R_X)\otimes(Y,R_Y),(Z,R_Z))\\ &\simeq \mathrm{Hom}_{\mathbf{MGr}}((X,R_X),(Y,R_Y)\multimap(Z,R_Z))\\ &\simeq \mathrm{Hom}_{\mathbf{Gr}}((X,R_X),(Y,R_Y)\hat{\multimap}(Z,R_Z))\end{aligned}$$

6.1.3 Aspects sémantiques pour une syntaxe

La structure symétrique monoïdale fermée nous fournit la sémantique pour un λ-calcul linéaire (cf section 2.6). En fait, eu égard au morphisme :

$$\begin{array}{rcl} X & \to & X \otimes X \\ x & \mapsto & (x,x) \\ p : x - y & \mapsto & (p,p) : (x,x) - (y,y) \end{array}$$

ce calcul dispose des diagonales, il bénéficie alors de la propriété de contraction (cf Jacobs [Jac94]). En revanche, toujours d'après [Jac94], nous n'avons pas de morphisme $X \otimes Y \to X$ qui nous donne l'affaiblissement. De ce fait, et l'expérience le montre, un tel calcul est *lourd* à utiliser[27] . D'autre part, l'interface avec les types dépendants est mauvaise ; en effet, le lien entre la logique linéaire et les types dépendants demeure mystérieux. Pour remédier à cela, nous restreignons le calcul aux graphes réflexifs mais avec le produit ordinaire de multi-graphes réflexifs.

Proposition 6.1.13. Soient deux multi-graphes réflexifs $(G,\mathtt{Ref}_G)$ et $(H,\mathtt{Ref}_H)$. Il existe une fonction de réflexivité pour le graphe $G \otimes H$. Par la suite, on note le graphe obtenu $(G,\mathtt{Ref}_G) \otimes_{\mathbf{R}} (H,\mathtt{Ref}_H)$.

Démonstration. Nous définissons la fonction de réflexivité :

$$\begin{array}{rcl} |G \otimes H| & \to & R_{G \otimes H} \\ (x,y) & \mapsto & (\mathtt{Ref}_G(x),\mathtt{Ref}_H(y)) : (x,y) - (x,y) \end{array}$$

□

27. Notons qu'il y a un autre exemple de modèle de calcul avec contraction mais sans affaiblissement, il s'agit de la structure monoïdale fermée des ensembles pointés présentée au chapitre 2.

Proposition 6.1.14. Soient deux multi-graphes réflexifs $(G,\mathtt{Ref}_G)$ et $(H,\mathtt{Ref}_H)$. Il existe une fonction de réflexivité pour le graphe $G \multimap H$. Nous notons ce graphe $(G,\mathtt{Ref}_G) \multimap_{\mathbf{R}} (H,\mathtt{Ref}_H)$

Démonstration. Nous définissons alors une fonction de réflexivité :

$$\begin{array}{rcl} |G \multimap H| & \rightarrow & R_{G\multimap H} \\ f & \mapsto & ([x \in |G| \mapsto f\mathtt{Ref}_G(x)],[p : x - y \in G \mapsto fp]) : f - f \end{array}$$

□

Remarque 6.1.15. Notons qu'il y avait une autre fonction de réflexivité possible, il s'agit de :

$$\begin{array}{rcl} |G \multimap H| & \rightarrow & R_{G\multimap H} \\ f & \mapsto & ([x \in |G| \mapsto \mathtt{Ref}_H(fx)],[p : x - y \in G \mapsto fp]) : f - f \end{array}$$

Définition 6.1.16. La catégorie **Refl** que nous utiliserons pour construire un calcul monotone est composée de :

- les objets sont les multi-graphes réflexifs $(X,\mathtt{Ref}_X)$,
- les morphismes sont les morphismes de multi-graphes. Notons que les morphismes *ne respectent pas nécessairement la réflexivité*,

Proposition 6.1.17. La structure $(\mathbf{Refl},\otimes_{\mathbf{R}},\multimap_{\mathbf{R}})$ est monoïdale fermée. Nous avons en effet les équations :

$$\begin{array}{rcl} \mathrm{Hom}_{\mathbf{Refl}}((G,\mathtt{Ref}_G) \otimes_{\mathbf{R}} (H,\mathtt{Ref}_H),(K,\mathtt{Ref}_K)) & \simeq & \mathrm{Hom}_{\mathbf{MGr}}(G \otimes H,K) \\ & \simeq & \mathrm{Hom}_{\mathbf{MGr}}(G,H \multimap K) \\ \mathrm{Hom}_{\mathbf{Refl}}((G,\mathtt{Ref}_G),(H,\mathtt{Ref}_H) \multimap_{\mathbf{R}} (K,\mathtt{Ref}_K)) & \simeq & \end{array}$$

En fait, on peut voir la restriction aux graphes réflexifs comme une structure additionnelle de la catégorie : les objets vérifient des propriétés qui sont transparentes du point de vue des morphismes, mais qui rajoutent de la structure aux objets de la catégorie.

Nous montrons maintenant comment se servir de la catégorie **Refl** comme d'un modèle du λ-calcul avec contraction et affaiblissement.

L'idée est d'utiliser le produit comme opération binaire à la place du tenseur, ce qui assure l'affaiblissement. Il suffit alors de se débrouiller pour que la « cohabitation » entre ce produit et le Hom-foncteur se passe correctement pour disposer de la β-réduction.

Sur, **Refl**, nous considérons les opérations suivantes :

- le tenseur qui sert à interpréter les contextes est le produit ordinaire des multi-graphes réflexifs ; on notera à ce propos que le tenseur n'est pas un produit catégorique : ceci est dû au fait que les morphismes ne respectent pas nécessairement la réflexivité.
- l'objet « $X \multimap Y$ » est celui de la SMC. Nous lui attribuons la fonction de réflexivité donnée par la proposition 6.1.14.
- le terminal est $\mathbf{1}$.

Nous interprétons un contexte $\Gamma = x_1 : X_1, \ldots, x_n : X_n$ comme suit : $\Gamma = \mathrm{X}_1 \times \cdots \times \mathrm{X}_n$; nous interprétons le contexte vide à l'aide du graphe $\mathbf{1}$.

La construction du morphisme `eval` et de la transformation naturelle $\mathbf{Refl}(\Gamma \times X, Y) \to \mathbf{Refl}(\Gamma, X \multimap Y)$ se fait à l'aide du *retract* : $X \otimes Y \underset{i_{XY}}{\overset{p_{XY}}{\rightleftarrows}} X \times Y$, c'est-à-dire que $i_{XY} \circ p_{XY} = 1_{X \times Y}$. Pour tout $p : x - y \in X$ et tout $q : z - t \in Y$,

$$
\begin{array}{rcl}
p_{XY} : X \otimes Y & \to & X \times Y \\
(x,y) & \mapsto & (x,y) \\
(p,q) & \mapsto & (p,q) \\
(=_x, q) & \mapsto & (\mathtt{Ref}_X x, q) \\
(p, =_z) & \mapsto & (p, \mathtt{Ref}_Y z)
\end{array}
\qquad
\begin{array}{rcl}
i_{XY} : X \times Y & \to & X \otimes Y \\
(x,y) & \mapsto & (x,y) \\
(p,q) & \mapsto & (p,q)
\end{array}
$$

On peut étendre le retract aux contextes en composant les morphismes p et i :

$$\Gamma_\otimes = \mathrm{X}_1 \otimes \cdots \otimes \mathrm{X}_n \underset{i_\Gamma}{\overset{p_\Gamma}{\rightleftarrows}} \mathrm{X}_1 \times \cdots \times \mathrm{X}_n = \Gamma$$

La currification des morphismes est définie comme suit. Si le contexte Γ n'est pas vide, étant donné un morphisme $f : \Gamma \times \mathrm{X} \to \mathrm{Y}$, nous définissons $\mathrm{cur}(f) = i_\Gamma \circ \mathrm{cursmc}(f \circ p_{\Gamma \times \mathrm{X}}) : \Gamma \to (X \multimap Y)$ où cursmc est la currification de la SMC.

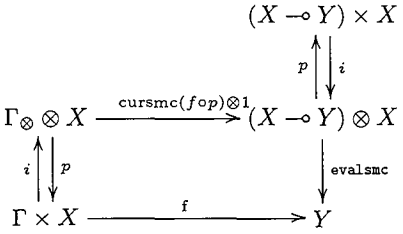
Si le contexte est vide, on associe à tout morphisme $f : \mathbf{1} \times \mathrm{X} \to \mathrm{Y}$ le morphisme $\mathrm{cur}(f) : \mathbf{1} \to (X \multimap Y)$ défini par $(f, \mathtt{Ref}_f)$.

Note 6.1.4. On peut donner une version explicite de la currification : étant donné un morphisme $h : X \times Y \to Z$, le morphisme currifié est :

$$
\begin{array}{rcl}
\mathrm{cur}(h) : & X \to (Y \multimap Z) & \\
x \mapsto & \left\{ \begin{array}{rcl} y & \mapsto & h(x,y) \\ q : y - y' & \mapsto & h(\mathtt{Ref}_X(x), q) : h(x,y) - h(x,y') \end{array} \right. & \\
p : x - x' \mapsto & \left\{ \begin{array}{rcl} y & \mapsto & h(p, \mathtt{Ref}_Y(y)) : h(x,y) - h(x',y) \\ q : y - y' & \mapsto & h(p,q) : h(x,y) - h(x',y') \end{array} \right. &
\end{array}
$$

Le morphisme $\mathtt{eval}_{X,Y}$ est donné par $\mathtt{evalsmc}_{X,Y} \circ \mathtt{i}$ où evalsmc est l'évaluation de la SMC.

Nous avons alors la propriété $\mathtt{eval} \circ (\mathrm{cur}(f) \times 1) = f$.



$$
\begin{array}{rcl}
f & = & f \circ p \circ i \\
& = & \mathtt{evalsmc} \circ (\mathrm{cursmc}(\mathtt{f} \circ \mathtt{p}) \otimes \mathtt{1}) \circ \mathtt{i} \\
& = & \mathtt{evalsmc} \circ \mathtt{i} \circ \mathtt{p}(\mathrm{cursmc}(\mathtt{f} \circ \mathtt{p}) \otimes \mathtt{1}) \circ \mathtt{i} \\
& = & \mathtt{evalsmc} \circ \mathtt{i} \circ (\mathrm{cur}(\mathtt{f}) \times \mathtt{1})
\end{array}
$$

Remarque 6.1.18. Nous n'avons pas la naturalité de cur en général, ce qui serait nécessaire pour interpréter la substitution dans le calcul (cf section 2.6). La naturalité signifierait que le diagramme suivant commute :

$$\begin{array}{ccc} X & \mathrm{Hom}(X\times Y,Z) & \xrightarrow{\mathrm{cur}_{X,Y,Z}} \mathrm{Hom}(X,Y\multimap Z) \\ f\uparrow & \downarrow{\scriptstyle \mathrm{Hom}(f\times 1,Z)} & \downarrow{\scriptstyle \mathrm{Hom}(f,Y\multimap Z)} \\ Y & \mathrm{Hom}(X'\times Y,Z) & \xrightarrow{\mathrm{cur}_{X',Y,Z}} \mathrm{Hom}(X',Y\multimap Z) \end{array}$$

Or, étant donné $h : X \times Y \to Z$, nous pouvons calculer explicitement le morphisme $\mathrm{Hom}(f,Y \multimap Z) \circ \mathrm{cur}_{X,Y,Z}(h) \in \mathrm{Hom}(X',Y \multimap Z)$:

$$\begin{array}{rcl} X' & \to & (Y \multimap Z) \\ x' & \mapsto & \left\{ \begin{array}{rcl} y & \mapsto & h(fx',y) \\ q : y - y' & \mapsto & h(\mathtt{Ref}_X(fx'),q) : h(fx',y) - h(fx',y') \end{array} \right. \\ p : x - x' & \mapsto & \left\{ \begin{array}{rcl} y & \mapsto & h(fp,\mathtt{Ref}_Y(y)) : h(fx,y) - h(fx',y) \\ q : y - y' & \mapsto & h(fp,q) : h(fx,y) - h(fx',y') \end{array} \right. \end{array}$$

Nous pouvons faire la même chose pour le morphisme de la currification $\mathrm{cur}_{X',Y,Z}(\mathrm{Hom}(f \times 1,Z)(h))$:

$$\begin{array}{rcl} X' & \to & (Y \multimap Z) \\ x' & \mapsto & \left\{ \begin{array}{rcl} y & \mapsto & h(fx',y) \\ q : y - y' & \mapsto & h(f\mathtt{Ref}_{X'}(x'),q) : h(fx',y) - h(fx',y') \end{array} \right. \\ p : x - x' & \mapsto & \left\{ \begin{array}{rcl} y & \mapsto & h(fp,\mathtt{Ref}_Y(y)) : h(fx,y) - h(fx',y) \\ q : y - y' & \mapsto & h(fp,q) : h(fx,y) - h(fx',y') \end{array} \right. \end{array}$$

On s'aperçoit que la composante sur les arêtes de $\mathrm{cur}_{X',Y,Z}(\mathrm{Hom}(f \times 1,Z)(h))(x')$ est différente de celle de $\mathrm{Hom}(f,Y \multimap Z) \circ \mathrm{cur}_{X,Y,Z}(h) \in \mathrm{Hom}(X',Y \multimap Z)(x')$. On a égalité seulement si $f(\mathtt{Ref}_X(x')) = \mathtt{Ref}_X(fx')$.

Pour cette raison, nous n'avons pas la substitution en général, mais cela n'est pas très grave comme nous allons le voir.

Propriété 6.1.19.

(i) Du fait que les projections $X \times Y \to X$ respectent la réflexivité, nous pouvons interpréter la règle d'affaiblissement (cf section 2.6) et comme les morphismes diagonaux respectent également les fonctions de réflexivité, nous pouvons utiliser la contraction.

(ii) À cause de l'égalité $\texttt{eval} \circ (\text{cur}(f) \times 1) = f$, nous avons également la β-réduction. En revanche, nous n'avons pas l'unicité du morphisme $\text{cur}(f)$, de ce fait, nous ne pouvons interpréter la η-réduction. Cette propriété qu'un modèle ait la β-réduction sans la η-réduction est peu commune (voir la notion de catégorie semi-cartésienne close [Hay85], et l'exemple de Hoofman [Hoo93]).

Remarque 6.1.20. Nous utilisons la structure (**Refl**, $\times$,$\multimap$) pour définir un calcul monotone (au chapitre 7), mais pour faire de l'algèbre universelle, nous utilisons la structure symétrique monoïdale fermée pour laquelle la théorie est bien établie. Il est en effet difficile de voir ce que signifient les notions de base d'algèbre universelle dans un contexte aussi peu familier que ce modèle de calcul.

6.1.4 Ordres constructifs

Définition 6.1.21. Un *ordre constructif* est un triplet $(G,\texttt{ref}(-),\texttt{tr}(-,-))$ où $(G,\texttt{ref}(-))$ est un multi-graphe réflexif et $\texttt{tr}(-,-)$ est une fonction partielle $R_G \times R_G \rightharpoonup R_G$ tels que pour toutes arêtes $p : a - b \in G$ et $q : b - c \in G$, nous avons $\texttt{tr}(p,q) : a - c \in G$.

En d'autres termes, G est une catégorie sans les lois de l'unité ni celles de la composition.

Par extension, on dit d'un multi-graphe G que c'est un ordre constructif s'il existe deux fonctions `ref` et `tr` telles que $(G,\texttt{ref}(-),\texttt{tr}(-,-))$ soit un ordre constructif.

Remarque 6.1.22. Un ordre constructif qui est squelettique s'identifie à un pré-ordre ordinaire. Par conséquent, pour tout ordre constructif $(G,\texttt{ref}(-),\texttt{tr}(-,-))$, le graphe $\check{G}$ est un pré-ordre ordinaire.

6.1.5 Constructions sur les multi-graphes

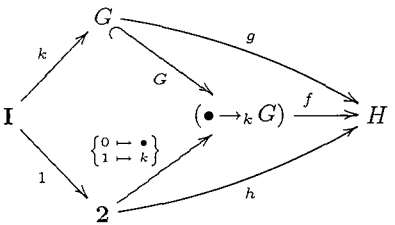
Remarque 6.1.23. La catégorie **MGr** admet les pushouts (cf [BW90] et la section 2.7.2).

Remarque 6.1.24. Nous introduisons ici une notation particulière pour certains pushouts dont nous aurons besoin par la suite. Étant

donné un multi-graphe G et un sommet $k \in G$, le pushout du diagramme $\mathbf{2} \xleftarrow{1} \mathbf{I} \xrightarrow{k} G$ noté $(\bullet \rightarrow_k G)$ est le multi-graphe composé du multi-graphe G et d'un nouveau sommet $\bullet$ qui est relié par une arête $\overline{k}$ au sommet k (par la suite nous notons $\overline{k}$ de la même manière que k).

- $|(\bullet \rightarrow_k G)| = |G| + \{\bullet\}$,
- $R_{(\bullet \rightarrow_k G)} = R_G + \{k\}$, les éléments de R_G ont le domaine et le codomaine qu'ils avaient dans G ; nous rajoutons deux équations pour la nouvelle arête k : $dom(k) = \bullet$, $codom(k) = k$.

C'est alors un simple exercice que de montrer que pour tous morphismes de multi-graphes $h : \mathbf{2} \rightarrow H$ et $g : G \rightarrow H$ tels que $g \circ k = h \circ 1$, il existe un morphisme (unique) f tel que le diagramme commute :



Ici, G dénote à la fois le multi-graphe G et l'injection canonique entre G et $(\bullet \rightarrow_k G)$.

Définition 6.1.25. Soit un diagramme $D \xleftarrow{l} K \xrightarrow{r} E$. Nous construisons le multi-graphe $[DKE]$ [28] :

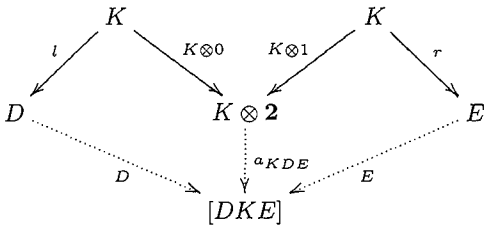
- $|[DKE]| = |D| + |E|$, la somme disjointe de $|D|$ et $|E|$,
- $R_{[DKE]} = R_D + R_E + |K| + R_K$. Le domaine et le codomaine des arêtes sont définis comme suit : si $p \in R_D$ ou $p \in R_E$, p a le domaine et le codomaine qu'il avait dans D ou E ; si $p \in |K|$, nous définissons $dom(p) = l(p)$ (c'est-à-dire l'élément $dom(p) \in |D|$ vu dans $|[DKE]|$) et $codom(p) = r(p)$ (l'élément est vu dans

28. pour simplifier la notation, nous supprimons (de la notation) les deux morphismes l et r mais nous attirons l'attention sur le fait qu'ils participent à la définition.

$|[DKE]|)$; si $p \in R_K$, $dom(p) = dom_D(l(p))$ et $codom(p) = codom_E(r(p))$.

Il y a deux injections canoniques : celle de D dans $[DKE]$ et celle de E dans $[DKE]$. Nous les notons : $D \overset{D}{\hookrightarrow} [DKE]$ et $E \overset{E}{\hookrightarrow} [DKE]$.

Remarque 6.1.26. Nous pouvons alors noter que le multi-graphe $[DKE]$ est la colimite du diagramme :



Le morphisme a_{DKE} est défini par :

$$
\begin{array}{rrcl}
a_{DKE} : & K \otimes \mathbf{2} & \to & [DKE] \\
 & (k,0) & \mapsto & l(k) \\
 & (k,1) & \mapsto & r(k) \\
 & (=_k,(0,1)) & \mapsto & k \text{ vu comme une arête de } [DKE] \\
 & (p,=_0) & \mapsto & l(p) \\
 & (p,=_1) & \mapsto & r(p) \\
 & (p,(0,1)) & \mapsto & p \text{ vu comme une arête de } [DKE]
\end{array}
$$

6.2 Ψ-algèbres

Cette section est au cœur de la problématique. Nous présentons ici une notion d'algèbre sur les multi-graphes (les algèbres Ψ) qui permettra de définir de nombreux ordres tel que ω, l'ordre de Kruskal sur les termes, etc. Nous définissons dans un premier temps les Ψ-algèbres libres.

Définition 6.2.1. Une Ψ-théorie est la donnée de :

(i) un ordre strict $(I, \prec)$ appelé l'ensemble des indices ; ce sont les

indices des opérateurs : pour chaque $\imath \in I$, $\Psi_\imath$ est un symbole d'opérateur dont l'*arité* est le multi-graphe $D_\imath$;

(ii) un sous-ensemble $J \subset I$ clos par le haut pour l'ordre $\prec$; et pour tout $\jmath \in J$, un ensemble $\Delta_\jmath \subset D_\jmath$. Pour tout $k \in \Delta_\jmath$, le symbole d'opérateur $\Psi_\jmath$ est dit *inflationnaire* en k.

Intuitivement, cela signifie que pour tout élément $\mathbf{u} : D_\jmath \to G$ (G étant une Ψ-algèbre, cf. définition suivante), il y a une arête entre les deux éléments $\mathbf{u}_k - \Psi_\jmath(\mathbf{u})$;

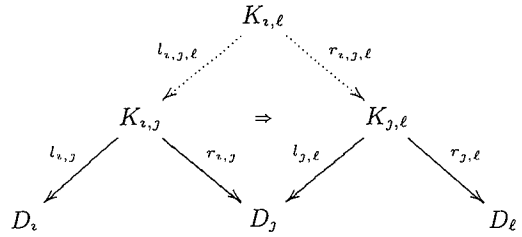
(iii) un diagramme $D_\imath \xleftarrow{l_{\imath,\jmath}} K_{\imath,\jmath} \xrightarrow{r_{\imath,\jmath}} D_\jmath$ pour toute paire $\imath \prec \jmath$. Nous supposons en outre que les deux morphismes $l_{\imath,\jmath}$ et $r_{\imath,\jmath}$ sont des injections (c'est-à-dire que leurs composantes sur les sommets *et* les arêtes sont injectives), et même, à la proposition 6.5.6, nous considérerons que les $K_{\imath,\jmath}$ s'identifient à des sous-ensembles d'arêtes et de sommets.

Intuitivement, cette règle permet de faire des comparaisons entre les opérateurs. Plus précisément, les opérations $\Psi_\imath(\vec{x},\vec{y})$ d'arité $D_\imath$ et $\Psi_\jmath(\vec{x},\vec{z})$ d'arité $D_\jmath$ ont en commun les variables (de sommet et d'arête) $\vec{x}$ qui correspondent à $K_{\imath,\jmath}$, les variables $\vec{y}$ et $\vec{z}$ étant disjointes.

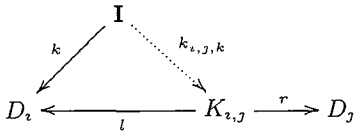
Lorsqu'il est clair quels sont les indices $\imath$ et $\jmath$, nous écrirons l et r à la place de $l_{\imath,\jmath}$ et $r_{\imath,\jmath}$.

Nous supposons en outre que la Ψ-théorie est munie des structures suivantes :

(a) pour tout triplet $\imath \prec \jmath \prec \ell$, il y a deux morphismes $l_{\imath,\jmath,\ell}$ et $r_{\imath,\jmath,\ell}$ tels que $l_{\imath,\ell} = l_{\imath,\jmath} \circ l_{\imath,\jmath,\ell}$ et $r_{\imath,\ell} = r_{\jmath,\ell} \circ r_{\imath,\jmath,\ell}$. En outre, pour chaque $k \in K_{\imath,\ell}$, nous disposons d'une arête : $p_{\imath,\jmath,\ell}(k) : r_{\imath,\jmath} \circ l_{\imath,\jmath,\ell}(k) - l_{\jmath,\ell} \circ r_{\imath,\jmath,\ell}(k) \in D_\jmath$. En résumé, nous avons les morphismes suivants (attention, le diagramme n'est pas commutatif! La flèche $\Rightarrow$ représente la fonction $k \mapsto p_{\imath,\jmath,\ell}(k)$) :



(b) pour tout $\imath \prec \jmath$ avec $\imath \in J$ et tout $k \in \Delta_\imath$, il y a un sommet $k_{\imath,\jmath,k} \in K_{\imath,\jmath}$ et une arête $p_{\imath,\jmath,k} : k - l(k_{\imath,\jmath,k})$. En plus de cela, $r(k_{\imath,\jmath,k}) \in \Delta_\jmath$. C'est pour disposer de cette hypothèse que nous avons supposé que J devait être clos par le haut.

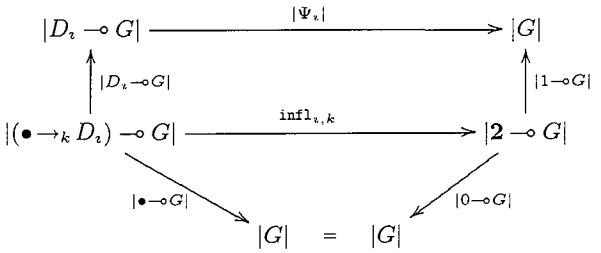


La structure additionnelle que nous avons introduite ici permet de disposer d'une propriété de conservation des ordres constructifs : l'algèbre libre engendrée par (l'ordre sous-jacent d') un ordre constructif peut être étendue en un ordre constructif.

Définition 6.2.2. Une Ψ-algèbre est la donnée d'un quadruplet $(G,(\Psi)_{\imath\in I},(\mathtt{infl})_{\imath\in J,k\in\Delta_\imath},(\mathtt{comp}_{\imath,\jmath})_{\imath\prec\jmath})$ [29] où G est un multi-graphe, où pour tout $\imath \in I$, $\Psi_\imath \in \mathbf{MGr}(D_\imath \multimap G,G)$ est une interprétation du symbole $\Psi_\imath$ et où $(\mathtt{infl})_{\imath\in J,k\in\Delta_\imath}$ et $(\mathtt{comp}_{\imath,\jmath})_{\imath\prec\jmath})$ sont deux familles telles que :

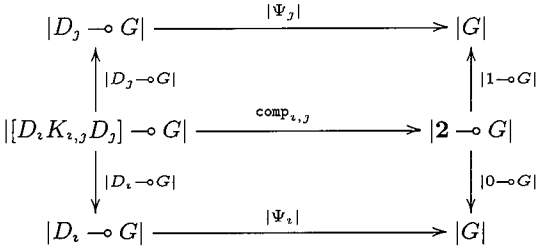
- Pour tout $\imath \in J$ et tout $k \in \Delta_\imath$, $\mathtt{infl}_{\imath,k}$ est une fonction telle que le diagramme commute :

29. Par la suite, nous omettons les indices, pour alléger la notation.



Ce diagramme exprime de manière formelle l'inflationnarité de l'opérateur $\Psi_\imath$ relativement à k. Comme nous l'avons déjà indiqué, cela signifie que la $k^{\text{ème}}$ composante de tout morphisme $\mathbf{u} : D_\imath \multimap G$ vérifie $\mathbf{u}_k - \Psi_\imath \mathbf{u}$. Toujours pour aider le lecteur, nous pouvons remarquer que dans le cas d'un opérateur $\Psi_\imath$ unaire, cela signifie que $x - \Psi_\imath x$.

- Pour tout $\imath \prec \jmath$, $\mathtt{comp}_{\imath,\jmath}$ est une fonction telle que le diagramme commute :



Ce diagramme permet de comparer les opérateurs entre eux. Le diagramme $D_\imath \xleftarrow{l_{\imath,\jmath}} K_{\imath,\jmath} \xrightarrow{r_{\imath,\jmath}} D_\jmath$ fait la correspondance entre $D_\imath$ et $D_\jmath$; les deux morphismes $l_{\imath,\jmath}$ et $r_{\imath,\jmath}$ opèrent des substitutions de variables dans les arbres. Nous revenons sur la construction dans la remarque ci-dessous.

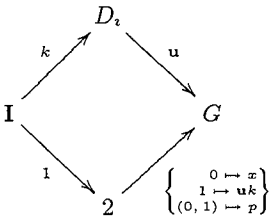
Remarque 6.2.3. Reprenons maintenant plus en détail les diagrammes commutatifs qui précèdent. En particulier, analysons le diagramme pour l'inflationnarité. Soit un élément $\overline{\mathbf{u}} \in (\bullet \to_k D_\imath) \multimap G$. En repre-

nant les définitions, nous voyons que $\overline{\mathbf{u}}$ est la donnée de :

- un morphisme $\mathbf{u} : D_\imath \multimap G$,
- un élément $x \in G$,
- une arête $p : x - \mathbf{u}_k$.

Maintenant, $\mathtt{infl}(\overline{\mathbf{u}})$ est la donnée d'une certaine arête, disons $q : z - t \in G$. Le diagramme exprime alors que $z = x$ et $t = \Psi_\imath \mathbf{u}$.

Inversement, de toute donnée de trois éléments $\mathbf{u}$, x et p, nous pouvons obtenir une preuve de $x - \Psi_\imath \mathbf{u}$. En effet, de $\mathbf{u}$, x et p, nous tirons le diagramme :



Comme $(\bullet \rightarrow_k D_\imath)$ est le pushout correspondant, nous disposons alors d'un morphisme $(\bullet \rightarrow_k D_\imath) \rightarrow G$, c'est-à-dire un élément de $(\bullet \rightarrow_k D_\imath) \multimap G$ sur lequel on peut appliquer le morphisme $\mathtt{infl}$; le résultat produit est une preuve de $x - \Psi_\imath \mathbf{u}$.

Analysons maintenant le deuxième diagramme. En reprenant la définition de $[D_\imath K_{\imath,\jmath} D_\jmath]$, nous voyons qu'un élément $\overline{\mathbf{uv}} \in [D_\imath K_{\imath,\jmath} D_\jmath] \multimap G$ est la donnée de :

- un morphisme $\mathbf{u} : D_\imath \multimap G$,
- un morphisme $\mathbf{v} : D_\jmath \multimap G$,
- pour tout $k \in K$, une arête : $lr(k) : \mathbf{u}_{l(k)} - \mathbf{v}_{r(k)}$,
- pour tout $p : k - j \in K$, une arête : $lr(p) : \mathbf{u}_{l(k)} - \mathbf{v}_{r(j)}$.

D'après le diagramme, l'arête $\mathtt{comp}_{\imath,\jmath}$ est telle que $\mathtt{comp}_{\imath,\jmath}(\overline{\mathbf{uv}}) : \Psi_\imath \mathbf{u} - \Psi_\imath \mathbf{v}$.

Inversement, du fait que $[D_\imath K_{\imath,\jmath} D_\jmath]$ est défini comme une colimite, toute donnée d'un quadruplet comme celui ci-dessus nous donne une preuve que $\Psi_\imath \mathbf{u} - \Psi_\imath \mathbf{v}$.

Définition 6.2.4. Un *homomorphisme entre deux Ψ-algèbres* $(G,\Psi^G,\texttt{infl}^G,\texttt{comp}^G) \to (H,\Psi^H,\texttt{infl}^H,\texttt{comp}^H)$ est un morphisme $f : G \to H$ qui respecte les opérations Ψ, `infl` et `comp`.

6.2.1 Un peu de syntaxe

Pour construire les Ψ-algèbres libres, nous avons besoin de construire des arbres infinis. Pour manipuler de tels arbres, nous mettons en place une syntaxe allégée. Nous représentons les sommets et les arêtes des algèbres à l'aide d'*arbres de dérivation* analogues à ce que l'on peut trouver dans le cadre des systèmes formels. Nous utilisons ainsi, à l'instar du calcul des séquents, des arbres dont les feuilles sont des axiomes (c'est-à-dire des sommets ou des arêtes connus au préalable dans la théorie) et les branchements sont des règles données par le contexte. Un jugement est la conclusion/racine d'un arbre de dérivation.

Le problème principal vient du fait que les arités dont nous disposons ne sont a priori pas finies. Nous avons donc des arbres de taille infinie mais aussi des collections non finies d'arbres. Nous décrivons maintenant les conventions de notation que nous employons pour manipuler de telles collections d'arbres : nous introduisons des règles qui sont transparentes du point de vue de l'induction mais qui du point de vue de la notation permettent d'isoler ou de regrouper des collections d'arbres. Nous en profitons pour présenter dans ce cadre les règles qui permettent d'utiliser le Hom-foncteur de la structure monoïdale.

Étant donné que nous travaillons avec des multi-graphes, nous avons deux sortes de jugements : $k \in G$ signifie que k est (montré/construit) un sommet de G et $p : k - j \in G$ signifie que p est une arête de G telle que $p : k - k'$.

La représentation d'un sommet $\mathbf{u} \in D \multimap X$ consiste en la donnée pour tout sommet $k \in D$, d'un arbre Π_k qui se conclut en $\mathbf{u}_k \in X$ et pour toute arête $p : k-j$ d'un arbre Π_p qui aboutit à $\mathbf{u}_p : \mathbf{u}_k - \mathbf{u}_j \in X$. Nous avons choisi deux notations pour décrire de telles collections. $k : D \vdash \mathbf{u}_k \in X$ (resp. $p : k - j \in D \vdash \mathbf{u}_k \in X$) désigne la collection des arbres $(\Pi_k)_{k\in D}$ (resp. la collection des arbres $(\Pi_p)_{p\in D}$). Dans le cas particulier des collections qui décrivent un morphisme $\mathbf{u} \in D \multimap X$, nous utilisons la notation $\mathbf{u} \in D \multimap X$ pour désigner l'union des deux collections $(\Pi_k)_{k\in K} \cup (\Pi_p)_{p\in K}$. Nous avons une meta-règle qui permet

de passer de la première notation à la seconde :

$$\frac{k \in D \vdash \mathbf{u}_k \in X \qquad p : k - k' \in D \vdash \mathbf{u}_p : \mathbf{u}_k - \mathbf{u}_{k'} \in X}{\mathbf{u} \in D \multimap X}$$

Remarque 6.2.5. Il y a lieu de noter que l'écriture $\mathbf{u} \in D \multimap X$ désigne à la fois le jugement et la dérivation qui aboutit à ce jugement. Cette confusion n'est pas due au hasard. En effet, la conclusion est entièrement déterminée par les prémices et la règle de dérivation. Elle n'est de ce point de vue qu'une sorte de décoration de l'arbre. Mais nous voyons également la conclusion comme le témoin de la construction de l'arbre, elle représente alors l'arbre de la dérivation.

Nous avons une règle analogue à la précédente pour les arêtes :

$$\frac{\mathbf{u} \in D \multimap X \qquad \mathbf{v} \in D \multimap X \qquad \forall k : D \vdash q_k^+ : \mathbf{u}_k - \mathbf{v}_k \qquad \forall p : k - k' \in D \vdash q_p^* : \mathbf{u}_k - \mathbf{v}_{k'}}{\lambda p.q : \mathbf{u} - \mathbf{v} \in D \multimap X}$$

L'abréviation $\lambda p.q$ dénote la paire de fonction (p^+, p^*) à la manière de ce que l'on fait pour les foncteurs.

Le parallèle avec la logique peut être prolongé : nous avons deux règles d'« élimination » qui correspondent à un choix d'une branche d'un arbre. Pour désigner la $k^{\text{ème}}$ branche d'un arbre $\mathbf{u} \in D \multimap X$, (resp. la $p^{\text{ème}}$ branche), nous écrivons :

$$\frac{\mathbf{u} \in D \multimap X \quad k \in D}{\mathbf{u}_k \in X} \qquad \frac{\mathbf{u} \in D \multimap X \quad p : k - k' \in D}{\mathbf{u}_p : \mathbf{u}_k - \mathbf{u}_{k'} \in X}$$

La dernière abréviation syntaxique que nous utilisons est la suivante. Étant donné un morphisme $f \in K \to D$ et une dérivation Π qui aboutit au jugement $\mathbf{u} \in D \multimap X$, la notation $\mathbf{u} \circ f$ désigne la collection des branches de Π qui aboutissent à $\mathbf{u}_{f(k)}$ pour tout sommet $k \in K$ et des branches de Π qui concluent $\mathbf{u}_{f(p)}$ pour toute arête $p \in K$.

De même, la notation $p : \mathbf{u} \circ l - \mathbf{u} \circ r \in K \multimap X$ (où $l \in K \to D$ et $r \in K \to E$ sont des morphismes et $\mathbf{u} \in D \multimap X$ et $\mathbf{v} \in E \multimap X$ sont deux dérivations) désigne la collection des branches de p qui aboutissent à $p_k : \mathbf{u}_{l(k)} - \mathbf{v}_{r(k)} \in X$ pour tout $k \in K$ et des branches qui concluent

$p_q : \mathbf{u}_{l(k)} - \mathbf{v}_{r(j)} s \in X$ pour tout $q : k - j \in K$.

6.2.2 Construction de la Ψ-algèbre libre

Définition 6.2.6. Étant donné un multi-graphe B, nous nous proposons de construire la Ψ-algèbre libre engendrée par B, ou de manière équivalente, la Ψ-algèbre libre qui a comme constantes les sommets et les arêtes de B. Pour cela, nous construisons le multi-graphe $U = F(B)$ défini comme étant le plus petit multi-graphe vérifiant les règles :

$$\frac{b \in B}{\Phi b \in U} \qquad \frac{\mathbf{u} \in D_\imath \multimap U}{\Psi_\imath \mathbf{u} \in U} \, (\imath \in I)$$

Ces règles construisent les sommets du multi-graphe U. La première règle correspond à l'inclusion (universelle) des sommets de B dans le multi-graphe et la deuxième règle introduit l'opérateur $\Psi_\imath$ d'arité $D_\imath$. Notons en ce qui concerne la syntaxe que $b \in B$ est un axiome, et que $\mathbf{u} \in D_\imath \multimap U$ est une dérivation qui conclut $\mathbf{u} \in D_\imath \multimap U$.

Pour les arêtes :

$$\frac{p : b - c \in B}{\Phi p : \Phi b - \Phi c \in U} \qquad \frac{p : \mathbf{u} - \mathbf{v} \in D_\imath \multimap U}{\Psi_\imath p : \Psi_\imath \mathbf{u} - \Psi_\imath \mathbf{v} \in U} \, (\imath \in I)$$

$$\frac{\mathbf{u} \in D_\imath \multimap U \quad p : x - \mathbf{u}_k \in U}{\mathtt{infl}(p,\mathbf{u},\imath,k) : x - \Psi_\imath \mathbf{u} \in U} \, (\imath \in J, k \in \Delta_\imath)$$

$$\frac{\mathbf{u} \in D_\imath \multimap U \quad \mathbf{v} \in D_\jmath \multimap U \quad p : \mathbf{u} \circ l - \mathbf{v} \circ r \in K_{\imath,\jmath} \multimap U}{\mathtt{comp}(p,\mathbf{u},\mathbf{v},\imath,\jmath) : \Psi_\imath \mathbf{u} - \Psi_\jmath \mathbf{v} \in U} \, (\imath \prec \jmath)$$

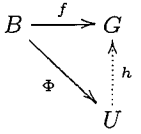
Deux remarques :

- la première règle correspond à l'inclusion (universelle) des arêtes du multi-graphe B. La deuxième fixe la monotonicité de l'opé-

rateur Ψ_i. La troisième régit l'inflationnarité de Ψ_i relativement à k. Quant à la quatrième, elle correspond à la comparaison des opérations.
- nous appelons la dernière règle d'un élément ou d'une arête de U sa *forme*.

Nota bene 6.2.7. Les arbres obtenus peuvent être infinis, mais en fait, ils sont « infinis en largeur » : chaque branche, elle, est de longueur finie. Tout en manipulant des arbres de taille infinie, nous bénéficions donc d'un principe d'induction.

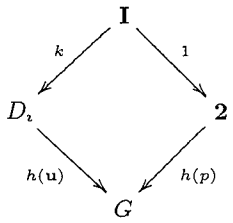
Théorème 6.2.7.1.
Le multi-graphe U est la Ψ-algèbre libre engendrée par B. C'est-à-dire que pour toute autre Ψ-algèbre $(G,\Psi^G,\mathtt{infl}^G,\mathtt{comp}^G)$ *et tout morphisme de multi-graphes $B \to G$, il existe un unique morphisme de Ψ-algèbres h, nous l'appelerons l'épine*[30], *tel que le diagramme commute :*



Démonstration. Nous définissons l'épine h par induction sur la construction de U.

- $h(\Phi b) = fb$,
- $h(\Psi_i \mathbf{u}) = \Psi_i^G(\lambda k : D_i.h(\mathbf{u}_k),\lambda q : k - k'.h(\mathbf{u}_q))$,
- $h(\Phi p : \Phi b - \Phi c) = fp : fb - fc$,
- $h(\Psi_i \mathbf{p} : \Psi_i \mathbf{u} - \Psi_i \mathbf{v}) = \Psi_i^X(\lambda k.h(\mathbf{p}_k),\lambda q : k - k'.h(\mathbf{p}_q)) : h(\Psi_i \mathbf{u}) - h(\Psi_i \mathbf{v})$,
- $h(\mathtt{infl}(p,\mathbf{u},i,k))$. Par induction, nous construisons le morphisme de multi-graphe $h(\mathbf{u}) = (\lambda k.h(\mathbf{u}_k),\lambda q : k - k'.h(\mathbf{u}_q)) : D_i \multimap G$. Notons qu'au vu de la définition de h nous avons $\Psi_i^G(h(\mathbf{u})) = h(\Psi_i^U(\mathbf{u}))$. Par induction, nous avons également une arête $h(p) : h(x) - h(\mathbf{u}_k)$. Nous observons alors que le diagramme commute :

30. La terminologie est due à Simmons.



Comme $(\bullet \rightarrow_k D_\imath)$ est le push-out du diagramme $D_\imath \xleftarrow{k} \mathbf{I} \xrightarrow{1} \mathbf{2}$, nous disposons d'un morphisme de multi-graphes (unique) $\mathbf{h} : (\bullet \rightarrow_k D_\imath) \rightarrow X$ tel que $\mathtt{infl}_{\imath,k}(\mathbf{h}) : h(x) - \Psi_\imath^G h(\mathbf{u})$.

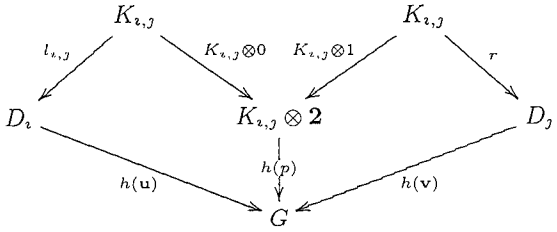
Nous définissons en conséquence : $h(\mathtt{infl}(p,\mathbf{u},\imath,k)) = \mathtt{infl}_{\imath,k}(\mathbf{h}) : h(x) - h(\Psi_\imath \mathbf{u})$.

- $h(\mathtt{comp}(p,\mathbf{u},\mathbf{v},\imath,\jmath))$. Par induction, nous construisons $h(\mathbf{u}) = (\lambda k : D_\imath.h(\mathbf{u}_k), \lambda q : k - k' : D_\imath.h(\mathbf{u}_q)) : D_\imath \multimap G$, $h(\mathbf{v}) = (\lambda k : D_\jmath.h(\mathbf{u}_k), \lambda q : k - k' : D_\jmath.h(\mathbf{u}_q)) : D_\jmath \multimap G$ pour lesquels nous avons $\Psi_\imath^G(h(\mathbf{u})) = h(\Psi_\imath^U(\mathbf{u}))$ et $\Psi_\jmath^G(h(\mathbf{u})) = h(\Psi_\jmath^U(\mathbf{v}))$.

  Par induction, nous pouvons définir le morphisme :

$$
\begin{array}{lrcl}
h(p) : & K_{\imath,\jmath} \otimes \mathbf{2} & \rightarrow & G \\
& (k,0) & \mapsto & h(\mathbf{u}_{l(k)}) \\
& (k,1) & \mapsto & h(\mathbf{v}_{r(k)}) \\
& (=_k, (0,1)) & \mapsto & h(p_k) : h(\mathbf{u}_{l(k)}) - h(\mathbf{v}_{r(k)}) \\
& (q : k - j, =_0) & \mapsto & h(\mathbf{u}_{l(q)}) : h(\mathbf{u}_{l(k)}) - h(\mathbf{u}_{l(j)}) \\
& (q : k - j, =_1) & \mapsto & h(\mathbf{v}_{r(q)}) : h(\mathbf{v}_{r(k)}) - h(\mathbf{v}_{r(j)}) \\
& (q : k - j, (0,1)) & \mapsto & h(p_q) : h(\mathbf{u}_{l(k)}) - h(\mathbf{v}_{l(j)})
\end{array}
$$

Nous observons alors que le diagramme suivant commute :



Ceci nous donne un (unique) morphisme $\mathbf{h} : [D_i K_{i,j} D_j] \multimap G$ tel que $\mathtt{comp}(\mathbf{h},h(\mathbf{u}),h(\mathbf{v}),i,j) : \Psi_i^G h(\mathbf{u}) - \Psi_j^G h(\mathbf{v})$.
Nous définissons alors $h(\mathtt{comp}(p,\mathbf{u},\mathbf{v},i,j)) = \mathtt{comp}^G(\mathbf{h},h(\mathbf{u}),h(\mathbf{v}),i,j) : h(\Psi_i^U \mathbf{u}) - h(\Psi_j^U \mathbf{v})$.

□

6.3 Préservation de la structure d'ordre constructif

Nous montrons dans cette section que la construction des Ψ-algèbres conserve la structure d'ordre constructif.

Théorème 6.3.0.2.
Si $(B,\mathtt{ref}(-),\mathtt{tr}(-,-))$ *est un ordre constructif, le multi-graphe* $U = F(B)$ *peut être étendu en un ordre constructif. Plus précisément, il existe une fonction de réflexivité* `Ref` *et une fonction de transitivité* `Tr` *telles que* $(U,\mathtt{Ref},\mathtt{Tr})$ *soit un ordre constructif.*

Démonstration. Nous construisons en premier lieu la fonction de réflexivité `Ref` sur U. Nous procédons par induction sur la définition des sommets $x \in U$:

- si $x = \Phi b$. Comme B est un ordre constructif, nous disposons d'une preuve : $\mathtt{ref}(b) : b \leq b \in B$. Par conséquent,

$$\frac{\mathtt{ref}(b) : b \leq b \in B}{\Phi\mathtt{ref}(b) : \Phi b - \Phi b \in U}$$

Nous définissons donc $\mathtt{Ref}(\Phi b) = \Phi\mathtt{ref}(b)$;

- si $x = \Psi_i \mathbf{u}$ avec $\mathbf{u} : D_i \multimap U$. Par hypothèse d'induction, nous avons pour tout $k \in D_i$, $\mathtt{Ref}(\mathbf{u}_k) : \mathbf{u}_k - \mathbf{u}_k$. Nous observons :

$$\dfrac{\dfrac{\text{induction}}{k \in D_i \vdash \mathtt{Ref}(\mathbf{u}_k) : \mathbf{u}_k - \mathbf{u}_k} \quad \dfrac{\mathbf{u} : D_i \multimap U}{q : k - j \in D_i \vdash \mathbf{u}_q : \mathbf{u}_k - \mathbf{u}_j}}{(\lambda k.\mathtt{Ref}(\mathbf{u}_k), \lambda q.\mathbf{u}_q) : \mathbf{u} - \mathbf{u}}}{\Psi_i(\lambda k.\mathtt{Ref}(\mathbf{u}_k), \lambda q.\mathbf{u}_q) : \Psi_i \mathbf{u} - \Psi_i \mathbf{u}}$$

Ce qui nous permet de conclure : $\mathtt{Ref}(\Psi_i \mathbf{u}) = \Psi_i(\lambda k : D_i.\mathtt{Ref}(\mathbf{u}_k), \lambda q : k - j.\mathbf{u}_q)$. (Pour éviter que la dérivation soit trop volumineuse, nous n'avons pas noté ce à quoi appartiennent les objets que nous manipulons).

Nous abordons maintenant le problème de la transitivité. En écrivant les arêtes $p : x - y$ sous la forme $x \vdash p : y$, la transitivité apparaît très similaire à un théorème d'élimination des coupures.

Nous construisons maintenant la fonction de transitivité. Pour cela, nous utilisons une induction sur les (structures des) paires $(p : x - y \in U, q : y - z \in U)$. Notons que le nombre de paires est limité du fait que y apparaît des deux cotés des inégalités.

- $(\dfrac{p : b \leq c \in B}{\Phi p : \Phi b - \Phi c \in U}, \dfrac{q : c \leq d \in B}{\Phi q : \Phi c - \Phi d \in U})$. Comme B est un ordre constructif, nous disposons d'une

  preuve : $\mathtt{tr}(p,q) : b \leq d$.

  En résumé : $\mathtt{Tr}(\Phi p, \Phi q) = \Phi(\mathtt{tr}(p,q))$.

- $(\dfrac{p : b \leq c \in B}{\Phi p : \Phi b - \Phi c \in U}, \dfrac{\mathbf{u} \in D_i \multimap U \qquad q : \Phi c - \mathbf{u}_k \in U}{\mathtt{infl}(q, \mathbf{u}, i, k) : \Phi c - \Psi_i \mathbf{u} \in U})$. Nous construisons par induction :

$$\dfrac{\mathbf{u} \in D_i \multimap U \quad \dfrac{\text{induction}}{\mathtt{Tr}(\Phi(p), q) : \Phi b - \mathbf{u}_k}}{\mathtt{infl}(\mathtt{Tr}(\Phi(p), q), \mathbf{u}, i, k) : \Phi b - \Psi_i \mathbf{u}}$$

D'où l'on définit : $\mathtt{Tr}(\Phi(p), \mathtt{infl}(q, \mathbf{u}, i, k)) = \mathtt{infl}(\mathtt{Tr}(\Phi(p), q), \mathbf{u}, i, k)$.

- $(\dfrac{p : \mathbf{u} - \mathbf{v} \in D_i \multimap U}{\Psi_i p : \Psi_i \mathbf{u} - \Psi_i \mathbf{v} \in U}, \dfrac{q : \mathbf{v} - \mathbf{w} \in D_i \multimap U}{\Psi_i q : \Psi_i \mathbf{v} - \Psi_i \mathbf{w} \in U})$. Nous construisons par induction :

$$\dfrac{\dfrac{\text{induction}}{k \in D_i \vdash \mathtt{Tr}(p_k,q_k) : \mathbf{u}_k - \mathbf{w}_k} \quad \dfrac{\text{induction}}{r : k - j \in D_i \vdash \mathtt{Tr}(p_r,q_j) : \mathbf{u}_k - \mathbf{w}_j}}{\Psi_i(\lambda k.\mathtt{Tr}(p_k,q_k),\lambda r : k - j.\mathtt{Tr}(p_r,q_j)) : \Psi_i \mathbf{u} - \Psi_i \mathbf{w} \in U}$$

D'où l'on définit : $\mathtt{Tr}(\Psi_i p,\Psi_i q) = \Psi_i(\lambda k.\mathtt{Tr}(p_k,q_k),\lambda r : k-j.\mathtt{Tr}(p_r,q_j))$.

- si $(\dfrac{p : \mathbf{u} - \mathbf{v} \in D_i \multimap U}{\Psi_i p : \Psi_i \mathbf{u} - \Psi_i \mathbf{v} \in U}, \dfrac{\mathbf{w} \in D_j \multimap U \qquad q : \Psi_i \mathbf{v} - \mathbf{w}_k \in U}{\mathtt{infl}(q,\mathbf{w},j,k) : \Psi_i \mathbf{v} - \Psi_j \mathbf{w}})$. Nous construisons par induction :

$$\dfrac{\mathbf{w} \in D_j \multimap U \quad \dfrac{\text{induction}}{\mathtt{Tr}(\Psi_i p,q) : \Psi_i \mathbf{u} - \mathbf{w}_k}}{\mathtt{infl}(\mathtt{Tr}(\Psi_i p,q),\mathbf{w},j,k) : \Psi_i \mathbf{u} - \Psi_j \mathbf{w}}$$

D'où l'on définit : $\mathtt{Tr}(\Psi_i p,\mathtt{infl}(q,\mathbf{w},i,k)) = \mathtt{infl}(\mathtt{Tr}(\Psi_i p,q),\mathbf{w},j,k)$.

- $(\dfrac{p : \mathbf{u} - \mathbf{v} \in D_i \multimap U}{\Psi_i p : \Psi_i \mathbf{u} - \Psi_i \mathbf{v} \in U}, \dfrac{\mathbf{v} \in D_i \multimap U \qquad \mathbf{w} \in D_j \multimap U \qquad q : \mathbf{v} \circ l - \mathbf{w} \circ r \in K_{i,j} \multimap U}{\mathtt{comp}(q,\mathbf{v},\mathbf{w},i,j) : \Psi_i \mathbf{v} - \Psi_j \mathbf{w}})$.
Nous construisons par induction :

$$\dfrac{\mathbf{u} \in D_i \multimap U \quad \mathbf{w} \in D_j \multimap U \quad \dfrac{\dfrac{\text{induction}}{k \in K_{i,j} \vdash \mathtt{Tr}(p_{l(k)},q_k) : \mathbf{u}_{l(k)} - \mathbf{w}_{r(k)}} \quad \dfrac{\text{induction}}{p' : k - j \in K_{i,j} \vdash \mathtt{Tr}(p_{l(p')},q_j) : \mathbf{u}_{l(k)} - \mathbf{w}_{r(j)}}}{(\lambda k.\mathtt{Tr}(p_{l(k)},q_k),\lambda p' : k - j.\mathtt{Tr}(p_{l(p')},q_j)) : \mathbf{u} \circ l - \mathbf{w} \circ r}}{\mathtt{comp}(\lambda k.\mathtt{Tr}(p_{l(k)},q_k),\lambda p' : k - j.\mathtt{Tr}(p_{l(p')},q_j),\mathbf{u},\mathbf{w},i,j) : \Psi_i \mathbf{u} - \Psi_j \mathbf{w}}$$

D'où l'on définit :

$\mathtt{Tr}(\Psi_i p,\mathtt{comp}(q,\mathbf{v},\mathbf{w},i,j)) = \mathtt{comp}(\lambda k.\mathtt{Tr}(p_{l(k)},q_k),\lambda p' : k-j.\mathtt{Tr}(p_{l(p')},q_j),\mathbf{u},\mathbf{w},i,j)$

- $\left(\dfrac{\mathbf{v} \in D_\imath \multimap U \qquad p : x - \mathbf{v}_k \in U}{\mathtt{infl}(p,\mathbf{v},\imath,k) : x - \Psi_\imath \mathbf{v} \in U}, \dfrac{q : \mathbf{v} - \mathbf{w} \in D_\imath \multimap U}{\Psi_\imath q : \Psi_\imath \mathbf{v} - \Psi_\imath \mathbf{w} \in U}\right)$. Nous construisons par induction :

$$\dfrac{\mathbf{w} \in D_\imath \multimap U \qquad \overset{\text{induction}}{\overline{\mathtt{Tr}(p,q_k) : x - \mathbf{w}_k}}}{\mathtt{infl}(\mathtt{Tr}(p,q_k),\mathbf{w},\imath,k) : x - \Psi_\imath \mathbf{w}}$$

D'où l'on définit : $\mathtt{Tr}(\mathtt{infl}(p,\mathbf{v},\imath,k),\Psi_\imath q) = \mathtt{infl}(\mathtt{Tr}(p,q_k),\mathbf{w},\imath,k)$

- $\left(\dfrac{\mathbf{v} \in D_\imath \multimap U \qquad p : x - \mathbf{v}_k \in U}{\mathtt{infl}(p,\mathbf{v},\imath,k) : x - \Psi_\imath \mathbf{v} \in U}, \dfrac{\mathbf{w} \in D_\jmath \multimap U \qquad q : \Psi_\imath \mathbf{v} - \mathbf{w}_{k'} \in U}{\mathtt{infl}(q,\mathbf{w},\jmath,k') : \Psi_\imath \mathbf{v} - \Psi_\jmath \mathbf{w} \in U}\right)$. Nous construisons par induction :

$$\dfrac{\mathbf{w} \in D_\jmath \multimap U \qquad \overset{\text{induction}}{\overline{\mathtt{Tr}(\mathtt{infl}(p,\mathbf{v},\imath,k),q) : x - \mathbf{w}_{k'}}}}{\mathtt{infl}(\mathtt{Tr}(\mathtt{infl}(p,\mathbf{v},\imath,k),q),\mathbf{w},\jmath,k') : x - \Psi_\imath \mathbf{w}}$$

D'où l'on définit :

$$\mathtt{Tr}(\mathtt{infl}(p,\mathbf{v},\imath,k),\mathtt{infl}(q,\mathbf{w},\imath',k')) = \mathtt{infl}(\mathtt{Tr}(\mathtt{infl}(p,\mathbf{v},\imath,k),q),\mathbf{w},\jmath,k')$$

- $\left(\dfrac{\mathbf{v} \in D_\imath \multimap U \qquad p : x - \mathbf{v}_k \in U}{\mathtt{infl}(p,\mathbf{v},\imath,k) : x - \Psi_\imath \mathbf{v} \in U}, \dfrac{\mathbf{v} \in D_\imath \multimap U \qquad \mathbf{w} \in D_\jmath \multimap U \qquad q : \mathbf{v} \circ l - \mathbf{w} \circ r \in K_{\imath,\jmath} \multimap U}{\mathtt{comp}(q,\mathbf{v},\mathbf{w},\imath,\jmath) : \Psi_\imath \mathbf{v} - \Psi_\jmath \mathbf{w} \in U}\right)$.

Nous construisons par induction :

$$\dfrac{\mathbf{w} \in D_\jmath \multimap U \qquad \dfrac{\dfrac{p : x - \mathbf{v}_k \qquad \overset{p_{\imath,\jmath,k} : k - l(k_{\imath,\jmath,k})}{\overline{\mathbf{v}_{p_{\imath,\jmath,k}} : \mathbf{v}_k - \mathbf{v}_{l(k_{\imath,\jmath,k})}}}}{\mathtt{Tr}(p,\mathbf{v}_{p_{\imath,\jmath,k}}) : x - \mathbf{v}_{l(k_{\imath,\jmath,k})}} \qquad \dfrac{q : \mathbf{v} \circ l - \mathbf{w} \circ r}{q_{k_{\imath,\jmath,k}} : \mathbf{v}_{l(k_{\imath,\jmath,k})} - \mathbf{w}_{r(k_{\imath,\jmath,k})}}}{\mathtt{Tr}(\mathtt{Tr}(p,\mathbf{v}_{p_{\imath,\jmath,k}}),q_{k_{\imath,\jmath,k}}) : x - \mathbf{w}_{r(k_{\imath,\jmath,k})}}}{\mathtt{infl}(\mathtt{Tr}(\mathtt{Tr}(p,\mathbf{v}_{p_{\imath,\jmath,k}}),q_{k_{\imath,\jmath,k}}),\mathbf{w},\jmath,r(k_{\imath,\jmath,k})) : x - \Psi_\jmath \mathbf{w}}$$

D'où l'on définit :

$$\mathtt{Tr}(\mathtt{infl}(p,\mathbf{v},i,k),\mathtt{comp}(q,\mathbf{v},\mathbf{w},i,j)) = \mathtt{infl}(\mathtt{Tr}(\mathtt{Tr}(p,\mathbf{v}_{p_{i,j,k}}),q_{k_{i,j,k}}),\mathbf{w},j,r(k_{i,j,k}))$$

- $\left(\dfrac{\mathbf{u} \in D_i \multimap U \qquad \mathbf{v} \in D_j \multimap U \qquad p : \mathbf{u} \circ l - \mathbf{v} \circ r \in K_{i,j} \multimap U}{\mathtt{comp}(p,\mathbf{u},\mathbf{v},i,j) : \Psi_i \mathbf{u} - \Psi_j \mathbf{v} \in U}, \dfrac{q : \mathbf{v} - \mathbf{w} \in D_j \multimap U}{\Psi_j q : \Psi_j \mathbf{v} - \Psi_j \mathbf{w} \in U}\right)$.

Nous construisons par induction :

$$\dfrac{\mathbf{u} \in D_i \multimap U \quad \mathbf{w} \in D_j \multimap U \qquad \dfrac{\dfrac{}{k \in K_{i,j} \vdash \mathtt{Tr}(p_k,q_{r(k)}) : \mathbf{u}_{l(k)} - \mathbf{w}_{r(k)}}\,\text{induction} \qquad \dfrac{}{p' : k - j \in K_{i,j} \vdash \mathtt{Tr}(p_{p'},q_{r(j)}) : \mathbf{u}_{l(k)} - \mathbf{w}_{r(j)}}\,\text{induction}}{(\lambda k.\mathtt{Tr}(p_k,q_{r(k)}),\lambda q.\mathtt{Tr}(p_{p'},q_{r(j)})) : \mathbf{u} \circ l - \mathbf{w} \circ r}}{\mathtt{comp}(\lambda k.\mathtt{Tr}(p_k,q_{r(k)}),\lambda p'.\mathtt{Tr}(p_{p'},q_{r(j)}),\mathbf{u},\mathbf{w},i,j) : \Psi_i \mathbf{u} - \Psi_j \mathbf{w}}$$

D'où l'on définit :

$$\mathtt{Tr}(\mathtt{comp}(p,\mathbf{u},\mathbf{v},l,r),\Psi_j q) = \mathtt{comp}(\lambda k.\mathtt{Tr}(p_k,q_{r(k)}),\lambda p'.\mathtt{Tr}(p_{p'},q_{r(j)},\mathbf{u},\mathbf{w},i,j)$$

- $\left(\dfrac{\mathbf{u} \in D_i \multimap U \qquad \mathbf{v} \in D_j \multimap U \qquad p : \mathbf{u} \circ l - \mathbf{v} \circ r \in K_{i,j} \multimap U}{\mathtt{comp}(p,\mathbf{u},\mathbf{v},i,j) : \Psi_i \mathbf{u} - \Psi_j \mathbf{v} \in U}, \dfrac{\mathbf{w} \in D_\ell \multimap U \qquad q : \Psi_j \mathbf{v} - \mathbf{w}_{k'} \in U}{\mathtt{infl}(q,\mathbf{w},\ell,k') : \Psi_j \mathbf{v} - \Psi_\ell \mathbf{w} \in U}\right)$.

Nous construisons par induction :

$$\dfrac{\mathbf{w} \in D_j \multimap U \qquad \dfrac{}{\mathtt{Tr}(\mathtt{comp}(p,\mathbf{u},\mathbf{v},i,j),q) : \Psi_i \mathbf{u} - \mathbf{w}_{k'}}\,\text{induction}}{\mathtt{infl}(\mathtt{Tr}(\mathtt{comp}(p,\mathbf{u},\mathbf{v},i,j),q),\mathbf{w},\ell,k') : \Psi_i \mathbf{u} - \Psi_i \mathbf{w}}$$

D'où l'on définit :

$$\mathtt{Tr}(\mathtt{comp}(p,\mathbf{u},\mathbf{v},i,j),\mathtt{infl}(q,\mathbf{w},i,k)) = \mathtt{infl}(\mathtt{Tr}(\mathtt{comp}(p,\mathbf{u},\mathbf{v},i,j),q),\mathbf{w},\ell,k')$$

- $\left(\dfrac{\mathbf{u} \in D_i \multimap U \qquad \mathbf{v} \in D_j \multimap U \qquad p : \mathbf{u} \circ l_{i,j} - \mathbf{v} \circ r_{i,j}}{\mathtt{comp}(p,\mathbf{u},\mathbf{v},i,j) : \Psi_i \mathbf{u} - \Psi_j \mathbf{v} \in U}, \dfrac{\mathbf{v} \in D_j \multimap U \qquad \mathbf{w} \in D_\ell \multimap U \qquad q : \mathbf{v} \circ l_{j,\ell} - \mathbf{w} \circ r_{j,\ell}}{\mathtt{comp}(q,\mathbf{v},\mathbf{w},j,\ell) : \Psi_j \mathbf{v} - \Psi_\ell \mathbf{w}}\right)$.

Nous construisons par induction avec $k \in K_{i,\ell}$:

$$\dfrac{\dfrac{p{:}\mathbf{u}\circ l_{\imath,\jmath}{-}\mathbf{v}\circ r_{\imath,\jmath}}{k\vdash p_{l_{\imath,\jmath,\ell}(k)}{:}\mathbf{u}_{l_{\imath,\ell}(k)}{-}\mathbf{v}_{r_{\imath,\jmath}\circ l_{\imath,\jmath,\ell}(k)}} \quad \dfrac{p_{\imath,\jmath,\ell}{:}r_{\imath,\jmath}l_{\imath,\jmath,\ell}{-}l_{\jmath,\ell}r_{\imath,\jmath,\ell}}{k\vdash \mathbf{v}_{p_{\imath,\jmath,\ell}}{:}\mathbf{v}_{r_{\imath,\jmath}\circ l_{\imath,\jmath,\ell}(k)}{-}\mathbf{v}_{l_{\jmath,\ell}\circ r_{\imath,\jmath,\ell}(k)}}}{k\vdash \mathtt{Tr}(p_{l_{\imath,\jmath,\ell}(k)},\mathbf{v}_{p_{\imath,\jmath,\ell}(k)}){:}\mathbf{u}_{l_{\imath,\ell}(k)}{-}\mathbf{v}_{l_{\jmath,\ell}r_{\imath,\jmath,\ell}(k)}} \quad \dfrac{q{:}\mathbf{v}\circ l_{\jmath,\ell}{-}\mathbf{w}\circ r_{\jmath,\ell}}{k\vdash q_{r_{\imath,\jmath,\ell}(k)}{:}\mathbf{v}_{l_{\jmath,\ell}\circ r_{\imath,\jmath,\ell}(k)}{-}\mathbf{w}_{r_{\imath,\ell}(k)}}}{k\vdash \mathtt{Tr}(\mathtt{Tr}(p_{l_{\imath,\jmath,\ell}(k)},\mathbf{v}_{p_{\imath,\jmath,\ell}(k)}),q_{r_{\imath,\jmath,\ell}(k)}){:}\mathbf{u}_{l_{\imath,\ell}(k)}{-}\mathbf{w}_{r_{\imath,\ell}(k)}}$$

Pour les arêtes, avec $p' : k - j \in K_{\imath,\ell}$,

$$\dfrac{\dfrac{p{:}\mathbf{u}\circ l_{\imath,\jmath}{-}\mathbf{v}\circ r_{\imath,\jmath}}{p'\vdash p_{l_{\imath,\jmath,\ell}(k)}{:}\mathbf{u}_{l_{\imath,\ell}(k)}{-}\mathbf{v}_{r_{\imath,\jmath}\circ l_{\imath,\jmath,\ell}(k)}} \quad \dfrac{p_{\imath,\jmath,\ell}{:}r_{\imath,\jmath}l_{\imath,\jmath,\ell}{-}l_{\jmath,\ell}r_{\imath,\jmath,\ell}}{p'\vdash \mathbf{v}_{p_{\imath,\jmath,\ell}}{:}\mathbf{v}_{r_{\imath,\jmath}\circ l_{\imath,\jmath,\ell}(k)}{-}\mathbf{v}_{l_{\jmath,\ell}\circ r_{\imath,\jmath,\ell}(k)}}}{p'\vdash \mathtt{Tr}(p_{l_{\imath,\jmath,\ell}(k)},\mathbf{v}_{p_{\imath,\jmath,\ell}(k)}){:}\mathbf{u}_{l_{\imath,\ell}(k)}{-}\mathbf{v}_{l_{\jmath,\ell}r_{\imath,\jmath,\ell}(k)}} \quad \dfrac{q{:}\mathbf{v}\circ l_{\jmath,\ell}{-}\mathbf{w}\circ r_{\jmath,\ell}}{p'\in K_{\imath,\ell}\vdash q_{r_{\imath,\jmath,\ell}(p')}{:}\mathbf{v}_{l_{\jmath,\ell}\circ r_{\imath,\jmath,\ell}(k)}{-}\mathbf{w}_{r_{\imath,\ell}(j}}}{p'\vdash \mathtt{Tr}(\mathtt{Tr}(p_{l_{\imath,\jmath,\ell}(k)},\mathbf{v}_{p_{\imath,\jmath,\ell}(k)}),q_{r_{\imath,\jmath,\ell}(p')}){:}\mathbf{u}_{l_{\imath,\ell}(j)}{-}\mathbf{w}_{r_{\imath,\ell}(k)}}$$

D'où l'on définit : $\mathtt{Tr}(\mathtt{comp}(p,\mathbf{u},\mathbf{v},\imath,\jmath),\mathtt{comp}(q,\mathbf{v},\mathbf{w},\jmath,\ell)) =$

$$\mathtt{comp}((\lambda k : K_{\imath,\ell}.\mathtt{Tr}(\mathtt{Tr}(p_{l_{\imath,\jmath,\ell}(k)},\mathbf{v}_{p_{\imath,\jmath,\ell}(k)}),q_{r_{\imath,\jmath,\ell}(k)}),\lambda p'.\mathtt{Tr}(\mathtt{Tr}(p_{l_{\imath,\jmath,\ell}(k)},\mathbf{v}_{p_{\imath,\jmath,\ell}(k)}),q_{r_{\imath,\jmath,\ell}(p')})),\mathbf{u},\mathbf{w},\imath,\ell)$$

□

6.4 Applications, exemples

Nous montrons dans cette section que le concept de Ψ-algèbre permet de capturer une grande variété d'ordres (capturer au sens où le collapse de la Ψ-algèbre libre est précisément l'ordre dont il est question). Nous avons essayé de montrer par des exemples suffisamment explicites l'intérêt de chacune des constructions que nous avons faites sur les Ψ-théories.

Exemple 6.4.1. À tout seigneur tout honneur? Un premier exemple sera l'ordinal ω, plus précisément son équivalent constructif (on le note $\boldsymbol{\omega}$). Nous l'obtenons en prenant un opérateur unaire inflationnaire. Plus formellement, nous définissons $I = J = \{\bullet\}$ avec $D_\bullet = \Delta_\bullet = \mathbf{I}$. Pour simplifier, nous utilisons la notation s plutôt que $\Psi_\bullet$, de même, nous laissons tomber les symboles Φ d'inclusion universelle. L'ordre constructif $\boldsymbol{\omega}$ est alors la Ψ-algèbre libre engendrée par une constante

($B = \mathbf{1}$). En effet, le lecteur reconnaîtra les règles d'induction de ω à travers les règles de la construction de la Ψ-algèbre libre :

$$\frac{0 \in B}{0 \in \omega} \qquad \frac{n \in \omega}{sn \in \omega}$$

Le lecteur notera que pour deuxième règle, nous utilisons l'isomorphisme $\mathbf{I} \multimap \omega \simeq \omega$.

$$\frac{1_0 : 0 - 0 \in B}{1_0 : 0 - 0 \in \omega} \qquad \frac{p : n - m \in \omega}{sp : sn - sm \in \omega} \qquad \frac{m : \omega \quad p : n - m \in \omega}{\mathtt{infl}(p,n) : n - sm \in \omega}$$

Un ordinal $n \in \omega$ est alors représenté par :

$$\overline{n} = \underbrace{s(\cdots s(0)\cdots)}_{n \text{ fois } s}$$

C'est un exercice simple de vérifier l'équivalence entre $n \leq m \in \omega$ et l'existence d'une arête $p : \overline{n} - \overline{m} \in \omega$. Il suffit de prendre :

$$\underbrace{\mathtt{infl}(\cdots(\mathtt{infl}}_{m-n \text{ fois } \mathtt{infl}}\underbrace{s(\cdots(s(s(1_0)))\cdots)}_{n \text{ fois } s}\cdots) : \overline{n} - \overline{m}$$

Il y a malgré tout lieu de remarquer que les preuves ne sont pas uniques. Dans l'exemple précédent, toute permutation des opérations `infl` et `s` aboutit à la même conclusion. Par conséquent, il y a $\binom{m}{n}$ preuves que $n \leq m$.

Cette multiplicité va poser problème dès lors que l'on va utiliser une arité non discrète. En effet, plusieurs objets ne seront distingués que par des arêtes (des inégalités) dans leur définition. Or nous utilisons les multi-graphes uniquement pour pouvoir nommer les arêtes (pour pouvoir faire les preuves des inégalités), mais le nom effectif de l'arête n'est d'aucun intérêt pour nous qui nous plaçons dans les graphes. Nous introduisons donc artificiellement des distinctions qui n'ont pas lieu d'être.

Dans la section suivante, nous montrons en quoi le collapse du multi-graphe correspond au graphe dont on a l'intuition.

Exemple 6.4.2. L'ordre $\mathbb{W}$ a un pendant dans les Ψ-algèbres libres. Nous prenons $I = \{\mathbf{s},\mathbf{L}\}$ avec l'ordre $\prec$ discret, $J = \{\mathbf{s}\}$. Nous donnons $D_{\mathbf{s}} = \Delta_{\mathbf{s}} = \mathbf{I}$ et $D_{\mathbf{L}} = \omega$. Enfin, nous prenons $B = \mathbf{1}$. Notons que nous pourrions utiliser l'ordre constructif $\boldsymbol{\omega}$ à la place de ω.

Exemple 6.4.3. Les termes ordinaux dus à Dennis-Jones et Wainer [DJW83] sont définis comme étant le plus petit ensemble $\mathcal{O}$ obtenu par les trois règles :

$$\frac{}{0 \in \mathcal{O}} \qquad \frac{\alpha \in \mathcal{O}}{s\alpha \in \mathcal{O}} \qquad \frac{f : \mathbb{N} \to \mathcal{O}}{f \in \mathcal{O}}$$

L'ordre sur les termes ordinaux est défini comme étant la fermeture transitive de la plus petite relation contenant :

$$\frac{}{0 \leq \alpha}\ (\alpha \in \mathcal{O}) \qquad \frac{}{\alpha \leq s\alpha}\ (\alpha \in \mathcal{O}) \qquad \frac{f : \mathbb{N} \to \mathcal{O}}{\forall n \quad f(n) \leq f}$$

Les termes ordinaux correspondent à la Ψ-théorie suivante. Prenons $I = J = \{\mathbf{s},\mathbf{L}\}$, avec l'ordre $\prec$ discret. Nous ajoutons $D_{\mathbf{s}} = \Delta_{\mathbf{s}} = \mathbf{I}$ et $D_{\mathbf{L}} = \Delta_{\mathbf{L}} = \mathbb{N}$. Enfin, nous prenons $B = \mathbf{1}$.

Remarque 6.4.4. L'ordre $\mathbb{W}_{\trianglelefteq}$ des arborescences (cf definition 5.4.1) s'obtient en supprimant l'équation d'inflationnarité de l'opérateur $\mathbf{L}$ dans l'ordre précédent ; en d'autres termes, on prend $D_{\mathbf{L}} = \mathbb{N}$. Pour les arborescences $\mathbb{W}_{\sqsubseteq^*}$, on choisit $D_{\mathbf{L}} = (\mathbb{N}, \sqsubseteq)$.

Exemple 6.4.5. Le troisième exemple est donné par l'équivalent constructif de l'ordre de Higman sur les mots. Soit un alphabet M, M^* dénote l'ensemble des mots sur M. L'ordre de Higman $\trianglelefteq$[31] sur M^* est défini inductivement par les règles :

$$\frac{}{w \trianglelefteq w}\ (w \in M^*) \qquad \frac{}{w \trianglelefteq aw}\ (w \in M^*, \quad a \in M) \qquad \frac{w \trianglelefteq w'}{aw \trianglelefteq aw'}\ (a \in M)$$

$\trianglelefteq$ est alors l'ordre libre obtenu en prenant $\mathrm{Card}(M)$ opérateurs inflationnaires. Plus précisément, nous prenons $I = J = M$ avec $\prec$

31. À ne pas confondre avec l'ordre $\trianglelefteq$ introduit au chapitre précédent.

l'ordre discret, $D_m = \Delta_m = \mathbf{I}$ pour tout $m \in M$. Un mot $w \in M^*$ est représenté par $\overline{w} \in F(\mathbf{1})$:

- $\overline{\epsilon} = 0$ avec ϵ le mot vide,
- $\overline{aw} = \Psi_a(\overline{w})$ pour tout $a \in M$.

Exemple 6.4.6. Nous pouvons aller un peu plus loin en considérant le cas des termes. Supposons que nous disposons d'un ensemble de symboles Σ ordonné par le pré-ordre $\preceq$. Nous considérons l'ensemble des arbres étiquetés par ces symboles. En fait, nous voyons ces arbres comme des termes. Ceci nous donne une « algèbre de termes » $\mathcal{T}$. L'ordre de Kruskal sur les termes est alors donné par les règles :

$$\frac{}{t_k \trianglelefteq f(t_1, \cdots, t_n)} \ (k \leq n)$$

$$\frac{}{f(t_1, \cdots, t_{i-1}, t_{i+1}, \cdots, t_n) \trianglelefteq f(t_1, \cdots, t_{i-1}, t_i, t_{i+1}, \cdots, t_n)} \ (\forall n, \forall i \leq n)$$

$$\frac{f \preceq g \quad \forall i \leq n, t_i \trianglelefteq t'_i}{f(t_1, \ldots, t_n) \trianglelefteq g(t'_1, \ldots, t'_n)}$$

Pour obtenir un équivalent constructif de l'ordre de Kruskal en termes de Ψ-théorie, nous devons étendre légèrement la notion de Ψ-algèbre. Pour chaque paire $i \prec j$, on autorise plusieurs diagrammes $D_i \xleftarrow{l^\alpha_{i,j}} K^\alpha_{i,j} \xrightarrow{r^\alpha_{i,j}} D_j$. Ceci nous permet de considérer plusieurs comparaisons possibles entre deux opérateurs. Les deux théorèmes, la construction de l'algèbre libre et la transitivité restent valables dans ce cas. Pour la construction de l'algèbre libre, il suffit de considérer une règle d'introduction pour chaque diagramme $D_i \xleftarrow{l^\alpha_{i,j}} K^\alpha_{i,j} \xrightarrow{r^\alpha_{i,j}} D_j$

$$\frac{\mathbf{u} \in D_i \multimap U \quad \mathbf{v} \in D_j \multimap U \quad p : \mathbf{u} \circ l^\alpha - \mathbf{v} \circ r^\alpha \in K^\alpha_{i,j} \multimap U}{\mathrm{comp}^\alpha(p, \mathbf{u}, \mathbf{v}, i, j) : \Psi_i \mathbf{u} - \Psi_j \mathbf{v} \in U}$$

On peut alors reprendre pas à pas la démonstration qu'il s'agit de la Ψ-algèbre libre. il en est en fait de même pour le théorème de transitivité.

Pour construire l'équivalent constructif, nous définissons $I = \Sigma \times \mathbb{N}$. Chaque symbol f est représenté par les symboles (f,n) dont l'arité est n. L'ordre sur les symboles est $(f,n) \leq (g,m)$ si $f \preceq g$ et $n \leq m$. Tous les opérateurs sont inflationnaires $I = J$, en toutes les variables $\Delta_i = D_i$ pour tout $i \in J$. Pour les comparaison, on considère pour tout $n \leq m$, l'ensemble des injections $I_{n,m}$ de n vers m. Les diagrammes de comparaison entre (f,n) et (g,m) sont alors les diagrammes $n \xleftarrow{1_n} n \xrightarrow{i} m$ pour chaque injection $i : n \to m$. Les constructions annexes se déduisent immédiatement de celles que nous venons de voir. La traduction d'un terme se passe de la manière suivante :

- $\overline{f} = \Psi_{(f,1)}(0)$,
- $\overline{f(t_1, \cdots, t_n)} = \Psi_{(f,n)}(\overline{t_1}, \cdots, \overline{t_n})$

6.5 Des multi-graphes vers les graphes

Si la structure d'ordre constructif est préservée par la construction de la Ψ-algèbre libre, ce n'est pas le cas pour la notion d'ordre en général, ainsi qu'en atteste l'exemple de l'ordinal ω que nous venons de voir.

Il est malgré tout possible d'obtenir un ordre à partir d'une Ψ-algèbre libre $U = F(B)$. Il s'agit premièrement de considérer le multi-graphe squelettique $\hat{U}$. Comme ce graphe est réflexif et transitif, c'est un pré-ordre. On obtient alors un ordre en identifiant tous les éléments x et y tels que $x - y$ et $y - x$. Mais est-ce la Ψ-algèbre d'ordre libre?

6.5.1 Ψ-algèbre sur les graphes

La notion de Ψ-algèbre que nous avons développée jusqu' ici peut être transposée dans l'univers des graphes. Il s'agit même de la partie qui nous guide informellement lorsque nous construisons une algèbre. Il s'agit de vérifier quand même que la construction à du sens.

Nous avons défini à la section 6.1.2, un tenseur et un Hom-foncteur qui sont étroitement liés avec le tenseur et le Hom-foncteur des multi-graphes. Disposant d'une structure monoïdale fermée, nous pouvons

faire de l'algèbre universelle. Mais est-ce que les constructions additionnelles des Ψ-théories ne rentrent pas en conflit avec la structure de graphe squelettique?

Nous supposons que les arités sont des graphes. Comme nous avons fait une hypothèse d'injectivité sur les morphismes $D_\imath \xleftarrow{l_{\imath,\jmath}} K_{\imath,\jmath} \xrightarrow{r_{\imath,\jmath}} D_\jmath$, les graphes $K_{\imath,\jmath}$ sont nécessairement des graphes (et les morphismes $l_{\imath,\jmath}$ et $r_{\imath,\jmath}$ des morphismes de graphes). Nous observons maintenant que la construction $(\bullet \rightarrow_k D_\imath)$ est un graphe, et que la construction $[D_\imath K_{\imath,\jmath} D_\jmath]$ en est un également. De ce fait, toutes les constructions (en dehors de la Ψ-algèbre libre que nous définissons ci-dessous) se font dans les graphes. C'est maintenant ce à quoi nous nous intéressons.

Dans la suite de cette section, nous faisons donc l'hypothèse que pour tout $\imath$, l'arité $D_\imath$ est un graphe. En outre, pour assurer le théorème 6.5.6.1 à venir, nous supposons que pour tous $\imath \prec \jmath$, le morphisme $l_{\imath,\jmath} = 1_{D_\imath}$ et que le morphisme $r_{\imath,\jmath}$ est une injection canonique. Cette hypothèse est utilisée pour montrer le lemme 6.5.4.

Définition 6.5.1 (Ψ-algèbre sur les graphes). Étant donné une Ψ-théorie donnée par le quintuplet suivant : $((I, \prec), (D_\imath)_{\imath \in I}, J, (\Delta_\imath)_{\imath \in J}, (K_{\imath,\jmath})_{\imath \prec \jmath})$, une Ψ-algèbre sur les graphes est un quadruplet

$$(G, (\psi)_{\imath \in I}, (\mathtt{infl})_{\imath \in J, k \in \Delta_\imath}, (\mathtt{comp})_{\imath < \jmath}))$$

où G est un graphe, où pour tout $\imath \in I$, $\Psi_\imath \in \mathbf{Gr}(D_\imath, G)$ est une interprétation du symbole $\Psi_\imath$, et où les familles $(\mathtt{infl})_{\imath \in J, k \in \Delta_\imath}$ et $(\mathtt{comp})_{\imath < \jmath}$ vérifient les équations des multi-graphes transposées dans les graphes.

Proposition 6.5.2. La Ψ-algèbre libre engendrée par le graphe B est définie inductivement par les règles (à comparer avec les règles pour

la Ψ-algèbre de multi-graphe) :

$$\frac{b \in B}{\phi b \in U} \qquad \frac{\mathbf{u} : D_\imath \multimap U}{\psi_\imath \mathbf{u} \in U}$$

$$\frac{b - c \in B}{\phi b - \phi c \in U} \qquad \frac{\mathbf{u} - \mathbf{v} \in D_\imath \multimap U}{\psi_\imath \mathbf{u} - \psi_\imath \mathbf{v} \in U} (\imath \in I)$$

$$\frac{\mathbf{u} : D_\imath \multimap U \quad x - \mathbf{u}_k \in U}{x - \psi_\imath \mathbf{u} \in U} (\imath \in J, k \in \Delta_\imath)$$

$$\frac{\mathbf{u} : D_\imath \multimap U \quad \mathbf{v} : D_\jmath \multimap U \quad \mathbf{u} \circ l - \mathbf{v} \circ r \in K_{\imath,\jmath} \multimap U}{\psi_\imath \mathbf{u} - \psi_\jmath \mathbf{v}} (\imath \prec \jmath)$$

Les définitions sont très proches de celles que nous avons faites pour les multi-graphes ; pour souligner la différence avec les multi-graphes, nous avons adopté des lettres minuscules.

Démonstration. La démonstration est analogue à celle que nous avons faite pour les multi-graphes. Il suffit d'« oublier » le nom des arêtes. On notera tout de même que le multi-graphe $K_{\imath,\jmath} \otimes \mathbf{2}$ qui apparait dans la construction est un graphe squelettique. □

Proposition 6.5.3. Comme dans le cas des multi-graphes, nous pouvons observer que la structure d'ordre constructif est conservée par la construction. Nous procédons pour montrer cela de la même manière que pour les multi-graphes.

Mais en plus de cela, nous avons un théorème supplémentaire :

Théorème 6.5.3.1.
Si l'on suppose que B est un ordre (et non un pré-ordre), l'algèbre libre $U = F(B)$ est un ordre.

La preuve repose sur le lemme crucial suivant.

Lemme 6.5.4. Étant donné un élément $\mathbf{u} \in D_i \multimap U$, il n'y a pas d'arc entre $\psi_i \mathbf{u} - \mathbf{u}_k$.

Démonstration. Le principe de la démonstration est de montrer que s'il existe un arc entre $\psi_i \mathbf{u} - \mathbf{u}_k$, alors $\psi_i \mathbf{u}$ contient une branche infinie. Nous définissons maintenant les branches des éléments de U.

Définition 6.5.5. Soit x un sommet de U, on appelle noyau[32] de x l'ensemble Γ_x des mots sur I défini inductivement :

$$\begin{aligned} \Gamma_{\phi b} &= \{\epsilon\} \\ \Gamma_{\psi_i \mathbf{u}} &= \{kw \mid k \in D_i \text{ et } w \in \Gamma_{\mathbf{u}_k}\} \end{aligned}$$

Proposition 6.5.6.

(i) Pour tout x, $\Gamma_x \neq \emptyset$.

(ii) Supposons que $x - y \in U$. Pour tout mot $w \in \Gamma_x$, il existe un mot $\alpha(w) \in \Gamma_y$ tel que $w \trianglelefteq \alpha(w)$ où $\trianglelefteq$ est l'ordre sous-mot (de Higman) : $w = a_1 a_2 \cdots a_n \trianglelefteq t$ ssi il existe $n+1$ mots éventuellement vides $w_1, \cdots, w_{n+1}$ tels que $t = w_1 a_1 w_2 a_2 \cdots w_n a_n w_{n+1}$.

(iii) Supposons que $x - y \in U$. Pour tous mots w, t_0, t_1 tels que $wt_0 \in \Gamma_x$ et $wt_1 \in \Gamma_x$, les mots $\alpha(wt_0)$ et $\alpha(wt_1)$ ont en commun un préfixe de longueur au moins $|w|$.

$$\begin{array}{lcl} \Gamma_x & & \Gamma_y \\ wt_0 & \trianglelefteq & \alpha(wt_0) \\ wt_1 & \trianglelefteq & \alpha(wt_1) \end{array}$$

Preuve de la proposition.

(i) Par induction sur x.

(ii) Nous procédons par induction sur $x - y$:

- Pour $\dfrac{b - c \in B}{\phi b - \phi c \in U}$. Nous avons $\Gamma_{\phi(b)} = \{\epsilon\}$. Nous notons alors que $\epsilon \trianglelefteq \epsilon \in \Gamma_{\phi c}$. On définit en conséquence $\alpha(\epsilon) = \epsilon$,

32. À rapprocher du noyau des arborescences

- pour $\dfrac{\mathbf{u}-\mathbf{v} : D_i \multimap U}{\psi_i\mathbf{u}-\psi_i\mathbf{v} \in U}$. Soit $w \in \Gamma_{\psi_i\mathbf{u}}$. Du fait de la définition de $\Gamma_{\psi_i\mathbf{u}}$, il existe un $k \in D_i$ tel que $w = kw'$ avec $w' \in \Gamma_{\mathbf{u}_k}$. Il existe donc un mot $\alpha(w') \in \Gamma_{\mathbf{v}_k}$ tel que $w' \trianglelefteq \alpha(w')$. D'où l'on déduit $w \trianglelefteq k\alpha(w')$ avec $k\alpha(w') \in \Gamma_{\psi_i\mathbf{v}}$, on définit en conséquence $\alpha(w) = k\alpha(w')$,

- pour $\dfrac{\mathbf{u} : D_i \multimap U \qquad x-\mathbf{u}_k \in U}{x-\psi_i\mathbf{u} \in U}$. Soit $w \in \Gamma_x$, nous avons par induction un mot $\alpha(w') \in \Gamma_{\mathbf{u}_k}$ tel que $w \trianglelefteq \alpha(w')$. On conclut en notant que $w \trianglelefteq k\alpha(w')$ avec $k\alpha(w') \in \Gamma_{\psi_i\mathbf{u}}$: on définit alors $\alpha(w) = k\alpha(w')$,

- pour $\dfrac{\mathbf{u} : D_i \multimap U \qquad \mathbf{v} : D_j \multimap U \qquad \mathbf{u} \circ l - \mathbf{v} \circ r \in K_{i,j} \multimap U}{\psi_i\mathbf{u}-\psi_j\mathbf{v}}$.
Soit un mot $w \in \Gamma_{\psi_i\mathbf{u}}$. En raison de la définition de $\Gamma_{\psi_i\mathbf{u}}$, il existe un $k \in D_i$ tel que $w = kw'$ avec $w' \in \Gamma_{\mathbf{u}_k}$. Il existe donc un mot $\alpha(w') \in \Gamma_{\mathbf{v}_k}$ tel que $w' \trianglelefteq \alpha(w')$. D'où l'on déduit $w \trianglelefteq k\alpha(w')$ avec $k\alpha(w') \in \Gamma_{\psi_i\mathbf{v}}$. On définit en conséquence : $\alpha(w) = k\alpha(w')$. On s'aperçoit ici de l'importance de l'hypothèse que nous avons faite sur l et r ; en particulier, le fait de supposer que l est surjective nous permet de nous ramener à l'hypothèse d'induction (le fait que l soit l'identité et r une injection canonique nous permet d'identifier les lettres k. Nous faisons la conjecture que la propriété reste vraie du seul fait que l soit surjective).

(iii) Nous procédons également par induction sur $x - y$:

- pour $\dfrac{b-c \in B}{\phi b-\phi c \in U}$. Dans ce cas, la propriété est réalisée (tous les mots se réduisent à ϵ),

- pour $\dfrac{\mathbf{u}-\mathbf{v} : D_i \multimap U}{\psi_i\mathbf{u}-\psi_i\mathbf{v} \in U}$. Supposons que nous ayons des mots t_0, t_1, w qui vérifient les hypothèses de (iii). Il y a deux cas. Supposons que $w = \epsilon$. D'après (ii), $t_1 \trianglelefteq \alpha(t_1)$. Le mot $\alpha(wt_1)$ a un prefixe commun de longueur 0 avec $\alpha(wt_0)$. Supposons maintenant que $w = kw'$, en raison de (ii), $\alpha(wt_0) = k\alpha(w't_0)$. Les mots $w't_0$ et $w't_1$ sont dans $\Gamma_{\mathbf{u}_k}$

et le mot $\alpha(w't_0)$ est dans $\Gamma_{\mathbf{v}_k}$. Par induction, $\alpha(w't_1)$ a un prefixe de longueur $|w'|$ avec $\alpha(w't_0)$. Par conséquent, $\alpha(wt_1)$ et $\alpha(wt_0)$ ont un prefixe commun de longueur $|w|$,

- pour $\dfrac{\mathbf{u} : D_i \multimap U \qquad x - \mathbf{u}_k \in U}{x - \psi_i \mathbf{u} \in U}$. Supposons que nous ayons des mots t_0, t_1, w qui vérifient les hypothèses de (iii). D'après (ii), $\alpha(wt_0) = k\alpha(w't_0)$. Par induction, $\alpha(w't_1)$ a un prefixe de longueur $|w'|$ avec $\alpha(w't_0)$. Par conséquent, $\alpha(wt_1)$ et $\alpha(wt_0)$ ont un prefixe commun de longueur $|w|$,
- pour $\dfrac{\mathbf{u} : D_i \multimap U \qquad \mathbf{v} : D_j \multimap U \qquad \mathbf{u} \circ l - \mathbf{v} \circ r \in K_{i,j} \multimap U}{\psi_i \mathbf{u} - \psi_j \mathbf{v}}$. Ce cas se traite comme le deuxième. Il suffit d'observer que l'on peut se ramener aux hypothèses d'induction en raison des hypothèses sur l et r.

□

Revenons maintenant à la preuve du lemme. Supposons que $\psi_i \mathbf{u} - \mathbf{u}_k$. Nous construisons par induction sur $n \in \mathbb{N}$ une suite de lettres $k_1, \cdots, k_n$ telle qu'il existe un mot t_{n+1} avec $kk_1 \cdots k_n t_{n+1} \in \Gamma_{\psi_i \mathbf{u}}$ et $kk_1 \cdots k_{n-1} t_n \trianglelefteq k_1 \cdots k_{n+1} t_{n+2} \in \Gamma_{\mathbf{u}_k}$. L'existence d'une telle suite de lettres $k_1, \cdots, k_n$ pour tout n est contradictoire avec le fait que toute branche doit être finie. Voici maintenant l'induction.

Pour obtenir la première lettre, à l'aide du point (i) de la proposition précédente, nous choisissons un mot $t_0 \in \Gamma_{\psi_i \mathbf{u}}$. En raison du point (ii), il existe un mot $t_1 \in \Gamma_{\mathbf{u}_k}$ tel que $t_0 \trianglelefteq t_1$. Construisons t_2, d'après (ii), nous avons :

$$\begin{array}{ccc} \Gamma_{\psi_i \mathbf{u}} & & \Gamma_{\mathbf{u}_k} \\ kt_1 & \trianglelefteq & r_2 \\ kr_2 & \trianglelefteq & r_3 \end{array}$$

Comme les mots kt_1 et kr_2 ont un même préfixe, on déduit que $r_2 = k_1 t_2$. La lettre k_1 et le mot t_2 ainsi construits vérifient les propriétés demandées.

Supposons maintenant que nous disposions de la propriété pour tout $m \leq n$. Nous construisons la lettre k_{n+1} et le mot t_{n+2}. Nous avons un mot r_{n+2} tel que :

$$\begin{array}{lcl} \Gamma_{\psi_i \mathbf{u}} & & \Gamma_{\mathbf{u}_k} \\ kk_1k_2\cdots k_{n-1}t_n & \trianglelefteq & k_1\cdots k_nt_{n+1} \\ kk_1k_2\cdots k_{n-1}k_nt_{n+1} & \trianglelefteq & k_1\cdots k_nr_{n+2} \\ kk_1k_2\cdots k_{n-1}k_nr_{n+2} & \trianglelefteq & k_1\cdots k_nr_{n+3} \end{array}$$

Mais en observant les deux dernières lignes, nous obtenons d'après (iii) que r_{n+2} peut s'écrire $r_{n+2} = k_{n+1}t_{n+2}$, ce qui termine l'induction. □

Démonstration. Reprenons maintenant la preuve du théorème. Nous étudions les paires $(x,y) \in U$ telles que $x - y$ et $y - x$. Par induction sur la paire de preuves, montrons que $x = y$:

- supposons $(\dfrac{b-c \in B}{\phi b - \phi c \in U}, \dfrac{c-b \in B}{\phi c - \phi b \in U})$. Comme B est un ordre, $b = c$.
  Si l'une des deux preuves est de la forme $\dfrac{b-c \in B}{\phi b - \phi c \in U}$, nous sommes assurés que l'autre est de la même forme. Nous nous intéressons donc aux paires de preuves qui ne contiennent pas une inclusion universelle,
- supposons $(\dfrac{\mathbf{u}-\mathbf{v}}{\psi_i\mathbf{u}-\psi_i\mathbf{v}}, \dfrac{\mathbf{v}-\mathbf{u}}{\psi_i\mathbf{v}-\psi_i\mathbf{w}})$.
  Pour tout $k \in D_i$, nous avons $\mathbf{u}_k - \mathbf{v}_k$ et $\mathbf{v}_k - \mathbf{u}_k$. D'où, par induction $\mathbf{u}_k = \mathbf{v}_k$ pour tout k. Par conséquent, $\mathbf{u} = \mathbf{v}$,
- supposons $(\dfrac{\mathbf{u}-\mathbf{v}}{\psi_i\mathbf{u}-\psi_i\mathbf{v}}, \dfrac{\psi_i\mathbf{v}-\mathbf{u}_k}{\psi_i\mathbf{v}-\psi_i\mathbf{u}})$. Par transitivité, nous avons $\psi_i\mathbf{u} - \mathbf{u}_k$, ce qui est absurde en raison du lemme,
- de même, on montre en se servant du lemme que les autres combinaisons sont impossibles. Le seul cas intéressant est

$$(\frac{\psi_i\mathbf{u}-\mathbf{v}k}{\psi_i\mathbf{u}-\psi_j\mathbf{v}}, \frac{\mathbf{v}\circ l_{j,i}-\mathbf{u}\circ r_{j,i}}{\psi_j\mathbf{v}-\psi_i\mathbf{w}})$$

  Comme $l_{j,i}$ est surjective, on a $\mathbf{v}_k - \mathbf{u}_{r(k)}$; ceci conduit à $\psi_i\mathbf{u} - \mathbf{u}_{r(k)}$, ce qui n'est pas possible d'après le lemme.

□

Théorème 6.5.6.1.
Étant donné un ordre B, le collapse de la Ψ-algèbre libre sur les multi-graphes est exactement la Ψ-algèbre libre sur les graphes.

Démonstration. D'après le théorème précédent, nous notons que V, la Ψ-algèbre libre de graphe, est un ordre.

Soit K, la plus petite fonction vérifiant :

$$\begin{array}{rccl} K: & U & \to & V \\ & \Phi b & \mapsto & \phi b \\ & \Psi_i \mathbf{u} & \mapsto & \phi(\lambda k.K(\mathbf{u}_k)) \end{array}$$

Nous montrons par récurence que K est totale et que K respecte la monotonie. Par induction sur x, nous montrons :

(1) $K(x)$ est défini,
(2) s'il existe une arête $p : y - x \in U$ alors $K(y)$ est défini et $K(y) - K(x) \in V$.

- Soit $x = \Phi b$. En ce cas, nous avons trivialement (1). Deuxièmement, s'il existe une arête $p : y - x$, nous avons $y = \Phi c$ avec $c - b$. Par conséquent, $K(y)$ est défini et $K(y) - K(x)$.
- Soit $x = \Psi_i \mathbf{u}$. Nous avons par induction $K(\mathbf{u}_k)$ est défini. De plus, si on choisi $k - j \in D_i$, nous avons $\mathbf{u}_k - \mathbf{u}_j \in U$. Par induction, $K(\mathbf{u}_k) - K(\mathbf{u}_j) \in V$. Par conséquent, $K(\Psi_i \mathbf{u})$ est bien défini. Il reste à voir que la fonction est monotone. Trois cas se présentent :

  - Supposons que nous avons une preuve de la forme $\dfrac{p : \mathbf{v} - \mathbf{u}}{\Psi_i p : \Psi_i \mathbf{v} - \Psi_i \mathbf{u}}$. Pour tout $k \in D_i$, nous avons $\mathbf{v}_k - \mathbf{u}_k$. De plus, si $j - k \in D_i$, nous avons par transitivité : $\mathbf{v}_k - \mathbf{v}_j - \mathbf{u}j$. Par conséquent, nous avons $\lambda k.K(\mathbf{v}_k) - \lambda k.K(\mathbf{u}_k)$. Ce qui assure $K(\Psi_i \mathbf{v}) - K(\Psi_i \mathbf{u})$.
  - Dans le cas d'une preuve de la forme

$$\frac{\mathbf{u} : D_i \multimap U \qquad p : y - \mathbf{u}_k \in U}{\mathtt{infl}(p,\mathbf{u},i,k) : y - \Psi_i \mathbf{u} \in U}$$

nous avons par induction $K(y) - K(\mathbf{u}_k)$. De là, nous concluons $K(y) - K(\Psi_i \mathbf{u})$.

- Dans le cas d'une preuve de la forme

$$\frac{\mathbf{u} : D_i \multimap U \qquad \mathbf{v} : D_j \multimap U \qquad p : \mathbf{v} - \mathbf{u} \circ r \in D_j \multimap U}{\mathtt{comp}(p,\mathbf{v},\mathbf{u},j,i) : \Psi_j \mathbf{v} - \Psi_i \mathbf{u}}$$

nous avons (par induction) pour tout $k \in D_j$, $\mathbf{v}_k - \mathbf{u}_k$ et pour tout $k - j \in D_i$, $\mathbf{v}_k - \mathbf{u}j$. Par conséquent, nous avons $\lambda k.\mathbf{v}_k - \lambda k.\mathbf{u}_k \in D_i \to V$, ce qui nous amène à la conclusion souhaitée.

Remarque 6.5.7. K est donc une fonction totale et monotone. Supposons que nous ayons deux éléments de U et deux arêtes tels que $p : x - y$ et $q : y - x$. Par monotonie, $Kx - Ky$ et $Ky - Kx$. Mais comme V est un ordre, nous concluons $Kx = Ky$. Inversement, si pour tous sommets x et y tels que $Kx = Ky$, nous pouvons montrer l'existence de deux arêtes p et q telles que $p : x - y$ et $q : y - x$, alors, nous sommes assurés que le collapse du multi-graphe est exactement le graphe libre V. Le reste de la preuve est donc consacré à cette démonstration.

Par induction sur a, nous montrons que si $a = Kx = Ky$, alors il existe deux arêtes p et q telles que $p : x - y$ et $q : y - x$.

- Si $a = \phi b$, nous avons $x = y = \Phi b$. D'où $\Phi 1_b : x - y$ et $\Phi 1_b : y - x$,
- Si $a = \psi_i \mathbf{u}$. On a $x = \Psi \mathbf{u}'$ et $y = \Psi \mathbf{u}''$. Par induction, pour tout k, nous avons une preuve $p_k : \mathbf{u}'k - \mathbf{u}''k$ et une preuve $q_k : \mathbf{u}'k - \mathbf{u}''k$. Par transitivité, nous pouvons construire pour tout $r : k - j \in D_i$, une arête $p_r : \mathbf{u}'k - \mathbf{u}''j$. Par conséquent, $(\lambda k.p_k, \lambda r : k - j.p_r) : \mathbf{u}' - \mathbf{u}''$. D'où $\Psi_i(\lambda k.p_k, \lambda r : k - j.p_r) : \Psi_i \mathbf{u}' - \Psi_i \mathbf{u}''$. De manière symétrique, nous construisons l'arête entre $\mathbf{u}''$ et $\Psi_i \mathbf{u}'$.

□

6.5.2 Collapse par normalisation

Nous nous intéressons ici au cas des Ψ-algèbres pour lesquelles l'ordre sur I est strict, autrement dit, au cas où l'opération comp ne s'applique pas. Nous supposons également que les ensembles Δ contiennent au plus un élément. La Ψ-algèbre pour l'ordre $\mathbb{W}$ vérifie ces deux hypothèses. Dans ces conditions, nous pouvons montrer qu'il existe un candidat naturel dans les classes d'équivalence du multi-graphe.

Théorème 6.5.7.1.
Soit une Ψ-théorie satisfaisant les deux hypothèses sus-mentionnées. Supposons que B soit un ordre. Nous disposons d'une procédure de normalisation sur les arbres de dérivation qui s'ajuste avec la Ψ-algèbre de graphe (qui est un ordre).

Le reste de la section est consacré à la définition de la normalisation.

Définition 6.5.8. Une arête $P \in U$ est dite régulière si :

- $p = \Phi q$,
- $p = \mathtt{infl}(q,\mathbf{v},\iota,k)$ avec q une arête régulière,
- $p = \Psi_\iota q : \Psi_\iota \mathbf{u} - \Psi_\iota \mathbf{v}$ et
  - pour tout $k \in D_\iota$, q_k^+ est régulière,
  - pour tout $r : k - k' \in D_\iota$, q_r^* est régulière,
  - il n'y a pas d'arête r telle que $r : \Psi_\iota \mathbf{u} - \mathbf{v}_k$, avec $k \in \Delta_\iota$.

Proposition 6.5.9. S'il y a une arête $p : x - y \in U$, alors il y a une unique arête régulière $\widehat{p} : x - y$.

Démonstration. Nous construisons les arêtes régulières par induction sur p :

- $p = \phi q : \Phi b - \Phi b'$. Nous définissons $\widehat{p} = p$ et observons que p est régulière. Supposons qu'il y ait une autre arête régulière $p' : \Phi b - \Phi b'$. En raison de la forme de Φb et de celle de $\Phi b'$, p' est nécessairement de la forme $\Phi q'$. Comme B est un ordre, nous avons $q' = q$.

- $p = \mathtt{infl}(q,\mathbf{v},\iota,k) : x - \Psi_\iota \mathbf{v}$. Nous définisssons $\widehat{p} = \mathtt{infl}(\widehat{q},\mathbf{v},\iota,k)$ qui est régulière. Est-elle unique? Supposons le contraire : il y a une arête $p' : x - \Psi_\iota \mathbf{v}$. Il y a deux cas selon que $x = \Phi b$ ou $x = \Psi_\iota \mathbf{u}$.
  - $x = \Phi b$. En ce cas, p' est nécessairement de la forme $\mathtt{infl}(q',\mathbf{v},\iota,k)$; par induction, nous observons que $q' = \widehat{q}$.
  - $x = \Psi_\iota \mathbf{u}$. Pour être régulière, p' est nécessairement de la forme $\mathtt{infl}(q',\mathbf{v},\iota,k)$, nous utilisons alors le même argument qu'au-dessus.
- $p = \Phi_\iota q : \Psi_\iota \mathbf{u} - \Psi_\iota \mathbf{v}$. Il y a deux cas :
  - Il y a une arête r telle que $r : \Psi_\iota \mathbf{u} - \mathbf{v}_k$.En ce cas, nous définissons $\widehat{p} = \mathtt{infl}(\widehat{r},\mathbf{v},\iota,k)$. S'il y a un autre morphisme $p' : \Psi_\iota \mathbf{u} - \Psi_\iota \mathbf{v}$, celui-ci est nécessairement de la forme $\mathtt{infl}$ ou Ψ_ι. Mais un morphisme de la forme Ψ_ι ne serait pas régulier. Nous avons donc l'unicité par induction.
  - Dans l'autre cas, nous définissons simplement $\widehat{\Psi_\iota q} = \Psi_\iota(\lambda k : D_\iota.\widehat{q_k^+},\lambda r : k - k'.\widehat{q_r^*})$. Pour voir qu'il est déterminé de façon unique, il faut noter qu'une autre arête dont la conclusion est $\Psi_\iota \mathbf{u} - \Psi_\iota \mathbf{v}$ ne peut pas être de la forme $\mathtt{infl}$. En appliquant l'induction, nous obtenons l'unicité.

□

Définition 6.5.10. Un sommet $x \in U$ est dit régulier si :

- $x = \Phi b$,
- $x = \Psi_\iota \mathbf{u}$ et
  - pour tout $k \in D_\iota$, $\mathbf{u}_k$ est régulier,
  - pour tout $p : k - k' \in D_\iota$, $\mathbf{u}_p$ est régulier.

Proposition 6.5.11. De la même manière que nous l'avons fait pour les arêtes, nous construisons des sommets « normalisés ». Pour chaque sommet x, il y a un unique sommet régulier $\overline{x}$ tel qu'il y ait deux arêtes $p_x : x - \overline{x}$ et $q_x : \overline{x} - x$.

Démonstration. Par induction sur la construction de x :

- si $x = \Phi b$. nous prenons $\overline{\Phi b} = \Phi b$. Nous avons alors :
  1- x est régulier, et
  2- $\Phi 1_b : \Phi b - \overline{\Phi b}$,
  3- $\Phi 1_b : \overline{\Phi b} - \Phi b$,
  4- supposons que y est un autre sommet tel que $p' : \Phi b - y$ et $q' : y - \Phi b$. En raison de la forme de Φb, q' est nécessairement de la forme $\Phi q : \Phi c - \Phi b$. Par conséquent, $p' = \Phi p : \Phi b - \Phi c$. Comme B est un ordre, nous obtenons $b = c$.
- $x = \Psi_\iota \mathbf{u}$. Par induction, nous notons que pour tout $k \in D_\iota$ et tout $p : k - k' \in D_\iota$, nous avons : $\overline{\mathbf{u}_k} \xrightarrow{q_{\mathbf{u}_k}} \mathbf{u}_k \xrightarrow{\mathbf{u}_p} \mathbf{u}_{k'} \xrightarrow{p_{\mathbf{u}_{k'}}} \overline{\mathbf{u}_{k'}}$. Il y a donc une (unique) arête régulière :

$$\mathtt{Tr}(q_{\mathbf{u}_k}, \widehat{\mathtt{Tr}(\mathbf{u}_p, p_{\mathbf{u}_{k'}})}) : \overline{\mathbf{u}_k} - \overline{\mathbf{u}_{k'}}$$

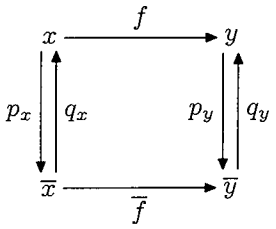
  Nous définissons

$$\overline{\Psi_\iota \mathbf{u}} = \Psi_\iota(\lambda k.\overline{\mathbf{u}_k}, \lambda p : k - k'.\mathtt{Tr}(q_{\mathbf{u}_k}, \widehat{\mathtt{Tr}(\mathbf{u}_p, p_{\mathbf{u}_{k'}})}))$$

  1- x est régulier et
  2- Pour tout $k \in D_\iota$, $p_{\mathbf{u}_k} : \mathbf{u}_k - \overline{\mathbf{u}_k}$ et tout $p : k - k'$: $\mathtt{Tr}(\mathbf{u}_p, p_{\mathbf{u}_{k'}}) : \mathbf{u}_k - \overline{\mathbf{u}_{k'}}$.
  Donc $\Psi_\iota(\lambda k.p_{\mathbf{u}_k}, \lambda p : \widehat{k - k'}.\mathtt{Tr}(\mathbf{u}_p, p_{\mathbf{u}_{k'}})) : \Psi_\iota \mathbf{u} - \overline{\Psi_\iota \mathbf{u}}$
  3- De la même manière, nous obtenons :
  $\Psi_\iota(\lambda k.q_{\mathbf{u}_k}, \lambda p : \widehat{k - k'}.\mathtt{Tr}(q_{\mathbf{u}_k}, \mathbf{u}_p)) : \overline{\Psi_\iota \mathbf{u}} - \Psi_\iota \mathbf{u}$
  4- En ce qui concerne l'unicité, nous étudions les quatre cas dépendants de la forme de la paire $p : \Psi_\iota \mathbf{u} - y$, $q : y - \Psi_\iota \mathbf{u}$. Si p et q sont de la forme Ψ_ι, nous utilisons l'induction. Les autres cas sont traités par le lemme 6.5.4.

□

$\overline{(-)}$ peut-être étendu en un morphisme de multi-graphes. Supposons que $f : x - y$, nous définissons $\overline{f}$ comme étant l'unique arête régulière entre $\overline{x} - \overline{y}$.



Proposition 6.5.12. $\overline{(-)}$ est une procédure de normalisation. C'est-à-dire : $\overline{\overline{(-)}} = \overline{(-)}$. De plus, on peut observer que les sommets et les arêtes régulières constituent un ordre qui est en correspondance bijective qvec la Ψ-algèbre des ordres. Par conséquent, nous avons le théorème 6.5.7.1.

Démonstration. Par induction. □

Chapitre 7

Un λ-calcul monotone

Poursuivons notre étude des termes ordinaux, et plus généralement des Ψ-algèbres libres. Le présent chapitre conclut d'une certaine manière les deux précédents en donnant une syntaxe (un λ-calcul) qui permet de manipuler les ordres, tout du moins les ordres qui sont obtenus comme des Ψ-algèbres libres. Par manipuler, nous entendons premièrement que l'on peut montrer à l'aide du calcul des inégalités (monotonie de l'addition, de la multiplication, ...) et deuxièmement que l'on bénéficie du principe d'induction de la Ψ-algèbre libre (pour définir les hiérarchies de fonctions sous-récursives par exemple). Nous nous concentrons sur le cas des arborescences croissantes $\mathbb{W}$ (cf la section 5.4 et la section 6.4) qui nous semble suffisament révélateur du procédé de construction. Comme nous l'avons justifié dans l'introduction du chapitre 6, nous nous plaçons dans le cadre de la théorie des types dépendants.

De manière générale, nous avons cherché à construire un système qui met en jeu des constructions les plus élémentaires possibles, l'objectif étant que le cadre de travail n'interfère pas avec les schémas d'inductions ou qu'il les contienne. Le paradigme que nous avons suivi dans cette optique est le système T de Gödel. En particulier, nous avons évité un codage imprédicatif des schémas d'inductions ; ceux-ci sont introduits de manière explicite avec leurs règles d'évaluation. D'autre part, comme nous ne considérons qu'une seule sorte de pré-

dicats ($p \in R_X(x,y)$), nous n'avons pas besoin d'univers. En dehors des constructions standard (c'est-à-dire du type Σ et du type Π), le seul aspect des types dépendants que nous utilisons est le nommage des arêtes de graphes. En particulier, les constructeurs de types sont de la forme $x_1 : X_1, \ldots, x_n : X_n \vdash S(x_1, \ldots, x_n)$ où $(X_i)_{i \leq n}$ sont des types définis au préalable et qui ne dépendent que des variables qui les précèdent dans le contexte.

Pour construire un calcul qui rend compte de l'induction sur $\mathbb{W}$, nous devons représenter les ω-chaînes de $\mathbb{W}$, en d'autres termes, les morphismes $\omega \to \mathbb{W}$. Pour cela, nous utilisons le principe d'induction de ω. Nous construisons donc tout d'abord un calcul (nommé λ-ω) qui permet d'utiliser ω comme une arité. Plus techniquement, le « type » $\boldsymbol{\omega}$ associé à ω est la donnée d'un type pour $\mathbb{N}$, d'un type pour la relation d'ordre ω sur $\mathbb{N}$ et d'un terme de réflexivité. Le calcul fournit en plus le principe d'induction correspondant à ces deux types via une règle d'élimination du type $\boldsymbol{\omega}$. Notons enfin que du fait qu'il inclut le principe d'induction de $\mathbb{N}$, λ-ω est une extension du système T de Gödel.

Une fois défini λ-ω, nous sommes prêts à introduire le calcul λ-$\mathbb{W}$ qui traite de l'ordre $\mathbb{W}$ des arborescences croissantes. Une application de ce calcul est la construction des hiérarchies de fonctions sous-récursives. Nous montrerons à l'aide du système quelques propriétés de ces dernières comme la monotonie des fonctions de la hiérarchie.

La différence essentielle entre la construction de λ-$\mathbb{W}$ et celle de λ-ω vient du fait que dans le cas de $\mathbb{W}$, nous avons un opérateur $\mathbf{L}$:

- d'arité non finie,
- d'arité un graphe *non discret.*

En ce qui concerne le premier point, la solution consiste à construire l'arité à un premier niveau, c'est le rôle de λ-ω. La conséquence du deuxième point est qu'il n'y a pas de séparation claire entre les sommets d'une part et les arêtes de l'autre comme pour ω mais une définition mutuellement dépendante des sommets et des arêtes ; ceci complique le principe d'induction, et par conséquent le schéma d'élimination du type $\mathbb{W}$. En revanche, la construction de λ-$\mathbb{W}$ peut être utilisée comme paradigme pour le cas général.

Venons-en à la syntaxe plus en détail. Nous considérons trois uni-

vers sémantiques, celui des ensembles (**Ens**, $\times$, $\Rightarrow$), celui de la SMC (**MGr**, $\otimes$,$\multimap$) dans lequel nous avons défini la notion de Ψ-algèbre et enfin la structure (**Refl**, $\times$,$\multimap$) qui nous donne une syntaxe bénéficiant de l'affaiblissement, de la contraction et des types dépendants (cf la section 6.1.3). Un des premiers objectifs de notre syntaxe est de pouvoir faire coexister ces trois univers au sein d'un même système. Pour cela, nous choisissons comme univers de base (de notre sémantique, mais aussi de notre syntaxe) les ensembles. Les calculs correspondant aux autres univers sémantiques sont alors codés dans la syntaxe de base. Nous prenons soin de faire un codage qui soit « correct ». Ceci nous permet de manipuler les graphes comme s'ils constituaient des types à part entière.

La représentation de la structure (**MGr**, $\otimes$,$\multimap$) dans la syntaxe n'est pas très intéressante, en effet, du fait qu'il ne bénéficie pas de principe d'affaiblissement, le calcul est lourd a employer (voir la discussion de la section6.1.3) ; en outre, nous n'avons pas les types dépendants (à cause de la linéarité du calcul, voir l'introduction du chapitre 6). Nous utiliserons plutôt le calcul fondé sur la structure (**Refl**, $\times$,$\multimap$) qui est plus pratique. Nous pouvons dans ce calcul définir le principe d'induction de la même manière que ce que nous ferions dans le calcul (**MGr**, $\otimes$,$\multimap$) à ceci près que nous devons introduire un terme de réflexivité « bidon », alors que celui-ci peut être défini au seul moyen du principe d'induction.

Nota bene 7.0.13. Dans ce chapitre, nous éludons du terme multigraphe la préposition « multi », en effet, en termes de syntaxe, les graphes sont codés comme des multi-graphes.

7.1 Conventions de notation pour les graphes réflexifs

Du point de vue sémantique, les types que nous utilisons sont des « ensembles », or, nous travaillons essentiellement avec des graphes réflexifs. Si on code de manière directe les schémas d'induction en termes d'ensembles, l'expérience a montré que le calcul devient très lourd. Pour combler ce hiatus entre graphe et ensemble, nous utilisons des abréviations de notation qui nous permettent de travailler

comme si les graphes réflexifs étaient des types. Nous adoptons ainsi des conventions pour chaque opération qui correspond moralement au λ-calcul des graphes réflexifs : l'introduction d'un graphe réflexif, d'un morphisme de graphe, etc. Nous introduisons de nouveaux jugements comme (nous expliquons ce que ces jugements recouvrent plus loin) :

$$\begin{aligned} \mathbf{\Gamma} &\vdash \mathbf{X}\ \mathtt{grfR} \\ \mathbf{\Gamma} &\vdash \mathbf{t} : \mathbf{X} \end{aligned}$$

Le jugement $\mathbf{\Gamma} \vdash \mathbf{X}\ \mathtt{grfR}$ définit le graphe réflexif $\mathbf{X}$. Il désigne en fait deux jugements de type usuels et un jugement de terme usuel ; le premier jugement de type correspond aux sommets du graphe réflexif $\mathbf{X}$, le deuxième aux arêtes du graphe réflexif, le terme correspond à la réflexivité. Le deuxième jugement dérivé, $\mathbf{\Gamma} \vdash \mathbf{t} : \mathbf{X}$, introduit un morphisme de graphes ou une arête selon le contexte. Par la suite, nous qualifions $\mathbf{t}$ de *terme de graphe*. Nous notons en gras les « objets » du calcul correspondant à **Refl**.

Enfin, il arrive qu'un graphe réflexif dépende d'un ou plusieurs autres graphes réflexifs. Pour exprimer cette dépendance, nous utilisons des *variables de graphe* et des *contextes de graphe* (notées en gras).

En résumé, nous faisons coexister deux calculs, un calcul ordinaire noté en « fin », le calcul des « ensembles » et un calcul « gras », le calcul des « graphes », le second étant un système d'abréviations dans le premier. Le système gras factorise des ensembles de jugements « fins », de termes « fins », de variables « fines », etc. Nous pourrions donc écrire tout le système dans le calcul fin, mais au prix d'une complication (apparente, et qui ne révèle pas la structure des constructions) des schémas d'induction.

Pour passer d'un calcul à l'autre, nous utilisons un système d'indices. Celui-ci est fondé sur le fait que dans les abréviations qui suivent, il y a généralement les trois parties que nous avons vues pour le jugement « $\mathbf{\Gamma} \vdash \mathbf{X}\ \mathtt{grfR}$ » : la première composante correspond aux sommets du graphe réflexif, la deuxième aux arêtes et la troisième à la réflexivité. Il s'avère en fait plus élégant de dupliquer le premier jugement : la première copie correspond aux sommets vus comme des

domaines et la deuxième aux sommets vus comme des codomaines. Comme on utilise des variables différentes pour les domaines et les codomaines, la deuxième copie s'obtient en fait par remplacement dans la première copie des variables de « domaines » par les variables de « codomaine » correspondantes. Une deuxième justification de la duplication des sommets est que le mécanisme de substitution dans les termes $\mathbf{X}-\mathtt{grfR}$ et dans les termes $\mathtt{t}$ de graphe réflexif est unifié.

Pour que la notation traduise de manière explicite cette manière de voir, nous utilisons quatre indices 0, 1, 2 et $\mathtt{r}$ (notés comme des exposants !) ; l'indice 0 correspond aux sommets vus comme des domaines, l'indice 1 aux sommets vus comme des codomaines, l'indice 2 aux arêtes et l'indice $\mathtt{r}$ pour la réflexivité. Ceci nous permet de considérer un jugement « fin » comme une composante d'un jugement « gras ».

Pour illustrer notre propos, nous utiliserons le graphe $\boldsymbol{\omega}$. Mais notons que ce graphe restera une enveloppe vide (en ce sens que nous ne donnons pas le principe d'induction) jusqu'à la section 7.2 où nous donnerons l'ensemble des règles pour le graphe ω.

7.1.1 Graphe réflexif et contexte de graphe

Le principe général de la construction du calcul des graphes réflexifs est le suivant[33]. On définit d'abord les sommets du graphe réflexif, les arêtes sont définies ensuite comme dépendant des sommets. Deuxièmement, dans le cas où le graphe réflexif dépend d'un autre graphe réflexif, les sommets ne *dépendent que des sommets précédemment définis*, les arêtes (dépendant des autres arêtes), elles, *dépendent de toutes les variables.*

Note 7.1.1. Nous supposons disposer d'un nombre dénombrable de lettres $x, y, \ldots$, ceci nous donne cinq réservoirs distincts et « parallèles » de variables, les variables « nues » (ou d'ensemble) : x, y, ... ; les variables de type 0 (ou de domaine) : x^0, y^0, ... ; les variables de type 1 (ou de codomaine) : x^1, y^1, ... ; les variables de type 2 (ou d'arête) : x^2, y^2, ... ; enfin, les variables de graphe réflexif : $\mathbf{x}$, $\mathbf{y}$, ...

33. Celui-ci est obtenu en dévissant la sémantique des types dépendants dans la catégorie des graphes réflexifs.

Les variables ne sont pas typées d'avance, mais elles ont un usage défini a priori. Une variable de domaine désigne un domaine, une variable de graphe réflexif désigne un graphe, etc.

Définition 7.1.1 (Contextes et graphes réflexifs). Les contextes $\mathbf{\Gamma}$ de graphe réflexif sont obtenus inductivement à l'aide des trois règles :

$$\frac{}{\Diamond \ \texttt{ctx-grfR}} \qquad \frac{\mathbf{\Gamma} \vdash \mathbf{X} \ \texttt{grfR}}{\mathbf{\Gamma}, \mathbf{x} : \mathbf{X} \ \texttt{ctx-grfR}}$$

Un contexte de graphe réflexif n'est qu'une notation qui recouvre trois contextes (usuels) : les contextes Γ^0 et Γ^1 des sommets, et le contexte Γ^2 des arêtes.

- Le contexte Γ^0 est le contexte $\mathbf{\Gamma}$ dans lequel on a remplacé toutes les variables $\mathbf{x} : \mathbf{X}$ par $x^0 : X^0$;
- Le contexte Γ^1 est le contexte $\mathbf{\Gamma}$ dans lequel on a remplacé toutes les variables $\mathbf{x} : \mathbf{X}$ par $x^1 : X^1$;
- Le contexte Γ^2 est le contexte $\mathbf{\Gamma}$ dans lequel on a remplacé toutes les variables $\mathbf{x} : \mathbf{X}$ par $x^0 : X^0, x^1 : X^1, x^2 : X^2$.

Les règles de construction des graphes réflexifs assurent que ces trois contextes issus de $\mathbf{\Gamma}$ sont correctement définis. Nous y reviendrons à la remarque 7.1.3.

Note 7.1.2. Nous utilisons une double barre d'inférence pour indiquer de manière explicite l'utilisation d'une convention.

Note 7.1.3. On peut étendre les indices aux graphes (définis ci-dessous et plus tard aux termes), mais dans ce cas, les indices 0 et 1 n'ont pas la même fonction. Les variables x^0 et x^1 sont des variables distinctes. En revanche, pour les types (et plus tard les termes), les choses sont différentes. Lorsqu'on emploie les indices 0 et 1 dans un type, X^0 et X^1 ne diffèrent que par des substitutions de variables, et il en sera de même pour les termes.

Étant donné un contexte de graphe réflexif $\mathbf{\Gamma}$, un graphe réflexif est la donnée de trois jugements :

$$\frac{\begin{array}{rcl}\Gamma^0 & \vdash & Y^0 \text{ type} \\ \Gamma^2, y^0 : Y^0, y^1 : Y^1 & \vdash & Y^2(y^0, y^1) \text{ type} \\ \Gamma^{\mathrm{r}}, y^0 : Y^0 & \vdash & \mathtt{Ref}_{\mathbf{Y}} : Y^{\mathrm{r}}(y^0, y^0)\end{array}}{\Gamma \vdash \mathbf{Y} : \mathtt{grfR}}$$

avec les *restrictions suivantes* :

- Y^1 est le type Y^0 dans lequel on a substitué toutes les variables de domaine par les variables de codomaine correspondantes : $Y^1 = Y^0(x_1^0 \backslash x_1^1, \cdots, x_n^0 \backslash x_n^1)$ où $\mathbf{x}_1, \ldots, \mathbf{x}_n$ sont les variables de graphe du contexte $\mathbf{\Gamma}$. Nous reprenons l'écriture que nous avions dans le chapitre 1 pour les substitutions multiples : nous notons $Y^0(\vec{x}^0 \backslash \vec{x}^1)$ le terme $Y^0(x_1^0 \backslash x_1^1, \cdots, x_n^0 \backslash x_n^1)$.
- Le type $Y^{\mathrm{r}}(y^0, y^1)$ est le type Y^2 dans lequel on a substitué chaque variable de type 1 du contexte $\mathbf{\Gamma}$ par la variable de type 0 correspondante et chaque variable de type 2 du contexte $\mathbf{\Gamma}$, par le terme de réflexivité du type correspondant. En d'autres termes, nous avons l'égalité

$$Y^{\mathrm{r}}(y^0, y^1) = Y^2[x_1^1 \backslash x_1^0, x_1^2 \backslash \mathtt{Ref}_{\mathbf{x}_1}(x_1^0), \ldots, x_n^1 \backslash x_n^0, x_n^2 \backslash \mathtt{Ref}_{\mathbf{x}_n}(x_n^0)]$$

où $\mathbf{x}_1, \ldots, \mathbf{x}_n$ sont les variables de graphe du contexte $\mathbf{\Gamma}$.

$\mathbf{Y}$ désigne alors le quadruplet $(Y^0, Y^1, Y^2, \mathtt{Ref}_{\mathbf{Y}})$.

Inversement, si l'on a un graphe réflexif $\mathbf{Y}$, on note Y^i la composante i du triplet et $\mathtt{Ref}_{\mathbf{Y}}$ désigne le terme de réflexivité.

Remarque 7.1.2. Dans la conclusion de l'introduction du graphe réflexif, nous n'avons pas noté les deux variables y^0 et y^1 dans le contexte. Néanmoins, elles font partie du contexte pour l'introduction des arêtes. Nous avons en fait une écriture pour deux contextes différents, mais il n'y a pas de risques de confusion.

Remarque 7.1.3. L'ajout d'une variable dans un contexte recouvre donc plusieurs règles de construction de contexte usuel. Étant donné un jugement $\mathbf{\Gamma} \vdash \mathbf{X}$ grfR, la règle de formation de contexte :

$$\frac{\mathbf{\Gamma} \vdash \mathbf{X} \text{ grfR}}{\mathbf{\Gamma}, \mathbf{x} : \mathbf{X} \text{ ctx-grfR}}$$

représente la formation de trois contextes usuels :

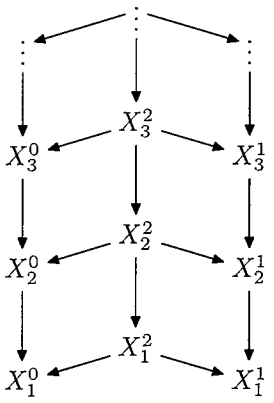


FIG. 7.1 – *Dépendance des types dans le contexte* $\mathbf{x_1} : \mathbf{X_1}, \mathbf{x_2} : \mathbf{X_2}, \cdots, \mathbf{x_n} : \mathbf{X_n}$

$$\frac{\Gamma^0 \vdash X^0 \text{ type}}{\Gamma^0, x^0 : X^0 \text{ ctx}} \qquad \frac{\Gamma^1 \vdash X^1 \text{ type}}{\Gamma^1, x^1 : X^1 \text{ ctx}}$$

$$\frac{\Gamma^2, x^0 : X^0, x^1 : X^1 \vdash X^2(x^0, x^1) \text{ type}}{\Gamma^2, x^0 : X^0, x^1 : X^1, x^2 : X^2(x^0, x^1) \text{ ctx}}$$

Exemple 7.1.4. Pour définir le graphe ω, nous introduisons un « type atomique »

$$\frac{}{\vdash \omega \ \texttt{grfR}}$$

Celui-ci recouvre les trois règles suivantes :

$$\frac{}{\vdash \omega^0 \text{ type}} \qquad \frac{}{n^0 : \omega^0, n^1 : \omega^1 \vdash \omega^2(n^0, n^1) \text{ type}}$$

$$\frac{\Gamma \vdash t : \omega}{\Gamma \vdash \mathtt{Ref}_\omega(t) : \omega^2(t,t)}$$

Notons que les indices 0 et 1 sur ω ne sont que des décorations qui rendent symétrique la notation, ce sont les indices 0 et 1 sur les variables qui indiquent la qualité de domaine ou de codomaine de la variable.

Exemple 7.1.5. Pour voir la dépendance des types vis-à-vis des variables dans les graphes réflexifs, voyons sur un exemple le cas d'un graphe réflexif dépendant d'un (unique) graphe réflexif.

On suppose que $\vdash \mathbf{X}$ `grfR`. Un graphe réflexif $\mathbf{x} : \mathbf{X} \vdash \mathbf{Y}$ `grfR` est la donnée de trois jugements :

$$\begin{aligned} x^0 : X^0 &\vdash Y^0(x^0) \text{ type} \\ x^0 : X^0, x^1 : X^1, x^2 : X^2(x^0,x^1), & \\ y^0 : Y^0(x^0), y^1 : Y^1(x^1) &\vdash Y^2(x^0,x^1,x^2,y^0,y^1) \text{ type} \\ x^0 : X^0, y^0 : Y^0 &\vdash \mathtt{Ref_Y} : Y^2(x^0,x^0,\mathtt{Ref_X}(x^0),y^0,y^0) \end{aligned}$$

où comme nous l'avons dit, X^1 et $Y^1(x)$ sont syntaxiquement égaux à X^0 et $Y^0(x)$.

En termes catégoriques, cet ensemble de jugement est la traduction *exacte* d'un morphisme de graphe réflexif dont toutes les composantes sont des display maps.

7.1.2 Abréviations pour les termes de graphe

Nous nous intéressons maintenant aux termes, pour lesquels nous avons une convention inspirée de ce que nous venons de présenter. Nous commençons par décrire ce que recouvre un « terme » de graphe dans le calcul fin. Par la suite, nous présentons les règles de jugement

de termes de graphe. Un terme de graphe réflexif est la donnée de deux termes :

$$\frac{\begin{array}{lll}\Gamma^0 & \vdash & t^0 : Y^0 \\ \Gamma^2 & \vdash & t^2 : Y^2(t^0,t^1)\end{array}}{\begin{array}{lll}\mathbf{\Gamma} & \vdash & \mathbf{t} : \mathbf{Y}\end{array}}$$

avec la contrainte que t^1 s'obtient par substitution dans t^0 des variables de domaine par les variables de codomaine :

$$t^1 = t^0(x_1^0 \backslash x_1^1, \cdots, x_n^0 \backslash x_n^1)$$

Le fait que l'on puisse dériver $\Gamma^0 \vdash t^0 : X^0$ implique que l'on peut dériver $\Gamma^1 \vdash t^1 : X^1$ par substitution des variables de domaine par leur variables de codomaine correspondantes.

En d'autres termes, $\mathbf{t}$ est la donnée d'un triplet (t^0,t^1,t^2) qui vérifie la contrainte qui précède. Notons qu'il n'y a pas de terme t^r : les morphismes de graphes dans **Refl** ne dépendent pas des fonctions de réflexivité (voir la section 6.1.3) !

Remarque 7.1.6. Ici, contrairement à ce que nous avions fait pour l'introduction des types (cf. remarque 7.1.2), $\mathbf{\Gamma}$ dans la conclusion est une abréviation orthodoxe : les variables y^0 et y^1 sont substituées par les termes t^0 et t^1 dans le deuxième jugement.

Exemple 7.1.7 (Affaiblissement, axiome). Étant donné un graphe réflexif $\mathbf{\Gamma} \vdash \mathbf{X}$ `grfR`. On peut faire les trois jugements suivants :

$$\Gamma^0,x^0 : X^0 \vdash x^0 : X^0 \qquad \Gamma^1,x^1 : X^1 \vdash x^1 : X^1$$
$$\Gamma^2,x^0 : X^0,x^1 : X^1,x^2 : X^2(x^0,x^1) \vdash x^2 : X^2(x^0,x^1)$$

ce qui peut se compacter en :

$$\mathbf{\Gamma},\mathbf{x} : \mathbf{X} \vdash \mathbf{x} : \mathbf{X}$$

Le terme $\mathbf{x}$ représente le triplet de termes (x^0,x^1,x^2). Nous retrouvons ainsi la règle d'affaiblissement :

$$\frac{\Gamma \vdash \mathbf{X}\ \mathtt{grfR}}{\Gamma,\mathbf{x} : \mathbf{X} \vdash \mathbf{x} : \mathbf{X}}$$

Règles de construction pour les symboles

Nous définissons deux sortes de termes, les termes de récursion et les opérateurs ; pour ces deux sortes de termes, nous utilisons un mécanisme de règle d'introduction ordinaire.

Définition 7.1.8. Parmi les opérateurs, on distingue les opérations sur les graphes réflexifs et les opérations d'ensembles. Par exemple, les opérations monotones sont des opérations de graphes, tandis que l'inflationnarité est une opération d'ensemble. Toutes ces opérations peuvent être décrites sous forme de règles d'introduction de symboles fonctionnels dans le calcul fin, mais pour nous simplifier la tâche, nous nous autorisons des règles de la forme :

$$\frac{\Gamma \vdash \mathbf{t} : \mathbf{X}}{\Gamma \vdash \mathbf{p(t)} : \mathbf{Y}}$$

$\mathbf{p}$ s'interprète comme une opération $\mathbf{X} \to \mathbf{Y}$. Cette règle est une écriture pour deux règles :

$$\frac{\Gamma^0 \vdash t^0 : X^0}{\Gamma^0 \vdash p^0(t^0) : Y^0} \qquad \frac{\Gamma^2 \vdash t^2 : X^2(t^0,t^1)}{\Gamma^2 \vdash p^2(t^2) : X^2(p^0(t^0),p^1(t^1))}$$

où p^1 est le symbole p^0, mais que l'on note p^1 pour la symétrie de la notation.

Exemple 7.1.9. Dans le cas de $\boldsymbol{\omega}$, le successeur s'introduit par la règle :

$$\frac{\Gamma \vdash \mathbf{t} : \boldsymbol{\omega}}{\Gamma \vdash \mathbf{s(t)} : \boldsymbol{\omega}}$$

qui résume les deux règles :

$$\frac{\Gamma^0 \vdash t^0 : \omega}{\Gamma^0 \vdash s^0 t^0 : \omega} \qquad \frac{\Gamma^2 \vdash t^2 : \omega^2(t^0,t^1)}{\Gamma^2 \vdash s^2 t^2 : \omega^2(s^0 t^0,s^0 t^1)}$$

Nota bene 7.1.10. Pour disposer d'une écriture des jugements plus uniforme, nous utilisons les conventions de notations suivantes.

Dans un jugement de type 0, c'est-à-dire lorsque l'on introduit le type X^0 d'un graphe réflexif, l'écriture d'un contexte $\mathbf{\Gamma}$ remplace l'écriture Γ^0. Nous avons une convention analogue pour les jugements de type 1, 2 ou $\mathtt{r}$.

Dans un type Y^0, l'écriture $Y^0(\mathbf{x})$ remplace $Y^0(x^0)$, de la même manière, $Y^1(\mathbf{x})$ remplace $Y^1(x^1)$. En revanche, la notation $Y^2(\mathbf{x})$ vaut pour $Y^2(x^0,x^1,x^2)$. Il en est de même pour les termes. $t^0(\mathbf{x})$ désigne le terme $t^0(x^0)$, $t^1(\mathbf{x})$ désigne $t^1(x^1)$ et $t^2(\mathbf{x})$ désigne le terme $t^2(x^0,x^1,x^2)$. En fait, de manière plus générale, nous pouvons faire des substitutions étant donné un terme de graphe $\mathbf{t}$. $r^0(\mathbf{t})$ vaut alors pour $r^0(t^0)$, il en est de même pour les autres types de jugements.

Le principe est donc le suivant : dans un contexte, un type ou un terme de type 0 (resp 1) et « dépendant d'une (ou plusieurs) variable de graphe réflexif », on ne développe que les variables de type 0 (resp 1) ; pour les types ou les termes d'indice 2, l'expansion porte sur toutes les variables.

À l'aide de ces conventions, le jugement de l'exemple 7.1.5 s'écrit :

$$\begin{aligned} \mathbf{x} : \mathbf{X} &\vdash Y^0(\mathbf{x}) \text{ type} \\ \mathbf{x} : \mathbf{X}, y^0 : Y^0(\mathbf{x}), y^1 : Y^1(\mathbf{x}) &\vdash Y^2(\mathbf{x},y^0,y^1) \text{ type} \\ \mathbf{x} : \mathbf{X}, \mathbf{y} : \mathbf{Y} &\vdash \mathtt{Ref}_{\mathbf{Y}} : Y^2(x^0,x^0,\mathtt{Ref}_{\mathbf{X}}(x^0),y^0,y^0) \end{aligned}$$

Application et abstraction sur les graphes réflexifs

Notation 7.1.1. Nous introduisons ici une notation pour les arêtes. Supposons que $\mathbf{X}$ soit un graphe réflexif et Y un type (dépendant éventuellement de $\mathbf{X}$).

$$\Pi_{x^2 : x^0 - x^1 : \mathbf{X}}.Y \stackrel{\text{def}}{=} \Pi_{x^0 : X^0}.\Pi_{x^1 : X^1}.\Pi_{x^2 : X^2(x^0,x^1)}.Y$$

Définition 7.1.11 (Graphe type produit dépendant). La règle :

$$\frac{\Gamma \vdash \mathbf{Y} : \mathtt{grfR} \qquad \Gamma,\mathbf{y} : \mathbf{Y} \vdash \mathbf{Z} : \mathtt{grfR}}{\Gamma \vdash \mathbf{\Pi y} : \mathbf{Y}.\mathbf{Z} : \mathtt{grfR}}$$

est une notation pour la construction suivante :

$$\begin{array}{rcl}
\Gamma & \vdash & \Sigma_{f:\Pi_{y^0:Y^0}.Z^0}.\Pi_{y^2:y^0-y^1:\mathbf{Y}}. \\
 & & Z^2(\vec{x}^1\backslash\vec{x}^0,\vec{x}^2\backslash\vec{\mathtt{Ref}}_{\mathbf{X}}(x^0),y^0,y^1,y^2,fy^0,fy^1) \text{ type} \\
\Gamma,a^0{:}(\mathbf{\Pi y}{:}\mathbf{Y}.\mathbf{Z})^0, & & \\
\quad a^1{:}(\mathbf{\Pi y}{:}\mathbf{Y}.\mathbf{Z})^1 & \vdash & \Pi_{y:Y^0}.Z^2((\pi a^0)y,(\pi a^1)y)\times \\
 & & \Pi_{y^2:y^0-y^1:\mathbf{Y}}.Z^2((\pi a^0)y^0,(\pi a^1)y^1) \text{ type} \\
\Gamma,a^0{:}(\mathbf{\Pi y}{:}\mathbf{Y}.\mathbf{Z})^0 & \vdash & \left\langle \lambda y^0 : Y^0.\mathtt{Ref}_{\mathbf{Z}}(\pi a^0 y^0),\right. \\
 & & \left.\lambda y^0.\lambda y^1.\lambda y^2.(\pi' a^0)y^0y^1y^2 \right\rangle : (\mathbf{\Pi y} : \mathbf{Y}.\mathbf{Z})^2(a^0,a^0) \\
\hline
\Gamma & \vdash & \mathbf{\Pi y} : \mathbf{Y}.\mathbf{Z} : \mathtt{grfR}
\end{array}$$

Intuitivement, un sommet de $\mathbf{\Pi y} : \mathbf{Y}.\mathbf{Z}$ est la donnée d'une fonction $f : Y^0 \to Z^0$ et d'une arête $Z^2(fx,fy)$ pour toute arête $p : x - y \in \mathbf{Y}$. De même, pour les arêtes de $\mathbf{\Pi y} : \mathbf{Y}.\mathbf{Z}$, nous traduisons la définition de l'exponentielle de la SMC à l'aide des types Π (pour rendre compte des fonctions) et Σ : la première partie du type $\Pi_{y:Y^0}.Z^2((\pi a^0)y,(\pi a^1)y)\times\Pi_{y^2:y^0-y^1:\mathbf{Y}}.Z^2((\pi a^0)y^0,(\pi a^1)y^1)$ correspond à la fonction p^+ de la définition 6.1.7 et la deuxième partie à la fonction $p^\star$. Enfin, la réflexivité est directement inspirée de la sémantique.

Notation 7.1.2. Étant donné un graphe $\mathbf{Y}$, supposons que l'on dispose d'un jugement $\mathbf{\Gamma,x} : \mathbf{X} \vdash \mathbf{Y}$ grfR. Si $\mathbf{Y}$ ne dépend pas des variables x^0, x^1 et x^2, on note $\mathbf{X} \multimap \mathbf{Y}$ le type $\mathbf{\Pi x} : \mathbf{X}.\mathbf{Y}$. Par la suite, nous nous servons du graphe réflexif $\mathbf{X} \multimap \mathbf{Y}$ pour introduire des opérations d'arité le graphe réflexif $\mathbf{X}$.

Il y a un cas de construction de graphe bien particulier au schéma d'induction pour lequel nous utilisons une notation spécifique. Supposons que nous disposions des jugements $\mathbf{\Gamma,y} : \mathbf{Y} \vdash \mathbf{Z}$ grfR, et du

jugement $\Gamma \vdash \mathbf{X}$ grfR. Nous définissons le graphe $\mathbf{X} \multimap \mathbf{Z}\{\mathbf{u}\} = \mathbf{\Pi x} : \mathbf{X}.\mathbf{Z}((\mathbf{u})\mathbf{x})$ dans le contexte: $\Gamma,\mathbf{u} : \mathbf{X} \multimap \mathbf{Y} \vdash \mathbf{X} \multimap \mathbf{Z}\{\mathbf{u}\}$ grfR.

Définition 7.1.12 (Abstraction sur les graphes réflexifs). La règle d'abstraction:

$$\frac{\Gamma,\mathbf{y} : \mathbf{Y} \vdash \mathbf{t} : \mathbf{Z}}{\Gamma \vdash \lambda\mathbf{y}.\mathbf{t} : \mathbf{\Pi y} : \mathbf{Y}.\mathbf{Z}}$$

est une notation pour la construction suivante:

$$\Gamma \vdash \left\langle \lambda y^0.t^0,\ \lambda y^0.\lambda y^1.\lambda y^2.t^2[\vec{x}^1\backslash\vec{x}^0,\vec{x}^2\backslash\vec{\mathtt{Ref}}(x^0)] \right\rangle : (\mathbf{\Pi y} : \mathbf{Y}.\mathbf{Z})^0$$

où les variables $\mathbf{x}$ sont les variables de graphe du contexte Γ.
Le terme d'arête est obtenu de la manière suivante:

$$\Gamma \vdash \left\langle \lambda y^0.t^2[y^1\backslash y^0,y^2\backslash\mathtt{Ref}(y^0)],\ \lambda y^0.\lambda y^1.\lambda y^2.t^2 \right\rangle : (\mathbf{\Pi y} : \mathbf{Y}.\mathbf{Z})^2((\lambda\mathbf{y}.\mathbf{t})^0,(\lambda\mathbf{y}.\mathbf{t})^1)$$

On définit:

$$\lambda\mathbf{y}.\mathbf{t} = (\left\langle \lambda y^0.t^0,\ \lambda y^0.\lambda y^1.\lambda y^2.t^2[\vec{x}^1\backslash\vec{x}^0,\vec{x}^2\backslash\vec{\mathtt{Ref}}(x^0)] \right\rangle,\ \left\langle \lambda y^0.t^2[y^1\backslash y^0,y^2\backslash\vec{\mathtt{Ref}}(y^0)],\ \lambda y^0.\lambda y^1.\lambda y^2.t^2 \right\rangle)$$

Définition 7.1.13 (Application sur les graphes réflexifs). Le jugement:

$$\frac{\Gamma \vdash \mathbf{t} : \mathbf{\Pi y} : \mathbf{Y}.\mathbf{Z} \qquad \Gamma \vdash \mathbf{a} : \mathbf{Y}}{\Gamma \vdash (\mathbf{t})\mathbf{a} : \mathbf{Z}}$$

est donné par les deux jugements:

$$\Gamma \vdash (\pi t^0)a^0 : Z^0 \qquad \Gamma \vdash (\pi' t^2)a^0a^1a^2 : Z^2((\pi t^0)a^0,(\pi t^1)a^1)$$

On remarquera en particulier que nous avons la réduction (double):

$$\begin{array}{rcl}(\lambda\mathbf{y}.\mathbf{t})\mathbf{a} & = & ((t^0)a^0 : Z^0[y^0\backslash a^0],\ (t^2)a^0a^1a^2 : Z^2(t^0[y^0\backslash a^0],t^1[y^1\backslash a^1])) \\ & = & (t^0[y^0\backslash a^0] : Z^0[y^0\backslash a^0],\ t^2[y^0\backslash a^0,y^1\backslash a^1,y^2\backslash a^2] : Z^2(t^0[y^0\backslash a^0],t^1[y^1\backslash a^1]))\end{array}$$

Ici, l'égalité représente l'égalité de deux termes fins. Mais nous ne définissons pas en général d'égalité grasse.

Note 7.1.4. Nous pourrions définir un type somme, mais nous ne nous en servons pas par la suite.

7.1.3 Lemme de substitution

Pour les règles fines, la substitution découle de ce que nous n'employons que les constructions classiques (produit, somme) et que les autres termes sont introduits à l'aide de règles d'introduction d'une forme tout à fait ordinaire. Toutes les règles de graphes que nous utilisons peuvent être dévissées en des règles usuelles, pour lesquelles on peut faire l'induction pour le lemme de substitution.

Mais pour le calcul gras, avons-nous une substitution ? On peut en effet se poser la question vu que l'on peut construire des termes complexes. Étant donné deux termes de graphes $\mathbf{\Gamma} \vdash \mathbf{t} : \mathbf{X}$ et $\mathbf{\Gamma}, \mathbf{x} : \mathbf{X}, \mathbf{\Delta} \vdash \mathbf{r} : \mathbf{Y}$, cela correspond aux jugements :

$$\begin{aligned} \Gamma^0 &\vdash t^0 : X^0 \\ \Gamma^2 &\vdash t^2 : X^2 \\ \Gamma^0, x^0 : X^0, \Delta^0 &\vdash r^0 : Y^0 \\ \Gamma^2, x^0 : X^0, x^1 : X^1, x^2 : X^2, \Delta^2 &\vdash r^2 : Y^2(r^0, r^1) \end{aligned}$$

nous pouvons imaginer une substitution naïve qui consiste à remplacer la variable x^0 dans le troisième jugement par t^0, la variable x^1 par t^1 et la variable x^2 par t^2. On peut simuler dans le calcul fin cette substitution dans le cas où $\mathbf{r}$ et $\mathbf{t}$ ne contiennent pas d'abstraction (on procède par induction). Sous ces hypothèses, nous avons la règle :

$$\frac{\mathbf{\Gamma} \vdash \mathbf{t} : \mathbf{X} \qquad \mathbf{\Gamma}, \mathbf{x} : \mathbf{X}, \mathbf{\Delta} \vdash \mathbf{r} : \mathbf{Y}}{\mathbf{\Gamma}, \mathbf{\Delta}[\mathbf{x} \backslash \mathbf{t}] \vdash \mathbf{r}[\mathbf{x} \backslash \mathbf{t}] : \mathbf{Y}[\mathbf{x} \backslash \mathbf{t}]}$$

avec la notion de substitution naïve.

Remarque 7.1.14.

1. Mais dans le cas où $\mathbf{r}$ ou $\mathbf{t}$ contient une λ-abstraction, il y a un problème de substitutions vis-à-vis des termes `Ref` (qui sont introduits par un autre processus que par les règles d'introduction d'opérateurs), voir la discussion à ce sujet dans la section

6.1.3. Prenons par exemple, deux jugements : $\mathbf{\Gamma} \vdash \mathbf{t} : \mathbf{X}$ et un jugement $\mathbf{\Gamma}, \mathbf{x} : \mathbf{X} \vdash \boldsymbol{\lambda}\mathbf{y}.\mathbf{r} : \boldsymbol{\Pi}\mathbf{y} : \mathbf{Y}.\mathbf{Z}$. Nous avons donc deux jugement :

$$\Gamma^0 \vdash \quad t^0 : X^0$$

$$\Gamma^0, x^0 : X^0 \quad \vdash \quad \left\langle \lambda y^0.r^0, \lambda y^0.\lambda y^1.\lambda y^2.r^2[\vec{u}^1 \backslash u^0, \vec{u}^2 \backslash \mathtt{Ref}_U(u^0), x^1 \backslash x^0, x^2 \backslash \mathtt{Ref}_X(x^0)] \right\rangle :$$
$$\Sigma_{f:\Pi_{y^0;Y^0}.Z^0}.\Pi_{y^2:y^0-y^1:\mathbf{Y}}.Z^2(\vec{u}^1 \backslash \vec{u}^0, \vec{u}^2 \backslash \vec{\mathtt{Ref}}_{\mathbf{U}}(u^0), x^1 \backslash x^0, x^2 \backslash \mathtt{Ref}_X(x^0), y^0, y^1, y^2, fy^0, fy^1)$$

où les variables de graphes de $\mathbf{\Gamma}$ sont les variables $\mathbf{u}$. Si l'on fait la substitution naïve, on obtient :

$$\Gamma^0 \vdash \left\langle \lambda y^0.r^0[x^0 \backslash t^0], \lambda y^0.\lambda y^1.\lambda y^2.r^2[\vec{u}^1 \backslash \vec{u}^0, \vec{u}^2 \backslash \vec{\mathtt{Ref}}_{\mathbf{U}}(u^0), x^1 \backslash t^0, x^2 \backslash \mathtt{Ref}_X(t^0)] \right\rangle :$$
$$\Sigma_{f:\Pi_{y^0;Y^0}.Z^0}.\Pi_{y^2:y^0-y^1:\mathbf{Y}}.Z^2(\vec{u}^1 \backslash \vec{u}^0, \vec{u}^2 \backslash \vec{\mathtt{Ref}}_{\mathbf{U}}(u^0), x^1 \backslash t^0, x^2 \backslash \mathtt{Ref}_X(t^0), y^0, y^1, y^2, fy^0, fy^1)$$

Or le terme de réflexivité correspondant à x dans

$$r^2[\vec{u}^1 \backslash \vec{u}^0, \vec{u}^2 \backslash \mathtt{Ref}_X(u^0), x^1 \backslash t^0, x^2 \backslash \mathtt{Ref}_X(t^0)]$$

ne devrait pas être du type $\mathtt{Ref}_X(\dots)$ mais dépendre des termes de réflexivité $\mathtt{Ref}_{\mathbf{U}}$ des types dans $\mathbf{\Gamma}$. On retrouve ici le problème que nous avons rencontré en sémantique : pour pouvoir faire la substitution, les termes « devraient commuter avec les termes de réflexivité ».

De ce fait, nous n'avons pas la substitution en général.

2. Cela n'est pas trop gênant ; en effet, nous ne nous servons pas du système pour calculer, mais pour définir des termes. Le calcul gras est avant tout un système d'abréviations, le système « réel » étant le calcul fin. Il est d'ailleurs probablement possible de définir une notion de substitution spéciale par induction.
3. Nous n'avons donc pas un calcul complet, mais nous bénéficions de propriétés intéressantes : l'affaiblissement et la contraction, ce qui rend le calcul très pratique pour définir des termes.
4. Notons enfin que nous avons la β réduction : la substitution naïve nous donne les « égalités » :

$$(\boldsymbol{\lambda}\mathbf{y}.\mathbf{t})\mathbf{a} = \mathbf{t}[\mathbf{y} \backslash \mathbf{a}] = (t^0[y^0 \backslash a^0],\ t^2[y^0 \backslash a^0, y^1 \backslash a^1, y^2 \backslash a^2])$$

L'égalité étant une égalité composante par composante.

7.2 Calcul sur ω

Dans cette section, nous introduisons un calcul (λ-ω) qui permet de traiter la notion de monotonie sur les entiers. Par rapport au système T de Gödel, nous rajoutons un type pour les inégalités (larges) et le récurseur correspondant aux preuves de monotonie.

Le système λ-ω, en induisant la notion de preuve de monotonie, a un intérêt en soi, mais il nous sert aussi de sous-système pour manipuler l'arité ω dans le calcul des termes ordinaux (que nous présentons par la suite).

Voici donc les règles spécifiques de λ-ω:

Règles de construction de types

Comme nous l'avons vu dans l'introduction, le type $\boldsymbol{\omega}$ se décrit à l'aide de trois règles:

$$\frac{}{\vdash \omega \text{ type}} \qquad \frac{}{n^0 : \omega, n^1 : \omega \vdash n^0 - n^1 \text{ type}} \qquad \frac{\Gamma \vdash t : \omega}{\Gamma \vdash \mathtt{Ref}_\omega(t) : \omega^2(t,t)}$$

On en déduit immédiatement une règle de graphe:

$$\frac{\begin{array}{rcl} & \vdash & \omega \text{ type} \\ n^0 : \omega, n^1 : \omega & \vdash & n^0 - n^1 \text{ type} \\ n^0 : \omega & \vdash & \mathtt{Ref}_\omega(n^0) : n^0 - n^0 \end{array}}{\begin{array}{rcl} & \vdash & \boldsymbol{\omega}\ \mathtt{grfR} \end{array}}$$

Nous résumons tout ceci en nous donnant une règle de formation de graphe:

$$\frac{}{\vdash \boldsymbol{\omega}\ \mathtt{grfR}}$$

Règles de formation de termes

Les règles que nous introduisons ici reflètent la construction de la Ψ-algèbre libre correspondant à ω. Nous présentons d'abord les règles dans leur forme compactée:

$$\frac{}{\vdash \mathbf{0} : \boldsymbol{\omega}} \qquad \frac{\Gamma \vdash \mathbf{t} : \boldsymbol{\omega}}{\Gamma \vdash \mathbf{st} : \boldsymbol{\omega}}$$

Dans leur forme développée, les règles précédentes se présentent sous la forme de quatre règles :

$$\frac{}{\vdash 0^0 : \omega^0} \qquad \frac{\Gamma \vdash t^0 : \omega^0}{\Gamma \vdash s^0 t^0 : \omega^0}$$

$$\frac{}{\vdash 0^2 : \omega^2(0^0,0^1)} \qquad \frac{\Gamma \vdash t^2 : \omega^2(t^0,t^1)}{\Gamma \vdash s^2 t^2 : \omega^2(s^0 t^0, s^1 t^1)}$$

À quoi s'ajoute la règle d'inflationnarité de l'opérateur s :

$$\frac{\Gamma \vdash t^2 : \omega^2(t^0,t^1)}{\Gamma \vdash \mathrm{i}^2(t^2) : \omega^2(t^0, s t^1)}$$

Note 7.2.1. Ici, nous devons faire une remarque à propos des constantes de termes de graphe réflexif. Les indices 0 et 1 ne sont qu'une décoration ainsi que nous en avions fait la remarque à la définition 7.1.8 ; pour alléger l'écriture, nous supprimons les deux indices 0 et 1 sur les symboles de fonction. Nous gardons l'indice 2 pour distinguer les opérations sur les arêtes.

En dehors du cadre des conventions, nous employons les lettres n, m, ... pour désigner des variables de type ω et les lettres p, q, ... pour les preuves de type $n - m$.

Règles d'élimination des termes du type ω

Nous avons deux règles d'élimination, une pour les sommets de ω et une pour les arêtes de ω. Ces règles sont construites à l'image de la ω-algèbre correspondante.

Le principe de la règle est classique : on suppose que l'on dispose d'un graphe réflexif qui est une ω-algèbre (du point de vue de la sémantique). On conclut alors par un terme qui représente l'épine entre

ω et le graphe réflexif en question. L'épine étant un morphisme de graphe, ce terme est en fait un terme de graphe réflexif.

Le rôle des prémices est donc de montrer que le graphe réflexif sur lequel on fait l'induction est une ω-algèbre ; il y a donc une correspondance fidèle entre les prémices de la règle d'élimination et les règles d'introduction pour $\boldsymbol{\omega}$ (celles-la même qui construisent la ω-algèbre libre). La seule différence vient du fait que le graphe réflexif introduit peut dépendre de la variable sur laquelle on fait l'induction. De ce fait, les variables dans le membre gauche des prémices sont dupliquées par rapport à celles des règles d'introduction des termes de $\boldsymbol{\omega}$: on en a une pour $\boldsymbol{\omega}$ et l'autre pour le graphe réflexif.

Les deux règles d'induction peuvent être réunies en une règle de graphe réflexif :

$$\frac{\begin{array}{rcl}\Gamma,\mathbf{n}:\boldsymbol{\omega} & \vdash & \mathbf{X}(\mathbf{n})\ \mathtt{grfR} \\ \Gamma & \vdash & \mathbf{a}:\mathbf{X}(\mathbf{0}) \\ \Gamma,\mathbf{n}:\boldsymbol{\omega},\ \mathbf{x}:\mathbf{X} & \vdash & \mathbf{t}(\mathbf{n},\mathbf{x}):\mathbf{X}(\mathbf{sn}) \\ \Gamma,\mathbf{n}:\boldsymbol{\omega},\ \mathbf{x}:\mathbf{X} & \vdash & \mathtt{i}(\mathbf{n},\mathbf{x}):X^2(n^0,sn^1,\mathtt{i}^2n^2,x^0,t^1(n^1,x^1))\end{array}}{\Gamma,\mathbf{m}:\boldsymbol{\omega} \vdash \mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}\mathbf{m}:\mathbf{X}(\mathbf{m})}$$

Remarque 7.2.1.

1. Les crochets que nous avons mis dans le terme $\mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}\mathbf{m}$ indiquent que les variables n^0, n^1, n^2, x^0, x^1 et x^2 sont liées dans le terme $\mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}\mathbf{m}$.
2. On voit dans cet exemple tout l'intérêt des conventions que nous avons faites. D'une part, elles factorisent tout ce qui correspond au graphe réflexif. L'écriture des prémices rend compte correctement de la structure sous-jacente de graphe réflexif. D'autre part, la conclusion est double, ceci traduit le fait qu'il s'agit d'une règle qui s'applique « moralement » aux graphes réflexifs.

Le terme de graphe réflexif $\mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}\mathbf{m}$ recouvre les deux termes :

$$\begin{aligned}\Gamma,\mathbf{m}:\omega &\vdash \mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}m^0 : X^0(m^0)\\ \Gamma,\mathbf{m}:\omega &\vdash \mathrm{I}^2{}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}m^0m^1m^2 : X^2(\mathbf{m},\mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}m^0,\mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}m^1)\end{aligned}$$

Remarque 7.2.2. On voit ici que le terme de réflexivité n'a rien à voir avec l'induction, c'est un artefact technique dont le rôle est simplement d'éviter d'utiliser de manière explicite le calcul de la SMC. Nous pourrions en effet travailler entièrement avec ce calcul, mais cela alourdit de manière non négligeable la construction de termes.

Nous verrons plus loin que l'on peut construire à l'aide du principe d'induction un terme `Ref` qui représente la réflexivité. Nous déciderons à ce moment là d'identifier ce terme `Ref` avec $\mathtt{refl}_\omega$ à l'aide d'une règle de réduction.

Égalité des termes

Nous avons trois règles qui correspondent à la réduction des récurseurs. Il s'agit de :

$$\begin{aligned}\mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}\mathbf{0} &\rhd \mathbf{a}\\ \mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}(\mathbf{sn}) &\rhd \mathbf{t}(\mathbf{n},\mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}\mathbf{n})\\ \mathrm{I}^2{}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}n^0(sn^1)(\mathrm{i}^2n^2) &\rhd \mathbf{i}(\mathbf{n},\mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}n^0,\mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}n^1,\mathrm{I}^2{}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}\mathbf{n})\end{aligned}$$

Ici, les deux premières règles de réduction sont des abréviations. La première réduction désigne les deux réductions :

$$\begin{aligned}\mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}0 &\rhd a^0\\ \mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}000^2 &\rhd a^2\end{aligned}$$

La deuxième équivaut à :

$$\begin{aligned}\mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}(sn) &\rhd t^0(n,\mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}n)\\ \mathrm{I}^2{}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}(\mathbf{sn}) &\rhd t^2(\mathbf{n},\mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}n^0,\mathrm{I}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}n^1,\mathrm{I}^2{}_{[\mathbf{n},\mathbf{x}]\mathtt{ati}}\mathbf{n})\end{aligned}$$

Remarque 7.2.3. Ces règles sont tout à fait naturelles. Dans cette thèse, nous ne prouvons pas le théorème de normalisation (dont nous faisons la conjecture) et elles peuvent être lues comme des égalités.

Remarque 7.2.4. Les règles de réduction respectent le typage, le lecteur pourra le vérifier. Ceci donne la propriété de « subject reduction » au calcul.

7.2.1 Le système T vu dans λ-ω, règles dérivées

Nous montrons dans cette section que le système T s'inscrit de manière très naturelle dans λ-ω, pour cela, nous établissons un codage du principe d'induction de T dans λ-ω.

En premier lieu, il convient de faire une petite remarque. L'ensemble ordonné ω a la propriété que la construction inductive des sommets est indépendante de la construction des arêtes (ce qui n'est pas le cas de $\mathbb{W}$ par exemple). Ceci peut être observé dans la syntaxe à deux occasions.

Premièrement, les termes 0 et sn^0 sont indépendant des variables de type 2. Ceci provient du fait que les arités pour $\boldsymbol{\omega}$ sont des graphes réflexifs discrets (ce qui ne sera donc pas le cas pour $\mathbb{W}$) : les sommets sont construits sans contrainte d'ordre, c'est-à-dire sans dépendance des variables d'arête. Ceci se traduit d'autre part par le fait que pour le schéma d'élimination de ω, les termes a^0 et t^0 sont également indépendants des variables de type 2. De ce fait, les deux règles d'évaluation des termes $\mathrm{I}_{\mathtt{[n,x]ati}}n$ (de type 0) ne dépendent pas des trois termes a^2, t^2 et i.

Il y a donc une sorte de sous-principe d'induction, le principe qui correspond justement à l'induction sur $\mathbb{N}$. Nous utilisons cette remarque pour coder le principe d'induction du système T dans λ-ω.

Définition 7.2.5. Nous commençons par traduire les types du système T dans λ-ω. Pour cela, nous nous appuyons sur le fait que nous pouvons construire de manière uniforme un graphe réflexif étant donné n'importe quel type X. Il suffit d'employer le type ω [34] pour l'ensemble des arêtes. La construction est la suivante :

34. En fait, n'importe quel autre type défini dans le contexte vide convient.

$$\frac{\begin{array}{rcl}\Gamma & \vdash & X \text{ type} \\ \Gamma,x^0 : X,x^1 : X & \vdash & \omega \text{ type} \\ \Gamma,x^0 : X & \vdash & 0 : \omega\end{array}}{\Gamma \vdash (X,\omega,0) : \texttt{grfR}}$$

Par la suite, on note $\hat{X}$ le graphe correspondant au type X. Étant donné un terme $\Gamma \vdash t : X$, on peut construire un terme de graphe $\hat{t}$:

$$\frac{\begin{array}{rcl}\Gamma & \vdash & t : X \\ \Gamma,x : X^0,x^1 : X^1 & \vdash & 0 : \omega\end{array}}{\Gamma \vdash (t,0) : \hat{X}}$$

On note par la suite $\hat{t}$ la paire $(t,0)$.

Supposons que nous disposons des trois jugements :

$$\begin{array}{rcl}\Gamma,n : \omega & \vdash & X(n) \text{ type} \\ \Gamma & \vdash & a : X(0) \\ \Gamma,n : \omega,x : X(n) & \vdash & t(n,x) : X(sn)\end{array}$$

Nous pouvons alors en déduire le jugement :

$$\left.\begin{array}{rcl}\Gamma,\mathbf{n} : \boldsymbol{\omega} & \vdash & X(n^0) \text{ type} \\ \Gamma,\mathbf{n} : \boldsymbol{\omega},x^0 : \widehat{X}^0(n^0),x^1 : \widehat{X}^1(n^1) & \vdash & \omega \text{ type} \\ \Gamma,,x^0 : \widehat{X}^0(n^0) & \vdash & 0 : \omega\end{array}\right\} \quad \Gamma,\mathbf{n} : \boldsymbol{\omega} \vdash \widehat{X}(\mathbf{n}) : \texttt{grfR}$$

$$\left.\begin{array}{rcl}\Gamma & \vdash & a : X(0) \\ \Gamma & \vdash & 0 : \boldsymbol{\omega}\end{array}\right\} \quad \Gamma \vdash \widehat{a} : \widehat{X}(\mathbf{0})$$

$$\left.\begin{array}{rcl}\Gamma,\mathbf{n} : \boldsymbol{\omega},x^0 : \widehat{X}^0(n^0) & \vdash & t : X(sn^0) \\ \Gamma,\mathbf{n} : \boldsymbol{\omega},\mathbf{x} : \widehat{X} & \vdash & 0 : \omega\end{array}\right\} \quad \Gamma,\mathbf{n} : \boldsymbol{\omega},\mathbf{x} : \widehat{X}(\mathbf{n}) \vdash \widehat{t} : \widehat{X}(\mathbf{sn})$$

$$\left.\begin{array}{rcl}\Gamma,\mathbf{n} : \boldsymbol{\omega},\mathbf{x} : \widehat{X} & \vdash & 0 : \omega\end{array}\right\} \quad \Gamma,\mathbf{n} : \boldsymbol{\omega},\mathbf{x} : \widehat{X}(\mathbf{n} \vdash 0 : \widehat{X}^2(\mathbf{n},x^0,tx^1$$

$$\Gamma,\mathbf{k} : \boldsymbol{\omega} \vdash \mathrm{I}_{[\mathbf{n},\mathbf{x}]\widehat{a}\widehat{t}}\mathbf{k} : \widehat{X}(\mathbf{k})$$

Note 7.2.2. Si l'on ne retient que la première composante de la deuxième conclusion, on s'aperçoit que l'on fait un jugement d'ensemble et non de graphe. Nous en déduisons une règle dérivée :

$$\frac{\Gamma,n:\omega \vdash X(n)\ \text{type} \quad \Gamma \vdash a : X(0) \quad \Gamma,n:\omega,x:X \vdash t : X(sn)}{\Gamma,k:\omega \vdash \mathrm{I}^{\mathbb{N}}_{[n,x]at}k : X(k)}$$

où le terme $\mathrm{I}^{\mathbb{N}}_{[n,x]at}k = \mathrm{I}^{0}_{[n,x]\widehat{a}\widehat{t}}k$.

Note 7.2.3. Nous pouvons observer qu'il s'agit de la règle ususelle de la récursion sur $\mathbb{N}$ dans le cas des types dépendants. La seule différence avec le système T vient du fait que le type du terme se construit en même temps que la récursion ; par conséquent, il y a des substitutions dans les types parallèlement à l'induction sur les termes.

Nous sommes maitenant prêts à traduire le système T dans λ-ω. Le système T est le calcul simplement typé avec un type atomique $\mathbb{N}$, un symbole fonctionnel 0-aire 0 et un symbole fonctionnel $s : \mathbb{N} \to \mathbb{N}$, et la règle d'induction :

$$\frac{\Gamma \vdash a : X \qquad \Gamma,n:\mathbb{N},x:X \vdash t : X}{\Gamma,k:\mathbb{N} \vdash I_{[n,x]at}k : X}$$

Nous traduisons maintenant le type $\mathbb{N}$ par $\overline{\mathbb{N}} = \omega$ et tout type fonctionnel $X \to Y$ par $\Pi x : \overline{X}.\overline{Y}$. Pour les termes, la traduction $\overline{(-)}$ est la suivante :

$$\begin{array}{rcl} 0 & \mapsto & 0 \\ s & \mapsto & \lambda n.\mathrm{s}(n) \\ \lambda x.t & \mapsto & \lambda x.\overline{t} \\ (t)a & \mapsto & (\overline{t})\overline{a} \\ I_{[n,x]at}k & \mapsto & \mathrm{I}^{\mathbb{N}}_{[n,x]\overline{a}\overline{t}}k \end{array}$$

On notera que les règles de réduction du terme $I_{[n,x]at}k$ correspondent aux règles usuelles de réductions du récurseur.

Conjectures 7.2.6. Pour autant, nous avons de bonnes raisons de penser que λ-ω n'est pas plus puissant que le système T. Il y a plusieurs manière d'appréhender ce fait.

Premièrement, si l'on considère les termes de λ-ω construits uniquement à l'aide du type atomique ω, du produit et de la somme, des symboles de type 0, et si l'on se restreint à l'induction sur $\mathbb{N}$ comme présenté plus haut, on peut alors procéder à la traduction inverse : c'est-à-dire ω en $\mathbb{N}$, 0 en 0, etc. En effet, dans ce cas, les types ne sont pas dépendants, et le schéma d'induction est très semblable au schéma d'induction de T.

Deuxièmement, si l'on se place dans le système λ-ω au complet et que l'on obtient un terme $x_1 : \overline{X_1}, \ldots, x_n : \overline{X_n} \vdash t : \overline{Y}$, ce terme représente une fonction (ou plutôt une fonctionnelle) ; nous faisons la conjecture que le terme t peut être obtenu dans T, c'est-à-dire que $[\![t]\!] = [\![\overline{r}]\!]$ pour un certain terme de T (l'égalité étant au niveau sémantique).

Mais, il nous semble également que le système entier peut être codé dans T ; en effet, le schéma d'induction est élémentaire, nous venons de voir que la partie sur les sommets est (en dehors des types dépendants) semblable au schéma classique d'induction sur $\mathbb{N}$ et la partie concernant les arêtes peut être assimilée à une induction sur des notations binaires d'entiers.

7.2.2 Exemples de termes

Commençons par prouver l'inégalité $i \leq i$ sur ω. Elle est représentée par le terme $\mathtt{Ref} \stackrel{\text{def}}{=} \lambda k.\mathrm{I}^{\mathbb{N}}_{[n,p]0^2(\mathrm{s}^2p)}k$ dont voici la preuve de typage :

$$\dfrac{\dfrac{n : \omega \vdash n - n \text{ type} \qquad \vdash 0^2 : 0 - 0 \qquad n : \omega, p : n - n \vdash \mathrm{s}^2 p : sn - sn}{n : \omega \vdash \mathrm{I}^{\mathbb{N}}_{[p]0^2(\mathrm{s}^2p)} n : n - n}}{\vdash \lambda n.\mathrm{I}^{\mathbb{N}}_{[p]0^2(\mathrm{s}^2p)} n : \Pi n : \omega.n - n}$$

Remarque 7.2.7. Nous refermons ici, en quelque sorte, une parenthèse technique. Nous avons introduit pour $\boldsymbol{\omega}$ un terme de réflexivité, mais si on se réfère à la construction sémantique de $\boldsymbol{\omega}$ comme Ψ-

algèbre, ce terme de réflexivité découle du principe d'induction. Et si nous avions travaillé avec des graphes et non des graphes réflexifs, nous aurions obtenu de la même manière ce terme de réflexivité. Pour refermer la parenthèse, nous pourrions alors considérer une règle de réduction :

$$\mathtt{Ref}_\omega(n^0) \rhd \mathrm{I}^{\mathbb{N}}_{[p]0^2(\mathrm{s}^2p)}n^0$$

Cette règle pourrait paraître aller d'un terme simple vers un terme compliqué, mais en fait, elle passe du terme « non défini » vers le terme « défini ».

Notons qu'étant donné le terme $\underline{n} = \underbrace{\mathrm{s}(\mathrm{s}(\cdots \mathrm{s}(0)))}_{n \text{ fois s}}$, nous avons l'égalité (par induction sur n) : $\mathrm{I}^{\mathbb{N}}_{[p]0^2(\mathrm{s}^2p)}n = \underbrace{\mathrm{s}^2(\mathrm{s}^2(\cdots \mathrm{s}^2}_{n \text{ fois } \mathrm{s}^2}(0^2))) : \omega^2(\overline{n},\overline{n})$, le deuxième membre de l'égalité étant le représentant naturel de la réflexivité.

Mais nous préférons ne pas considérer la règle pour ce qu'elle peut interférer avec la substitution.

Exemple 7.2.8. Nous disposons également d'un jugement $n : \omega \vdash \mathrm{I}^{\mathbb{N}}_{[p]0^2(\mathrm{i}^2p)}n : 0 - n$. On définit $\mathtt{Z}(n) = \mathrm{I}^{\mathbb{N}}_{[p]0^2(\mathrm{i}^2p)}n$.

Nous pouvons observer maintenant que l'addition est monotone :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{n},\mathbf{m} : \omega \vdash \omega\ \mathtt{grfR}}}{\mathbf{n} : \omega \vdash \mathbf{n} : \omega}}{\mathbf{n},\mathbf{m},\mathbf{x} : \omega \vdash \mathbf{sx} : \omega}}{\mathbf{n},\mathbf{m},\mathbf{x} : \omega \vdash \mathrm{i}^2x^2\omega(x^0,\mathrm{s}^2x^1)}}{\dfrac{\mathbf{n},\mathbf{m} : \omega \vdash \mathrm{I}_{[\mathbf{x}]\mathbf{n}(\mathbf{sx})(\mathrm{i}^2x^2)}\mathbf{m} : \omega}{\vdash \boldsymbol{\lambda}\mathbf{n}.\boldsymbol{\lambda}\mathbf{m}.\mathrm{I}_{[\mathbf{x}]\mathbf{n}(\mathbf{sx})(\mathrm{i}^2x^2)}\mathbf{m} : \omega \multimap \omega \multimap \omega}}$$

On définit le terme $\mathbf{Add} = \boldsymbol{\lambda}\mathbf{n}.\boldsymbol{\lambda}\mathbf{m}.\mathrm{I}_{[\mathbf{x}]\mathbf{n}(\mathbf{sx})(\mathrm{i}^2x^2)}\mathbf{m}$ qui représente l'addition et la preuve de monotonie de l'addition.

De même, on peut montrer que la multiplication est monotone :

$$\begin{array}{rcl}
\mathbf{n},\mathbf{m} : \omega & \vdash & \omega\ \mathtt{grfR} \\
\mathbf{n} : \omega & \vdash & \mathbf{0} : \omega \\
\mathbf{n},\mathbf{m},\mathbf{y} : \omega & \vdash & \mathrm{I}_{[\mathbf{x}]\mathbf{n}(\mathbf{sx})(\mathrm{i}^2 x^2)}\mathbf{y} : \omega \\
\mathbf{n},\mathbf{m},\mathbf{y} : \omega & \vdash & \mathrm{I}_{[\mathbf{z}]\mathbf{z}(\mathrm{I}_{[\mathbf{x}]\mathbf{n}(\mathbf{sx})(\mathrm{i}^2 x^2)}\mathbf{0})(\mathrm{s}^2 z^2)(\mathrm{i}^2 z^2)} : y^0 - \mathrm{I}_{[\mathbf{x}]\mathbf{n}(\mathbf{sx})(\mathrm{i}^2 x^2)} y^1 \\
\hline
\mathbf{n},\mathbf{m} : \omega & \vdash & \mathrm{I}_{[\mathbf{y}]\mathbf{0}(\mathbf{Add}\ \mathbf{n}\ \mathbf{y})\mathrm{I}_{[\mathbf{z}]\mathbf{z}(\mathrm{I}_{[\mathbf{x}]\mathbf{n}(\mathbf{sx})(\mathrm{i}^2 x^2)}\mathbf{0})(\mathrm{s}^2 z^2)(\mathrm{i}^2 z^2)}\mathbf{m}} : \omega \\
\hline
 & \vdash & \boldsymbol{\lambda}\mathbf{n}.\boldsymbol{\lambda}\mathbf{m}.\mathrm{I}_{[\mathbf{y}]\mathbf{0}(\mathbf{Add}\ \mathbf{n}\ \mathbf{y})\mathrm{I}_{[\mathbf{z}]\mathbf{z}(\mathrm{I}_{[\mathbf{x}]\mathbf{n}(\mathbf{sx})(\mathrm{i}^2 x^2)}\mathbf{0})(\mathrm{s}^2 z^2)(\mathrm{i}^2 z^2)}\mathbf{m}} : \omega \multimap \omega \multimap \omega
\end{array}$$

La multiplication (monotone) est représentée par le terme

$$\boldsymbol{\lambda}\mathbf{n}.\boldsymbol{\lambda}\mathbf{m}.\mathrm{I}_{[\mathbf{y}]\mathbf{0}(\mathbf{Add}\ \mathbf{n}\ \mathbf{y})\mathrm{I}_{[\mathbf{z}]\mathbf{z}(\mathrm{I}_{[\mathbf{x}]\mathbf{n}(\mathbf{sx})(\mathrm{i}^2 x^2)}\mathbf{0})(\mathrm{s}^2 z^2)(\mathrm{i}^2 z^2)}\mathbf{m}$$

7.3 Calcul sur $\mathbb{W}$

Le système λ-$\mathbb{W}$ est composé à partir du système λ-ω auquel on rajoute un certain nombre de règles de construction afin d'obtenir un système dans lequel on puisse utiliser le schéma d'induction de l'ordre partiel $\mathbb{W}$ que nous avons vu au chapitre 5.

Règle additionnelle pour les types

$$\frac{}{\vdash \mathbf{W} : \mathtt{grfR}}$$

Règles additionnelles pour les termes

$$\frac{}{\vdash \mathbf{0} : \mathbf{W}} \qquad \frac{\boldsymbol{\Gamma} \vdash \mathbf{t} : \mathbf{W}}{\boldsymbol{\Gamma} \vdash \mathbf{st} : \mathbf{W}}$$

Ces deux règles sont construites à l'image de celles que nous avions pour ω. Il reste à exprimer l'inflationnarité de $\mathbf{s}$:

$$\frac{\Gamma \vdash t^2 : \mathbb{W}^2(t^0, t^1)}{\Gamma \vdash \underline{\mathrm{i}}^2 t^2 : \mathbb{W}^2(t^0, st^1)}$$

et le cas limite :

$$\frac{\Gamma \vdash \mathbf{u} : \omega \multimap \mathbf{W}}{\Gamma \vdash \mathbf{lu} : \mathbf{W}}$$

Règles d'élimination du type $\mathbb{W}$

De la même manière que pour ω, nous avons deux règles d'élimination, une pour les éléments de $\mathbb{W}$ et une pour les arêtes de $\mathbb{W}$, ce qui se traduit par le jugement de graphe réflexif :

$$\begin{array}{rcl}
\Gamma,\mathbf{v} : \mathbf{W} & \vdash & \mathbf{X}(\mathbf{v})\ \mathtt{grfR} \\
\Gamma & \vdash & \mathbf{a} : \mathbf{X}(\mathbf{0}) \\
\Gamma,\mathbf{v} : \mathbf{W},\mathbf{x} : \mathbf{X} & \vdash & \mathbf{t}(\mathbf{v},\mathbf{x}) : \mathbf{X}(\mathbf{sv}) \\
\Gamma,\mathbf{v} : \mathbf{W},\ \mathbf{x} : \mathbf{X} & \vdash & \mathtt{i}(\mathbf{v},\mathbf{x}) : X^2(v^0,\underline{s}v^1,\mathtt{i}^2v^2,x^0,t^1(v^1,x^1)) \\
\Gamma,\mathbf{u} : \omega \multimap \mathbf{W},\mathbf{f} : \omega \multimap \mathbf{X}\{\mathbf{u}\} & \vdash & \mathbf{t_l}(\mathbf{u},\mathbf{f}) : \mathbf{X}(\mathbf{lu}) \\
\hline
\Gamma,\mathbf{v} : \mathbf{W} & \vdash & \mathbf{J}_{[\mathbf{v},\mathbf{u},\mathbf{x},\mathbf{f}]\mathtt{atit_l}}\mathbf{v} : \mathbf{X}(\mathbf{v})
\end{array}$$

Réduction des termes

Nous avons comme dans le cas de ω des règles de réduction pour les récurseurs. En notant $\mathbf{J}_{\_}$ le terme $\mathbf{J}_{[\mathbf{v},\mathbf{u},\mathbf{x},\mathbf{f}]\mathtt{atit_l}}$,

$$
\begin{aligned}
\mathbf{J}\_\mathbf{0} &\rhd \mathbf{a} \\
\mathbf{J}\_\mathbf{sv} &\rhd \mathbf{t}(\mathbf{v},\mathbf{J}_{[\mathbf{v},\mathbf{u},\mathbf{x},\mathbf{f}]\mathbf{atit}_1}\mathbf{v}) \\
\mathbf{J}^2_{\underline{\ }}\, x^0(\underline{\mathrm{s}}x^1)(\underline{\mathrm{i}}^2 x^2) &\rhd \mathbf{i}(\mathbf{x},\mathbf{J}\_x^0,\mathbf{J}\_x^1,\mathbf{J}^2_{\underline{\ }}\mathbf{x}) \\
\mathbf{J}\_(\underline{\mathrm{l}}\left\langle u,p_u\right\rangle) &\rhd t^0_l(u,\ \left\langle \Pi k:\omega.\mathbf{J}\_uk,\ \Pi_{p:k-j:\omega}.\mathbf{J}^2_{\underline{\ }}(uk)(uj)(p_u kjp)\right\rangle) \\
\mathbf{J}\_\underline{\mathrm{l}}\left\langle u,p_u\right\rangle \underline{\mathrm{l}}\left\langle v,p_v\right\rangle \underline{\mathrm{l}}^2\left\langle p_{uv},p'_{uv}\right\rangle &\rhd t^2_l(\left\langle u,p_u\right\rangle,\left\langle v,p_v\right\rangle,\left\langle p_{uv},p'_{uv}\right\rangle, \\
&\quad \left\langle \Pi_{k:\omega}.\mathbf{J}\_uk,\ \Pi_{p:k-j:\omega}.\mathbf{J}\_(uk)(uj)(p_u kjp)\right\rangle, \\
&\quad \left\langle \Pi_{k:\omega}.\mathbf{J}\_vk,\ \Pi_{p:k-j:\omega}.\mathbf{J}\_(vk)(vj)(p_v kjp)\right\rangle, \\
&\quad \left\langle \Pi_{k:\omega}.\mathbf{J}\_(uk)(vk)(p_{uv}k),\ \Pi_{p:k-j:\omega}.\mathbf{J}\_(uk)(vj)(p'_{uv}kjp)\right\rangle)
\end{aligned}
$$

Remarque 7.3.1. Les règles de réduction nous offrent une occasion d'observer que dans le cas de $\mathbf{W}$, l'induction est mutuelle, ceci est dû au fait que l'on fait de l'induction sur les graphes. On notera ainsi que le symbole $\mathbf{J}^2$ apparaît dans la réduction des termes $\mathbf{J}\_(\underline{\mathrm{l}}\left\langle u,p_u\right\rangle)$.

Remarque 7.3.2. Le lecteur courageux observera que les règles respectent le typage, ce qui assure la propriété de « subject reduction ».

Remarque 7.3.3. Nous aurions pu mettre en œuvre un opérateur $\mathbf{l}$ inflationnaire à l'aide d'un jugement :

$$\frac{\boldsymbol{\Gamma} \vdash \mathbf{u}^0 : (\boldsymbol{\omega} \multimap \mathbf{W})^0 \qquad \boldsymbol{\Gamma} \vdash t : \mathbb{W} \qquad \boldsymbol{\Gamma},n:\omega \vdash p : \mathbb{W}^2(t,\mathbf{u}^0)}{\boldsymbol{\Gamma} \vdash \mathtt{il}_{[n]}(t,\mathbf{u},p) : \mathbb{W}^2(t,\mathbf{l}^0\mathbf{u}^0)}$$

mais une telle règle aurait compliqué la construction des hiérarchies *monotones* de fonctions ; on peut toujours construire des hiérarchies de fonctions, mais les propriétés d'ordres sont différentes. En particulier, l'ordre ordinaire qui permet de considérer que la diagonale est plus grande que chacune des fonctions de la famille n'est pas l'ordre « pointwise » ordinaire, mais un ordre asymptotique. Informellement, $f \leq g$ s'il existe un entier à partir duquel $fx \leq gx$. Ce qui pourrait se traduire dans notre système par $\Sigma n : \omega.\Pi_{p:n-m}.\omega^2(fm,gm)$. Mais contentons nous pour l'instant de ne considérer qu'un opérateur limite monotone non inflationnaire.

7.3.1 Exemples de termes

Exemple 7.3.4. Nous proposons de construire l'arborescence $\mathbf{L}(\langle n\rangle)_{n\in\omega}$(cf chapitre 5). Nous définissons

$$\omega_{\leq_*} \simeq \underline{\mathrm{l}}(\boldsymbol{\lambda}\mathbf{n}.\mathrm{I}_{[\mathrm{x}]\mathrm{o}(\mathrm{sx})(\mathrm{i}x^2)}\mathbf{n}))$$

dont la preuve du typage est :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{n}:\omega \vdash \mathbf{W}\ \mathtt{grfR}}{\vdash \mathbf{0}:\mathbf{W}} \quad \mathbf{n}:\omega,\mathrm{x}:\mathbf{W} \vdash \mathrm{sx}:\mathbf{W}}{\mathbf{n}:\omega,\mathrm{x}:\mathbf{W} \vdash \mathrm{i}x^2:\mathbf{W}(x^0,\underline{\mathrm{s}}x^1)}}{\mathbf{n}:\omega \vdash \mathrm{I}_{[\mathrm{x}]\mathrm{o}(\mathrm{sx})(\mathrm{i}x^2)}\mathbf{n}:\mathbf{W}}}{\vdash \boldsymbol{\lambda}\mathbf{n}.\mathrm{I}_{[\mathrm{x}]\mathrm{o}(\mathrm{sx})(\mathrm{i}x^2)}\mathbf{n}:\omega \multimap \mathbf{W}}}{\vdash \mathrm{l}(\boldsymbol{\lambda}\mathbf{n}.\mathrm{I}_{[\mathrm{x}]\mathrm{o}(\mathrm{sx})(\mathrm{i}x^2)}\mathbf{n}):\mathbf{W}}$$

En fait, la partie sommet de la consclusion « est » l'arborescence $\omega_{\leq_*}$, la partie arête « est » une preuve que $\omega_{\leq_*} \leq \omega_{\leq_*}$.

Exemple 7.3.5. L'addition et la multiplication se définissent de manière analogue à ce que nous avons vu pour $\boldsymbol{\omega}$. Il suffit de rajouter la prémice qui donne la limite. On utilise pour cela l'opérateur l. Par exemple, pour l'addition, on fait le jugement (à comparer avec celui pour $\boldsymbol{\omega}$) :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{n},\mathbf{m}:\mathbf{W} \vdash \mathbf{W}\ \mathtt{grfR}}{\mathbf{n}:\mathbf{W} \vdash \mathbf{n}:\mathbf{W}} \quad \mathbf{n},\mathbf{m},\mathrm{x}:\mathbf{W} \vdash \mathrm{sx}:\mathbf{W}}{\mathbf{n},\mathbf{m},\mathrm{x}:\mathbf{W} \vdash \mathrm{i}^2x^2\mathbf{W}(x^0,\mathrm{s}^2x^1)}}{\mathbf{n},\mathbf{u}:\boldsymbol{\omega}\multimap\mathbf{W},\mathbf{v}:\boldsymbol{\omega}\multimap\boldsymbol{\omega} \vdash \mathrm{l}\mathbf{v}:\mathbf{W}}}{\mathbf{n},\mathbf{m}:\mathbf{W} \vdash \mathrm{I}_{[\mathrm{x}]\mathbf{n}(\mathrm{sx})(\mathrm{i}^2x^2)}\mathbf{m}:\mathbf{W}}}{\vdash \boldsymbol{\lambda}\mathbf{n}.\boldsymbol{\lambda}\mathbf{m}.\mathrm{I}_{[\mathrm{x}]\mathbf{n}(\mathrm{sx})(\mathrm{i}^2x^2)}\mathbf{m}:\mathbf{W}\multimap\mathbf{W}\multimap\mathbf{W}}$$

On procède de même pour les autres opérations.

Construction des hiérarchies de fonctions

Nous appliquons dans cette section le principe d'induction des arborescences pour obtenir la hiérarchie des fonctions à croissance lente induite par la fonction successeur.

En reprenant la définition de la hiérarchie à croissance lente :

$$\begin{aligned} G_0(x) &= 0 \\ G_{\alpha+1}(x) &= G_\alpha(x)+1 \\ G_\lambda(x) &= G_{\lambda_x}(x) \end{aligned}$$

Nous définissons d'abord le terme :

$$\mathtt{i}_{\omega\multimap\omega} \simeq \left\langle \lambda n^0.\mathtt{i}^2(\pi(f^2)n^0), \lambda n^0.\lambda n^1.\lambda n^2.\mathtt{i}^2(\pi'(f^2)n^0n^1n^2)\right\rangle$$

Nous construisons alors le terme qui représente la hiérarchie à croissance lente :

$$\dfrac{\dfrac{\begin{array}{rcl} \mathbf{v}:\mathbf{W} & \vdash & \omega\multimap\omega \ \mathtt{grfR} \\ & \vdash & \boldsymbol{\lambda}\mathbf{n.0} : \omega\multimap\omega \\ \mathbf{v}:\mathbf{W},\mathbf{f}:\omega\multimap\omega & \vdash & \boldsymbol{\lambda}\mathbf{n.s(fn)} : \omega\multimap\omega \\ \mathbf{v}:\mathbf{W},\mathbf{f}:\omega\multimap\omega & \vdash & \mathtt{i}_{\omega\multimap\omega}v^2 : \omega\multimap\omega(f^0,f^1) \\ \mathbf{u}:\omega\multimap\mathbf{W},\mathbf{v}:\omega\multimap\omega\multimap\omega & \vdash & \boldsymbol{\lambda}\mathbf{n.vnn} : \omega\multimap\omega \end{array}}{\mathbf{v}:\mathbf{W}\vdash \mathbf{J}_{[\mathbf{vf},\mathbf{u}]\boldsymbol{\lambda}\mathbf{n.0}(\boldsymbol{\lambda}\mathbf{n.s(fn)})(\mathtt{i}_{\omega\multimap\omega}v^2)(\boldsymbol{\lambda}\mathbf{n.vnn})}\mathbf{v} : \omega\multimap\omega}}{\vdash \boldsymbol{\lambda}\mathbf{v}:\mathbf{W}.\mathbf{J}_{[\mathbf{vf},\mathbf{u}]\boldsymbol{\lambda}\mathbf{n.0}(\boldsymbol{\lambda}\mathbf{n.s(fn)})(\mathtt{i}_{\omega\multimap\omega}v^2)(\boldsymbol{\lambda}\mathbf{n.vnn})}\mathbf{v} : \mathbf{W}\multimap\omega\multimap\omega}$$

Remarque 7.3.6. Le jugement qui précède contient deux faits. Premièrement, chaque fonction de la hiérarchie est monotone, deuxièmement, si $\alpha < \beta$ sont deux arborescences, alors $G_\alpha < G_\beta$.

En observant la règle,

$$\mathbf{f} : \boldsymbol{\omega} \vdash \left\langle \lambda n^0.(\pi' f^2)(\mathtt{i}^2\mathtt{Ref}_\omega(n^0)), \lambda n^0.\lambda n^1.\lambda n^2.(\pi' f^2)(\mathtt{i}^2 p) \right\rangle : (\boldsymbol{\omega} \multimap \boldsymbol{\omega})^2(f^0, \lambda n.f^1(n))$$

on définit le terme :

$$\mathtt{i}'_{\omega \multimap \omega} \simeq \left\langle \lambda n^0.(\pi' f^2)(\mathtt{i}^2\mathtt{Ref}_\omega(n^0)), \lambda n^0.\lambda n^1.\lambda n^2.(\pi' f^2)(\mathtt{i}^2 p) \right\rangle$$

On peut construire de manière analogue, la hiérarchie de Hardy :

$$\dfrac{\begin{array}{rcl} \mathbf{v} : \mathbf{W} & \vdash & \boldsymbol{\omega} \multimap \boldsymbol{\omega}\ \mathtt{grfR} \\ & \vdash & \boldsymbol{\lambda n.0} : \boldsymbol{\omega} \multimap \boldsymbol{\omega} \\ \mathbf{v} : \mathbf{W}, \mathbf{f} : \boldsymbol{\omega} \multimap \boldsymbol{\omega} & \vdash & \boldsymbol{\lambda n.f(sn)} : \boldsymbol{\omega} \multimap \boldsymbol{\omega} \\ \mathbf{v} : \mathbf{W}, \mathbf{f} : \boldsymbol{\omega} \multimap \boldsymbol{\omega} & \vdash & \mathtt{i}'_{\omega \multimap \omega} v^2 : \boldsymbol{\omega} \multimap \boldsymbol{\omega}(f^0, g f^1) \\ \mathbf{u} : \boldsymbol{\omega} \multimap \mathbf{W}, \mathbf{v} : \boldsymbol{\omega} \multimap \boldsymbol{\omega} \multimap \boldsymbol{\omega} & \vdash & \boldsymbol{\lambda n.vnn} : \boldsymbol{\omega} \multimap \boldsymbol{\omega} \end{array}}{\dfrac{\mathbf{v} : \mathbf{W} \vdash \mathbf{J}_{[\mathbf{vf},\mathbf{u}]\boldsymbol{\lambda n.0}(\boldsymbol{\lambda n.f(sn)})(\mathtt{i}'_{\omega \multimap \omega} v^2)(\boldsymbol{\lambda n.vnn})}\mathbf{v} : \boldsymbol{\omega} \multimap \boldsymbol{\omega}}{\vdash \boldsymbol{\lambda v} : \mathbf{W}.\mathbf{J}_{[\mathbf{vf},\mathbf{u}]\boldsymbol{\lambda n.0}(\boldsymbol{\lambda n.f(sn)})(\mathtt{i}'_{\omega \multimap \omega} v^2)(\boldsymbol{\lambda n.vnn})}\mathbf{v} : \mathbf{W} \multimap \boldsymbol{\omega} \multimap \boldsymbol{\omega}}$$

De même, dans ce cas, nous disposons de la monotonie pour chacune des fonctions de la hiérarchie, mais également de la monotonie relativement aux arborescences.

Conclusions et perspectives

Le problème initial était l'étude de certains ordres, notamment dans le cadre de la réécriture. Nous avons étudié la question sous deux angles différents : celui de l'analyse de la complexité des fonctions calculées par de tels systèmes de réécriture et celui de la construction de hiérarchies de fonctions sous-récursives.

Complexité et réécriture

En ce qui concerne le premier aspect, nous avons montré que de nombreuses méthodes de preuve de terminaison de systèmes de réécriture correspondent à des classes usuelles de complexité. Ceci est intéressant à plusieurs points de vue. Premièrement, cela permet de considérer d'autres modèles de référence en termes de complexité que les machines de Turing. Deuxièmement, cela nous donne une analyse plus fine de la complexité des fonctions calculées par les systèmes de réécriture que la traditionnelle approche en termes de longueur de dérivation. La première remarque nous donne un nouvel objectif qui est d'analyser les algorithmes en utilisant leur preuve de terminaison. Nous avons en effet jusque là considéré uniquement les aspects extensionnels des fonctions, et non les algorithmes qui les calculent. De tels travaux ont été initié par Marion, Cichon et Moyen dans [CM99, Mar99, MM00] pour les ordres MPO et LPO. En particulier, nous ne considérons pas de notion de stratégie dans les réductions ni les problèmes de doublons dans les calculs, ce qui arrive dans l'exemple suivant qui calcule la fonction de Fibonacci :

$$\begin{aligned} F(0) &\rightarrow s(0) \\ F(s(0)) &\rightarrow s(0) \\ F(s(s(n))) &\rightarrow F(n) + F(s(n)) \end{aligned}$$

Si l'on fait le calcul de manière naïve, la dérivation se fait en temps exponentiel alors qu'il existe un algorithme polynomial. Or, si l'on veut utiliser la réécriture comme une implantation de spécifications algébriques, il est indispensable du point de vue de la complexité de pouvoir reconnaître de telles situations. Les travaux de Marion, Cichon et Moyen sont encourageants pour continuer dans cette voie.

Un calcul monotone

La deuxième partie de notre étude a aboutit à la proposition d'un calcul, une extension du système T de Gödel, dans lequel on peut traiter des fonctions en termes de monotonie. L'objectif était de construire un système utilisant des moyens les plus élémentaires possibles. Les constructions se font moralement dans l'univers des graphes, et deux jugements sont nécessaires l'un pour les sommets, l'autre pour les arêtes. Nous avons proposé un système d'abréviations qui permet de simplifier de manière significative la manipulation du calcul. Il s'agit d'une nouvelle technologie dans l'univers du λ-calcul.

D'un point de vue sémantique, nous avons élaboré, à l'intérieur de la structure monoïdale sur les graphes (que nous présentons au chapitre 6), une notion d'algèbre universelle qui permet de recouvrir de nombreux ordres usuels. En particulier, l'objectif était de pouvoir représenter l'ordre sur les termes ordinaux, ce qui est possible comme nous l'avons montré.

Dans le calcul, nous considérons une nouvelle forme d'induction, nous faisons en effet de l'induction sur des graphes et non sur des ensembles ; cette nouvelle sorte d'induction peut être observée dans les règles d'élimination qui ont deux conclusions, les deux composantes d'un « morphisme » de graphe.

L'étude de ce calcul donne lieu à de nombreuses perspectives. Notons en particulier, que jusqu'à présent, nous n'avons pas considéré le

calcul en termes de réduction [35]. Nous n'avons pas établi pour l'instant de théorème de normalisation forte ; la difficulté pour obtenir un tel théorème n'est pas de nature logique (le système est prédicatif avec une sémantique ensembliste), mais provient de la complexité notationnelle du calcul fin. D'autre part, le mécanisme de substitution dans le calcul gras reste à développer (la substitution naïve ne convient pas à cet effet). Ceci nous permettrait de calculer directement sur les graphes, sans référence aux ensembles sous-jacents ; ceci nous donnerait également la possibilité de simplifier les règles de réduction des termes obtenus par induction.

À propos des deux systèmes λ-ω et λ-$\mathbb{W}$, nous n'avons pas réussi à montrer à l'intérieur de la syntaxe la transitivité. Nous faisons même la conjecture que cela n'est pas possible. Cela témoigne de la « faiblesse » du système. Mais en même temps, nous avons moralement la transitivité via le théorème de transitivité sémantique. Obtenir la transitivité à l'intérieur du système ne nécessite pas de principe logique très puissant, mais il est sans doute nécessaire d'introduire une construction supplémentaire pour manipuler les arêtes dans les graphes.

Enfin, nous jugeons que la technologie développée dépasse les deux exemples de λ-ω et λ-$\mathbb{W}$. Le passage au cas général d'une Ψ-algèbre peut être obtenu en prenant exemple sur le calcul λ-$\mathbb{W}$ qui réunit une arité à la fois infinie (ce qui nécessite de définir l'arité à une étape antérieure) et non discrète (ce qui fait que l'induction se fait mutuellement sur les sommets et les arêtes).

Mais en fait, nous pensons que la technologie dépasse même la notion de Ψ-algèbre. Il semble en effet possible de définir des opérations comme les bornes supérieures de ω-chaînes par exemple.

35. Même si nous avons orienté les règles.

Bibliographie

[Alt89] E. Altendorf. *Termination von Termersetzungssystemen: Theorische Untersuchungen und Implementierungen von KBO, RPO und KNS.* PhD thesis, TU Berlin, 1989.

[Bar84] H. P. Barendregt. *The Lambda Calculus.* Studies in Logic and the Foundations of Mathematics. Elsevier Science, 1984.

[BCL87] A. Ben Cherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137–160, October 1987.

[BCMT98] G. Bonfante, A. Cichon, J.-Y. Marion, and H. Touzet. Complexity classes and rewrite systems with polynomial interpretation. In *CSL'98, Brno, République Tchèque*, 1998.

[BCMT99] G. Bonfante, A. Cichon, J.-Y. Marion, and H. Touzet. Algorithms with polynomial interpretation termination proof. *Journal of Functional Programming*, 1999. À paraître.

[Ber70] C. Berge. *Graphe et Hypergraphe.* Dunod, 1970.

[Ber78] G. Berry. Stable models of typed λ-calculi. In Springer-Verlag, editor, *Fifth International Colloquium on Automata, Langages and Programs*, volume 62 of *Lecture Notes in Computer Science*, pages 72–89, 1978.

[Bon00] G. Bonfante. Complexity characterisation of restrictions

of KBO. Présenté au Workshop *Implicit Computational Complexity*, 2000.

[BW90] M. Barr and C. Wells. *Category theory for computing science*. Prentice Hall, 1990.

[Cic90] E.A. Cichon. Bounds on derivation lengths from termination proofs. Technical report, University of London, Royal Holloway and Bedford New College, 1990.

[CL92] A. Cichon and P. Lescanne. Polynomial interpretations and the complexity of algorithms. In *CADE '11*, number 607 in Springer Lecture Notes in Artificial Intelligence, 1992.

[CM99] A. Cichon and J.-Y. Marion. The light lexicographic path ordering. Communication personnelle, 1999.

[Cob62] A. Cobham. The intrinsic computational difficulty of functions. In Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Sciences*, pages 24–30. North Holland, 1962.

[CP89] T. Coquand and C. Paulin. Inductively defined types. In *Proceedings of the Workshop on Programming Logic*, Båstadt, 1989.

[CR36] A. Church and J. B. Rosser. Some properties of conversion. *Trans. Amer. Math. Soc.*, 40, 1936.

[Csi47] P. Csillag. Eine Bemerkung zur Auflösung der eingeschachtelten Rekursion. *Acta Sci. Math. Szeged*, 11:169–73, 1947.

[CT96] A. Cichon and H. Touzet. An ordinal calculus for proving termination in term rewriting. In H. Kirchner, editor, *Int. CAAP-Coll.on Trees in Algebra and Programming*, volume 1059 of *Lecture Notes in Computer Science*, pages 226–240. Springer-Verlag, 1996.

[CTB98] A. Cichon and E. Tahhan-Bittar. Ordinal recursive bounds for Higman's theorem. *Theoretical Computer Science*, 201(1-2):63–84, July 1998.

[CW83] A. Cichon and S. Wainer. The slow-growing and the grzegorczyk hierarchies. *Journal of Symbolic Logic*, 48(2):399–408, 1983.

[Dan99] N. Danner. *Ordinal notations in typed λ-calculi.* PhD thesis, Indiana University, 1999.

[Dav58] M. Davis. *Computability and Unsolvability.* McGraw-Hill Book Company, 1958.

[Dav82] M. Davis. Why didn't Gödel have Church's thesis. *Information and Control*, 54:3–24, 1982.

[dB72] N. de Bruijn. Lambda calculus with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. In *Proc. Koninkl. Nederl. Akademie van Wetenschappen*, volume 75, pages 381–392, 1972.

[Der82] N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.

[dG93] P. de Groote. The conservation theorem revisited. In Springer, editor, *International Conference on Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 163–178, 1993.

[DJ90] N. Dershowitz and J.-P. Jouannaud. *Rewrite systems.* Handbook of Theoretical Computer Science vol.B, northholland edition, 1990.

[DJW83] E. Dennis-Jones and S. Wainer. Subrecursive Hierarchies via Direct Limits. In Richter, Borger, Obershelp, Schinzell, and Thomas, editors, *Logic Colloquium 83*, pages 117–28. Springer Lecture Notes, 1983.

[DS99] P. Dybjer and A. Setzer. A finite axiomatization of inductive-recursive definitions. *Proceedings of TLCA 99, SLNCS*, 1999.

[Dyb91] P. Dybjer. Inductive sets and families in martin-löf's type theory and their set-theoretic semantics. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 280–306. Prentice Hall, 1991.

[Dyb97] P. Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *To appear in the Journal of Symbolic Logic*, 1997.

[Geu88] O. Geupel. Terminationbeweise bei termersetzungssystem. Diplomarbeit, 1988.

[GG95] E. Grädel and Y. Gurevich. Tailoring recursion for complexity. *Journal of symbolic logic*, 60(3):952–69, sep 1995.

[Gie95a] J. Giesl. Generating polynomial orderings for termination proofs. In Jieh Hsiang, editor, *Proceedings 6th Conference on Rewriting Techniques and Applications, Kaiserslautern (Germany)*, volume 914 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[Gie95b] J. Giesl. Polo - a system for termination proofs using polynomial orderings. Technical report, Technische Hochschule Darmstadt, Alexanderstr. 10, 64283 Darmstadt Germany, 1995.

[Gir81] J.-Y. Girard. Π^1_2-logic. *Annals of Mathematical Logic*, pages 75–219, 1981.

[Gir86] J.-Y. Girard. The system F of variable types, fifteen years later. *Theoretical computer science*, 45, 1986.

[Grz53] Grzegorczyk. Some classes of recursive functions. *Rozprawy Mate*, 4, 1953. Warsaw.

[GU71] P. Gabriel and F. Ulmer. *Lokal Präsentierbare Kategorien.* Springer Lecture notes in Mathematics 221, 1971.

[Gun92] C. A. Gunter. *Semantics of Programming Languages.* The MIT Press, 1992.

[Gur99] Y. Gurevich. The sequential asm thesis. *Bulletin of European Association for Theoretical Computer Science*, 67:93–124, 1999.

[Hay85] S. Hayashi. Adjunction of semi-functors: categorical structures in nonextensional lambda calculus. *Theoretical Computer Science*, 41:95–104, 1985.

[HL78] G. Huet and D. S. Lankford. On the uniform halting problem for term rewriting systems. Technical Report 283, Laboria, France, 1978.

[HL89] D. Hofbauer and C. Lautemann. Termination Proofs and the Length of Derivation. In *3rd RTA*, volume 355 of *Lecture Notes in Computer Science*, pages 167–177, 1989.

[HO80] G. Huet and D. Oppen. Equations and rewrite rules: A survey. In R. V. Book, editor, *Formal Language*

Theory: Perspectives and Open Problems, pages 349–405. AP, 1980.

[Hof92a] D. Hofbauer. *Termination Proofs and Derivation Lengths in Term Rewriting Systems.* PhD thesis, Technische Universität Berlin, 1992.

[Hof92b] D. Hofbauer. Termination proofs with multiset path orderings imply primitive recursive derivation lengths. *Theoretical Computer Science*, 105(1):129–140, 1992.

[Hoo93] Hoofman, R. Continuous information systems. *Math. Struct. Comput. Sci. 3, No.1,*, 1:93–128, 1993.

[HS86] J. R. Hindley and J. P. Seldin. *Introduction to Combinators and λ -Calculus*, volume 1 of *London Mathematical Society Student Texts.* Cambridge University Press, 1986.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, langages, and computation.* Addison Wesley, 1979.

[HW97] W. Handley and S. Wainer. Complexity of primitive recursion. In U. Berger and Schwichtenberg. H., editors, *Computational Logic*, volume 165 of *Computer and Science systems*, pages 273–300. Université de Munich, Springer, 1997. proceedings de l'école d'été tenue à Marktoberdorf.

[Jac94] B. Jacobs. Semantics of weakening and contraction. *Annals of pure and applied logic*, 69:73–106, 1994.

[JL86] J.-P. Jouannaud and P. Lescanne. La réécriture. *TSI*, 5(6):433–452, 1986.

[Kal43] L. Kalmar. Egyszerü példa eldönthetetlen aritmetikai problémára. *Mate és Fizikai Lapok*, 1943. Le résumé est en allemand.

[KB70] D.E. Knuth and P.B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational problems in abstract algebra*, Pergamon, 1970.

[Kel82] G. M. Kelly. *Basic Concepts of Enriched Category Theory.* LMS Lecture Note Series 64, Cambridge University Press, 1982.

[Kle36] S. C. Kleene. General recursive functions of natural numbers. *Math. Ann.*, 112:727–42, 1936.

[Kle52] S. C. Kleene. *Introduction to Metamathematics.* D. Van Nostrand Company Inc., Princeton, 1952.

[KP93] G. M. Kelly and J. Power. Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads. *J. Pure and App. Algebra*, 89:163–179, 1993.

[Kre88] M. Krentel. The complexity of optimization problems. *Journal of computer and system sciences*, 36:490–519, 1988.

[Lan79] D.S. Lankford. On proving term rewriting systems are nœtherian. Technical report, Louisiana Technical University, 1979.

[Lar] Larch. http://www.sds.lcs.mit.edu/spd/larch/index.html.

[Law64] F. W. Lawvere. An elementary theory of the category of sets. *Proc. Nat. Acad. Sci. USA*, 52:869–72, 1964.

[Law73] F. W. Lawvere. Metric spaces, generalized logics, and closed categories. *Seminario Matematico e Fisico di Milano*, 43:135–166, 1973.

[Lep00] I. Lepper. The computational strength of the Knuth-Bendix Ordering. Communication personnelle, 2000.

[LS86] J. Lambek and P.J. Scott. *Introduction to higher order categorical logic.* Cambridge University Press, 1986.

[LW70a] M.H. Löb and S. Wainer. Hierarchies of number-theoretic functions. part I. *Arch. Math. Logik*, 13:39–51, 1970.

[LW70b] M.H. Löb and S. Wainer. Hierarchies of number-theoretic functions. part II. *Arch. Math. Logik*, 13:97–113, 1970.

[LW71] M.H. Löb and S. Wainer. Hierarchies of number-theoretic functions. correction. *Arch. Math. Logik*, 14:198, 1971.

[Mai77] D. Maier. The complexity of some problems on subsequences and supersequences. *J. Assoc. Comput. Mach.*, 25:322–336, 1977.

[Mar87] U. Martin. How to chose the weights in the Knuth-Bendix ordering. In *2nd RTA*, volume 256 of *Lecture Notes in Computer Science*, pages 42–53, 1987.

[Mar99] J.-Y. Marion. Light MPO, two is better than one. In *first workshop on Implicit Computational Complexity*, 1999.

[Men64] E. Mendelson. *Introduction to Mathematical Logic.* The University Series in Undergraduate Mathematics. D. Van Nostrand Company, Inc., Princeton, New Jersey, 1964. First published December 1963. Reprinted January 1965, December 1965, July 1966, August 1968.

[Men91] N. Mendler. Predicative type universes and primitive recursion. In *Sixth annual IEEE symposium on Logic In Computer Science*, pages 173–184, 1991.

[ML91] S. Mac Lane. *Categories for the working mathematician.* Springer, 1991.

[MM00] J.-Y. Marion and J.-Y. Moyen. Efficient first order functional program interpretation with time bound certifications. Manuscript, 2000.

[Par66] Ch. Parsons. Ordinal recursion in partial systems of number theory. *Notices American Mathematical Society*, 13:857–58, 1966.

[Pea88] G. Peano. *Arithmetices principia, nova methodo exposita.* Turin, 1888. Traduction anglaise chez van Heijnoort, 1967.

[Pit89] Pitts, A. M. and Taylor, P. A note on Russell's paradox in locally Cartesian closed categories. *Stud. Logica 48*, 1989.

[PM93] C. Paulin-Mohring. Inductive definitions in the system Coq; rules and properties. In *Typed λ-calculus and Applications*, Lecture Notes in Computer Science, pages 328–45, 1993.

[Poi03] H. Poincaré. *La science et l'hypothèse.* Flamarion, 1903.

[Pos36] E. Post. Finite combinatory processes, formulation i. *Journal of symbolic logic*, 1, 1936.

[PS89] K. Petersson and D. Synek. A set constructor for inductive sets in Martin-Löf's type theory. *Springer Lecture Notes in Mathematics*, 389:128–40, 1989.

[Pét67] R. Péter. *Recursive Functions.* Academic Press, 1967.

[Rit63] R.W. Ritchie. Classes of predictably computable functions. *Trans. A. M. S.*, 106:139–173, 1963.

[Rob] J. Robbin. *Subrecursive Hierarchies.* PhD thesis, Princeton.

[Ros84] H.E. Rose. *Subrecursion.* Clarendon Press (Oxford University Press), 1984.

[RV99] A. Ryazanov and A. Voronkov. Vampire. In 16^{th} *International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, pages 292–296, 1999.

[Sav70] W.J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[Sim00] H. Simmons. Logic and computation. Cambridge University Press, 2000.

[Tai67] W. W. Tait. Intentional interpretation of functionals of type I. *Journ. of symbolic logic*, 32:198–212, 1967.

[Tou97] H. Touzet. *Propriétés combinatoires pour la terminaison de systèmes de réécriture.* PhD thesis, Université Henri Poincaré - Nancy 1, September 1997.

[Tur36] A. M. Turing. On computable numbers with an application to the Entscheindungsproblem. In *Proc. London Math. Soc.*, volume 42, pages 230–265, 1936.

[Wai72] S.S. Wainer. Ordinal recursion and a refinement of the extended Grzegorczyk hierarchy. *J.S.L.*, 37:281–292, 1972.

[Wei95] A. Weiermann. Termination proofs by lexicographic path orderings yield multiply recursive derivation lengths. *Theoretical Computer Science*, 139(1):355–362, 1995.

[Wei99] C. Weidenbach. *Handbook of Automated Reasoning*, chapter Combining superposition, sorts and splitting. Elsevier Science and MIT Press, 1999. To appear.

Index

Notations

Résumé

Le thème de la thèse concerne l'étude des ordres, en termes de complexité en réécriture, mais aussi dans le cadre des termes ordinaux, des hiérarchies de fonctions sous-récursives.

Nous avons cherché à extraire de la notion de preuve de terminaison des informations concernant la complexité des fonctions calculées par des systèmes de réécriture. Cela nous a permis d'établir de nombreuses caractérisations, en termes de complexité, des systèmes de réécriture. Ainsi, est-il possible de déterminer *a priori* la complexité d'un système, au seul vu de sa preuve de terminaison. Nous étudions plus spécifiquement le cas de l'ordre de Knuth-Bendix et celui de l'ordre par interprétation polynomiale.

D'autre part, nous avons entamé une recherche plus fondamentale concernant la représentation de la notion d'ordre dans les λ-calculs typés. Cet intérêt provient de ce qu'une telle représentation est nécessaire dès lors que l'on veut représenter des structures définies par des opérateurs ω-aire, ce qui est le cas des termes ordinaux par exemple. Nous étudions d'un point de vue sémantique l'univers de travail, nous dégageons en particulier une notion de structure monoïdale fermée, mais aussi de manière syntaxique en proposant un calcul pour lequel un type représente un graphe plutôt qu'un (traditionnel) ensemble.

Mots-clés: Complexité, Réécriture, Ordres, λ-calculs typé, Sémantique, Syntaxe

Abstract

Our concern in the thesis is in the study of orders, through the analysis of complexity within rewriting, but also through the construction of monotonous λ-calculi within the framework of ordinal terms, and subrecursive hierarchies.

We show how to extract some knowledge in terms of complexity from the proof of termination in rewriting theory. This allows us to establish some hierarchies of caracterisation of complexity classes in terms of rewriting. So, one has an *a priori* complexity measure of functions with respect to their proof of termination. We study more specifically the case of the Knuth-Bendix ordering and polynomial interpretations.

Next to that, we engaged ourselve into an other challenge, a more fundamental approach. This concerns the notion of ordering within typed λ-calculi. The interest comes from the fact that such a representation is necessary when one has to represent a structure with an ω-ary operation, that is an operation that only applies to monotonous sequences (which is the case of ordinal terms). We develop semantical tools, in particular we present a monoidal close category on graphs, but also syntactical constructions, in particular a calculus where types are graphs in which the traditional approach would be in terms of sets.

Keywords: Complexity, Rewriting, Orderings, Typed λ-calculi, Semantics, Syntax

INSTITUT NATIONA
POLYTECHNIQU
DE LORRAIN

AUTORISATION DE SOUTENANCE DE THESE
DU DOCTORAT DE L'INSTITUT NATIONAL
POLYTECHNIQUE DE LORRAINE

o0o

VU LES RAPPORTS ETABLIS PAR :
Monsieur JOUANNAUD Jean-Pierre, Professeur, LRI/Université de Paris-Sud-Orsay,
Monsieur DYBJER Peter, Professeur, Chalmers University of Technology Göteborg (Suède),
Monsieur JARAY Jacques, Maître de Conférence, LORIA/Ecole des Mines de Nancy.

Le Président de l'Institut National Polytechnique de Lorraine, autorise :

Monsieur BONFANTE Guillaume

NANCY BRABOIS
2, AVENUE DE LA
FORET-DE-HAYE
BOITE POSTALE 3
F - 5 4 5 0 1
VANDŒUVRE CEDEX

à soutenir devant un jury de l'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE, une thèse intitulée :

"Constructions d'ordres, analyse de la complexité".

en vue de l'obtention du titre de :

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

Spécialité : **"INFORMATIQUE"**

Fait à Vandoeuvre le, **19 octobre 2000**

Le Président de l'I.N.P.L.,

J. HARDY

TEL. 33/03.83.59.59.59
FAX. 33/03.83.59.59.55

Résumé

Le thème de la thèse concerne l'étude des ordres, en termes de complexité en réécriture, mais aussi dans le cadre des termes ordinaux, des hiérarchies de fonctions sous-récursives.

Nous avons cherché à extraire de la notion de preuve de terminaison des informations concernant la complexité des fonctions calculées par des systèmes de réécriture. Cela nous a permis d'établir de nombreuses caractérisations, en termes de complexité, des systèmes de réécriture. Ainsi, est-il possible de déterminer *a priori* la complexité d'un système, au seul vu de sa preuve de terminaison. Nous étudions plus spécifiquement le cas de l'ordre de Knuth-Bendix et celui de l'ordre par interprétation polynomiale.

D'autre part, nous avons entamé une recherche plus fondamentale concernant la représentation de la notion d'ordre dans les λ-calculs typés. Cet intérêt provient de ce qu'une telle représentation est nécessaire dès lors que l'on veut représenter des structures définies par des opérateurs ω-aire, ce qui est le cas des termes ordinaux par exemple. Nous étudions d'un point de vue sémantique l'univers de travail, nous dégageons en particulier une notion de structure monoïdale fermée, mais aussi de manière syntaxique en proposant un calcul pour lequel un type représente un graphe plutôt qu'un (traditionnel) ensemble.

Mots-clés: Complexité, Réécriture, Ordres, λ-calculs typé, Sémantique, Syntaxe

Abstract

Our concern in the thesis is in the study of orders, through the analysis of complexity within rewriting, but also through the construction of monotonous λ-calculi within the framework of ordinal terms, and subrecursive hierarchies.

We show how to extract some knowledge in terms of complexity from the proof of termination in rewriting theory. This allows us to establish some hierarchies of caracterisation of complexity classes in terms of rewriting. So, one has an *a priori* complexity measure of functions with respect to their proof of termination. We study more specifically the case of the Knuth-Bendix ordering and polynomial interpretations.

Next to that, we engaged ourselve into an other challenge, a more fundamental approach. This concerns the notion of ordering within typed λ-calculi. The interest comes from the fact that such a representation is necessary when one has to represent a structure with an ω-ary operation, that is an operation that only applies to monotonous sequences (which is the case of ordinal terms). We develop semantical tools, in particular we present a monoidal close category on graphs, but also syntactical constructions, in particular a calculus where types are graphs in which the traditional approach would be in terms of sets.

Keywords: Complexity, Rewriting, Orderings, Typed λ-calculi, Semantics, Syntax