Aspects mathématiques des langages de programmation

Syntaxe et sémantique

Adrien Guatto
guatto@irif.fr

22 janvier 2024

Avant-propos

Ces notes proposent une introduction à la théorie des langages de programmation. Elles sont en cours de rédaction, et contiennent indubitablement leur lot de fautes et d'erreurs plus ou moins graves. Merci de signaler celles-ci à *guatto@irif.fr*. Merci également de ne pas redistribuer le document.

1. Introduction

Ces notes visent à fournir une introduction à la théorie des langages de programmation. Elles prennent donc le sujet à son début. Toutefois, leur objectif est d'offrir un chemin aussi direct que possible aux questions de recherche contemporaines. Comme beaucoup de ces recherches, elles adoptent le point de vue de l'informatique mathématique. On y étudiera donc des langages de programmation très idéalisés, épurés de leurs caractéristiques contingentes. Cette idéalisation éloigne nécessairement des aspects les plus pratiques de l'activité de programmation. En contrepartie, elle donne accès à toute la précision et à toute la rigueur permises par les mathématiques.

Un langage de programmation est un langage symbolique dont les phrases sont obtenues par l'application répétée d'une collection d'opérations formelles fixée. Ses phrases, appelées programmes, sont équipés d'une mécanique d'exécution plus ou moins abstraite qui définit leur sens. Ce sens contribue notamment à déterminer quels programmes doivent être considérés comme équivalents. L'étude mathématique des langages de programmation se rapproche donc de l'algèbre, à ceci près que les opérations offertes par un langage de programmation prennent des formes plus riches que celles mises en jeu lors de la définition de la structure de groupe, d'anneau, de corps et des autres objets traditionnels du cours d'algèbre. La perspective adoptée au sein des présentes notes est donc celle de l'algèbre universelle, sous-discipline des mathématiques dédiée à l'étude du format général des structures algébriques plutôt qu'à celle d'une famille de structures algébriques particulière.

Le contenu des notes est découpé en trois parties. La première partie constitue une introduction au pendant le plus informatique de l'algèbre universelle classique conçue pour être compatible avec les développement ultérieurs. La seconde partie traite du système T, un prototype de langage fonctionnel dont tous les programmes terminent. La troisième partie traite du langage PCF, un langage fonctionnel dont les programmes peuvent diverger.

Ces notes s'adressent à des étudiants de quatrième ou cinquième année d'université ayant suivi un cours de logique mathémathique de premier cycle. La connaissance d'un langage de programmation fonctionnel est également la bienvenue. Les questions plus informatiques, notamment d'implémentation, seront mentionnées lorsqu'elles sont susceptibles d'éclairer les développements mathématiques. Nous espérons ainsi aider lectrices et lecteurs informaticiens à s'appuyer sur leurs intuitions pratiques.

Première partie

Généralités mathématiques

2. Rappels de logique

L'objectif de ce chapitre est de rappeler dans un style informel les opérations de base nécessaires à la construction des mathématiques, telles que nécessaires dans les chapitres suivants. Il n'a pas pour ambition de se substituer à un cours de logique formelle ou de fondements des mathématiques, bien que ces sujets puissent être abordés d'une manière étonamment proche de l'étude des langages de programmation.

2.1. Le langage informel de la logique et des preuves

La définition et la manipulation des objets mathématiques passe par l'énonciation de *propositions*. D'un point de vue formaliste, une proposition est un énoncé écrite dans un langage symbolique complètement codifié. La grammaire du langage détermine les propositions de base, ainsi que les règles permettant de construire de nouvelles propositions. Ces règles de construction sont les *connecteurs* logiques bien connus, comme la conjonction. Elles sont gouvernées par d'autres règles qui caractérisent la forme des preuves valides d'une proposition donnée.

Au delà de son contenu logique, une proposition porte sur les objets qui intéressent le mathématicien. Ces objets peuvent être de différentes *sortes*. Ils sont soumis à certaines *opérations* et *prédicats* logiques. Les opérations et prédicats sont liées par une collection de propositions admises appelées *axiomes*. Un système de sortes, opérations, relations et axiomes forme une *théorie axiomatique*. Par exemple, la géométrie euclidienne peut être présentée comme une théorie axiomatique où l'on manipule des points, des droites et des plans liés par les relations d'inclusion, d'incidence et de congruence. Ses axiomes expriment par exemple que pour tous deux points il existe une droite qui les contient tous les deux.

Depuis la première moitié du XXème siècle, il semble que la vaste majorité des mathématiciens aient adopté une théorie axiomatique particulière, la théorie de Zermelo-Fraenkel avec axiome du choix, en abrégé ZFC. Cette théorie s'est révélée très expressive, au sens où la plupart des objets mathématiques usuels peuvent être construits en utilisant les objets primitifs de ZFC. Les objets de ZFC sont des “ensembles”, et certains axiomes de cette théorie reflètent effectivement nos intuitions pré-mathématiques vis-à-vis des collections de choses rencontrées dans la vie ordinaire. D'autres axiomes sont toutefois d'une signification nettement moins intuitive. Par analogie avec l'informatique, on peut donc penser à ZFC comme à une théorie “de bas niveau” : à l'image du langage machine des processeurs, c'est un langage difficile à manier mais au dessus duquel il est possible de bâtir les abstractions qui constitueront des couches de plus haut niveau. Comme en informatique, l'existence d'une couche de bas niveau partagée assure la compatibilité des couches supérieures ; ainsi, un théorème d'analyse peut être utilisé en arithmétique

sans difficulté si besoin est, sans modification de sa preuve. Cette approche garantit la cohérence des mathématiques, non pas au sens logique de l'absence de contradiction, mais au sens où les développements au sein d'un certain pan de mathématiques sont directement utilisable dans d'autres, puisque tous sont ultimement traduisibles dans le langage de ZFC, sans changer de théorie axiomatique.

Le reste de ce chapitre contient un rappel des règles élémentaires de la logique, ainsi que des constructions ensemblistes usuelles. Ces dernières sont présentées dans un style informel, les règles précises de la théorie des ensembles de Zermelo-Fraenkel dépassant de loin le cadre de ces notes.

2.2. Propositions et prédicats

L'objectif de cette section est de faciliter la rédaction des démonstrations, et de clarifier les règles autorisées pour l'étudiant qui manquerait de bagage mathématique. Pour ce faire, on va adopter un point de vue très mécanique, et donc très détaillée, de l'activité démonstratoire, ramenée à un petit jeu de règles élémentaires. Il est évident qu'une preuve lisible n'explicite pas l'utilisation de chacune des règles, mais cherche plutôt à mettre en valeur les étapes essentielles. Toutefois, avoir en tête l'existence de ces règles, en particulier lorsqu'on est peu familier avec la pratique des mathématiques, peut aider à rédiger et surtout à éviter les erreurs de raisonnement.

À tout moment durant une preuve, le mathématicien cherche à démontrer un *but* formé d'une proposition appelée *conclusion* et d'un nombre fini de propositions présupposées appellées *hypothèses*. Les règles qui suivent permettent de progresser en remplaçant le but courant par une collection finie de nouveaux buts. Éventuellement, on atteint un point où cette collection de nouveaux buts est vide, ce qui signifie que la preuve est terminée. Les règles sont de plusieurs natures : certaines sont purement structurelles

2.2.1. Règles structurelles

Ces règles sont indépendantes du choix des connecteurs, et décrivent la gestion des hypothèses. Elles sont au nombre de deux. La première stipule qu'un but φ est immédiatement démontrable si φ apparaît également comme hypothèse. La seconde stipule qu'il est toujours possible d'introduire une nouvelle hypothèse φ pour prouver une conclusion ψ. On doit pour cela commencer par prouver φ sous la même liste d'hypothèse.

2.2.2. Connecteurs

Pour chaque connecteur, on donne les règles de démonstration qui permettent de l'*introduire*, c'est-à-dire de prouver un but dont il forme la racine de la conclusion, et de l'*éliminer*, c'est-à-dire d'utiliser une hypothèse dont il forme la racine.

Conjonction.

Étant donnée deux propositions φ et ψ, on peut former leur *conjonction*, notée $\varphi \wedge \psi$. Pour prouver $\varphi \wedge \psi$, il suffit de prouver φ et ψ. Pour prouver la conclusion τ en utilisant l'hypothèse $\varphi \wedge \psi$, on prouve la conclusion τ sous les hypothèses additionnelles φ et ψ.

Disjonction.

Étant donnée deux propositions φ et ψ, on peut former leur *disjonction*, notée $\varphi \vee \psi$. Pour prouver $\varphi \vee \psi$, il suffit de prouver φ, ou bien de prouver ψ. Pour prouver la conclusion τ en utilisant l'hypothèse $\varphi \vee \psi$, on prouve indépendamment la conclusion τ sous l'hypothèse φ et sous l'hypothèse ψ.

Implication.

Étant donnée deux propositions φ et ψ, on peut former leur *implication*, notée $\varphi \implies \psi$. Pour prouver $\varphi \implies \psi$, il suffit de prouver ψ sous l'hypothèse φ. Pour prouver le but τ en utilisant l'hypothèse $\varphi \wedge \psi$, on prouve le but φ sous les mêmes hypothèses, puis on prouve τ sous l'hypothèse additionnelle ψ.

Trivialité.

La proposition *triviale* est notée $\top$. Prouver le but $\top$ est immédiat. Aucune règle ne permet d'utiliser une hypothèse $\top$.

Contradiction.

La proposition *contradictoire* est notée $\bot$. Aucune règle ne permet de l'introduire. Tout but τ est prouvé immédiatement sous l'hypothèse $\bot$.

Remarque 1. Une théorie axiomatique qui démontre $\bot$ sans hypothèse est dite *contradictoire*. Dans ce cas, la règle citée ci-dessus entraîne immédiatement que la théorie démontre toute proposition.

Négation.

La *négation* d'une formule φ est notée $\neg\varphi$. La négation n'est pas un connecteur primitif mais est définie par $\neg\varphi \overset{def}{\iff} \varphi \implies \bot$. L'avantage de cette approche est qu'elle ne nécessite pas d'introduire de nouvelles règles. On peut dériver les règles suivantes à partir de cette définition. Pour introduire $\neg\varphi$, il faut démontrer $\bot$ sous l'hypothèse additionnelle φ. Pour prouver le but ψ en présence de l'hypothèse $\neg\varphi$, il suffit de démontrer la conclusion φ.

2.2.3. Le tiers exclu

Enfin, on ajoute aux règles ci-dessus l'axiome suivant, appelé *tiers exclu* :

$$\text{pour toute proposition } \varphi, \text{ on a } \varphi \vee \neg\varphi.$$

En la présence de cet axiome, appelé *tiers exclu*, on peut dériver les résultats bien connus de logique classique, notamment l'involutivité de la négation

$$\neg\neg\varphi \quad \iff \quad \varphi,$$

la formule bien connue de l'implication

$$\varphi \implies \psi \quad \iff \quad \neg\varphi \vee \psi,$$

ainsi que les lois de distributivité dites de *de Morgan* reproduites ci-dessous.

$$\neg(\varphi \wedge \psi) \iff \neg\varphi \vee \neg\psi \qquad \neg(\varphi \vee \psi) \iff \neg\varphi \wedge \neg\psi \qquad \neg\top \iff \bot \qquad \neg\bot \iff \top$$

L'involutivité de la négation permet notamment le *raisonnement par l'absurde*. Celui-ci prend la forme de la règle de preuve suivante : pour prouver une conclusion φ, il suffit de supposer $\neg\varphi$ et de dériver une contradiction, c'est-à-dire de prouver la conclusion $\bot$.

2.2.4. Prédicats

La logique pure telle que décrite précédemment ne suffit pas pour construire les mathématiques. On lui ajoute donc une notion d'*objets* définie axiomatiquement par un jeu d'opérations et de prédicats, et sur lesquels il est possible de quantifier logiquement. Les opérations et prédicats autorisés dépendent de la théorie axiomatique adoptée. Dans ces notes, comme il est d'usage en mathématiques, on adoptera une théorie des ensembles informelle. Nos objets seront donc des ensembles notés A, B, C et parfois t, u, v.

Égalité.

Le prédicat d'égalité entre deux individus, noté $=$, est le plus basique de tous, et joue un rôle un peu particulier. La seule façon d'introduire ce prédicat est via la proposition $A = A$, prouvable sous n'importe quelles hypothèses. On peut l'utiliser via une règle logique ad-hoc appelée *transport* : dès lors qu'on dispose d'une hypothèse de la forme $A = B$, alors on peut remplacer A par B et B par A n'importe où dans le but courant, que ce soit dans les hypothèses ou la conclusion. En particulier, on peut dériver la symétrie et la transitivité de l'égalité à partir de la règle de transport.

Appartenance.

La théorie des ensembles est une théorie axiomatique dénuée d'opérations et munie d'un seul prédicat, hormis l'égalité. Il s'agit du prédicat binaire appelé *appartenance* et noté $t \in A$. Intuitivement, il signifie que t est un élément de A. Ce prédicat est soumis à un certain nombre d'axiomes qui permettent de manipuler efficacement la notion d'ensemble, et qui ont permis de bâtir le reste des mathématiques. On décrit un certain nombre d'entre eux plus bas.

Remarque 2. Le prédicat d'appartenance relie des objets de même nature, c'est-à-dire que t et A sont tous deux des ensembles. Autrement dit, les éléments d'un ensemble sont d'autres ensembles. Ainsi, en théorie des ensembles, l'ensemble des entiers naturels, le nombre pi, la droite réelle, ou l'entier 42 sont tous des ensembles. Remarquons la similarité avec les langages informatiques de bas niveau où toutes les données manipulées sont vues comme des suites de bits. Cette économie de concepts a la conséquence contre-intuitive de permettre l'expression de prédicats dénués de sens intuitifs, comme par exemple $\mathbb{N} \in \pi$. Il est même possible que ces prédicats soient valides, selon le codage employé.

Quantification.

Étant donné un ensemble A et une proposition $\varphi(x)$ qui mentionne x, on peut former les propositions $\forall x \in A, \varphi(x)$ et $\exists x \in A, \varphi(x)$. On peut introduire la proposition $\forall x \in A, \varphi(x)$ en démontrant $\varphi(x)$ sous l'hypothèse d'existence d'un nouvel individu $x \in A$. On peut utiliser la proposition $\forall x \in A, \varphi(x)$ en hypothèse en spécifiant un objet $t \in A$, ce qui fournit l'hypothèse $\varphi(t)$. On peut introduire la proposition $\exists x \in A, \varphi(x)$ en spécifiant un objet $u \in A$ et en démontrant la conclusion $\varphi(u)$. On peut utiliser la proposition $\exists x \in A, \varphi(x)$ en spécifiant un objet $u \in A$ et en démontrant la conclusion $\varphi(u)$. Symétriquement, on peut utiliser la proposition $\exists x \in A, \varphi(x)$ pour obtenir un objet $x \in A$ et une nouvelle hypothèse $\varphi(x)$. À noter, on ne sait rien sur l'objet x en dehors du fait qu'il satisfait $\varphi(x)$.

Remarque 3. Formellement, la quantification n'est pas restreinte aux éléments d'un ensemble fixé. Sa forme primitive est $\forall x, \varphi(x)$, où x quantifie donc sur tous les ensembles. Toutefois, cet usage est moins intuitif et n'est utile que lors de certaines constructions d'usage assez rare dont on signalera les quelques apparitions dans ces notes. La quantification $\forall x \in A, \varphi(x)$ que nous employons ci-dessus est dite *bornée* et constitue simplement un raccourci pour $\forall x, x \in A \implies \varphi(x)$. De même pour $\exists x \in A, \varphi(x)$ qui abrège $\exists x, x \in A \wedge \varphi(x)$.

Inclusion.

Le prédicat d'*inclusion*, noté $A \subseteq B$, n'est pas primitif mais défini comme suit à partir du prédicat d'appartenance.

$$A \subseteq B \overset{def}{\iff} \forall x \in A, x \in B.$$

2.2.5. Quelques axiomes ensemblistes

On rappelle ici quelques axiomes et résultats élémentaires de théorie des ensembles. L'exposé complet des axiomes de ZFC est hors.

Extensionalité. L'axiome d'*extensionalité* exprime que deux ensembles sont égaux si et seulement s'ils ont les mêmes éléments. Autrement dit, il exprime que deux ensembles A

et B sont égaux si $A \subseteq B$ et $B \subseteq A$. Notons que la direction "seulement si" est une conséquence automatique du transport d'égalité.

Compréhension. L'axiome de *compréhension* exprime qu'étant donné un ensemble A et un prédicat φ sur les éléments de A, il existe un ensemble A' tel que $x \in A'$ si et seulement si $x \in A$ et $\varphi(x)$. L'axiome d'extensionalité implique que cet ensemble est unique. On le note $\{x \in A \mid \varphi(x)\}$.

Remarque 4. Contrairement à la quantification, la compréhension est nécessairement restreinte aux éléments d'un ensemble fixé. En l'absence de cette restriction, on obtient facilement une théorie paradoxale. Le plus célèbre de ces paradoxes est dû à Bertrand Russel. Supposons qu'on puisse définir un ensemble en compréhension sans restriction sur le domaine de variation de x. On définit alors $\Omega \equiv \{x \mid x \notin x\}$ et on raisonne par cas sur $\Omega \in \Omega$. Si $\Omega \in \Omega$, alors par définition $\Omega \notin \Omega$, ce qui est contradictoire. De même, si $\Omega \notin \Omega$, alors $\Omega \in \Omega$. Dans les deux cas, on aboutit à une contradiction et le système s'effondre.

Ensemble des parties. L'axiome de l'*ensemble des parties* exprime que pour tout ensemble A, il existe un ensemble A' tel que les éléments de A' sont les parties de A. L'axiome d'extensionalité implique que cet ensemble est unique. On le note $\mathbb{P}A$.

Réunion. L'axiome de la *réunion* exprime que pour tout ensemble d'ensembles A, il existe un ensemble A' tel qu'un élément de A' est un élément d'un élement de A. L'axiome d'extensionalité implique que cet ensemble est unique. On le note $\bigcup A$. Lorsque l'ensemble A ne comprend que deux éléments A_1 et A_2, on écrit $A_1 \cup A_2$ pour sa réunion.

Infini. Il existe un ensemble infini. Sans énoncer précisément le sens de cet axiome, il entraîne l'existence de l'ensemble des entiers naturels, noté $\mathbb{N}$. On reviendra sur les propriétés de cet ensemble ultérieurement.

Quelques ensembles utiles. On définit l'ensemble vide, noté $\emptyset$, par compréhension comme $\{x \in \mathbb{N} \mid \bot\}$. On obtient un ensemble à un seul élément, noté $\mathbb{1}$, comme $\mathbb{P}\emptyset$. Son seul élement est donc $\emptyset$, que l'on notera $*$ dans ce contexte pour éviter toute confusion. À son tour, l'ensemble $\mathbb{P}\mathbb{1}$ a deux éléments, à savoir $\emptyset$ et $\{*\}$. Par convention, on note le premier élement 0 et le second 1, ce qui revient à définir l'ensemble $\mathbb{B}$ des booléens comme $\mathbb{P}\mathbb{1}$.

Étant donnés deux ensembles A et B, on peut définir leur produit cartésien $A \times B$ comme l'ensemble de toutes les paires (a, b) telles que $a \in A$ et $b \in B$.

Remarque 5. La définition du produit cartésien présuppose la possibilité de former les paires ordonnés, qui est assurée par les axiomes de ZFC que nous avons omis.

2.3. Relations et fonctions

2.3.1. Définitions de base

Définition 1 (Relation)**.** Une *relation* R est un triplet (A, B, G) où A et B sont des ensembles et G est un sous-ensemble du produit cartésien $A \times B$.

On appelle respectivement *domaine* et *codomaine* de R l'ensemble A et l'ensemble B. Par extension, on appelle *relation de A dans B* une relation de domaine A et codomaine B, et on écrit $R : A \nrightarrow B$ pour signifier que R est une telle relation. On note $\mathbf{Rel}(A, B)$ l'ensemble de toutes les relations de A dans B.

Définition 2 (Composition des relations)**.** Soient $R : A \nrightarrow B$ et $S : B \nrightarrow C$ deux relations. La *composée* de R et S, notée $R\,;S$, est la relation de domaine A et codomaine C définie par

$$R\,;S \equiv \{(x, z) \in A \times C \mid \exists y \in B, (x, y) \in R \wedge (y, z) \in S\}.$$

Définition 3 (Relation identité)**.** La relation *identité* sur un ensemble A, notée Id_A, est la relation de domaine et codomaine A définie par

$$Id_A \equiv \{(x, x) \in A\}.$$

La composition des relations est associative et admet la relation identité comme élément neutre à gauche et à droite, c'est-à-dire que

$$R\,;(S\,;T) = (R\,;S)\,;T \qquad\qquad R\,;Id_B = R = Id_A\,;R$$

pour toutes relations $R : A \nrightarrow B$, $S : B \nrightarrow C$ et $T : C \nrightarrow D$.

Définition 4 (Relation pleine et relation vide)**.** Soient A et B deux ensembles. La relation *pleine* et la relation *vide* de A dans B, notées respectivement $\mathbf{1}_{A,B}$ et $\mathbf{0}_{A,B}$, sont définies comme suit.

$$\mathbf{1}_{A,B} \equiv A \times B \qquad\qquad \mathbf{0}_{A,B} \equiv \emptyset$$

La relation vide est absorbante pour la composition, c'est-à-dire que pour toute relation $R : A \nrightarrow B$ et tout ensemble C on a $\mathbf{0}_{C,A}\,;R = \mathbf{0}_{C,B}$ et $R\,;\mathbf{0}_{B,C} = \mathbf{0}_{A,C}$.

Définition 5 (Relation réciproque)**.** La *réciproque* d'une relation $R : A \nrightarrow B$, notée R°, est la relation de domaine B et codomaine A définie par

$$R^\circ \equiv \{(y, x) \in R\}.$$

La réciproque, qui renverse une relation, est une opération essentielle. Elle satisfait une propriété importante de *fonctorialité*, c'est-à-dire de compatibilité avec la composition. Cette propriété signifie que pour tous A, B, C et toutes relations $R : A \nrightarrow B$ et $S : B \nrightarrow C$, on a $(R\,;S)^\circ = S^\circ\,;R^\circ$ et $Id_A^\circ = Id_A$. De plus, la réciproque est involutive, c'est-à-dire que $R^{\circ\circ} = R$ pour toute relation $R : A \nrightarrow B$.

Les relations de domaine A et codomaine B fixées sont des parties du produit cartésien $A \times B$, et sont donc ordonnées par l'inclusion. Il est clair que la relation vide est la plus petite relation de A dans B, et la relation pleine la plus grande. La composition des relations est croissante, au sens où si $R_1 \subseteq R_2$ et $S_1 \subseteq S_2$ sont des relations de A dans B, alors $R_1 \,;\, S_1 \subseteq R_2 \,;\, S_2$. De même pour la réciproque : on aura $R_1^\circ \subseteq R_2^\circ$ dès lors que $R_1 \subseteq R_2$. On reviendra au caractère ordonné des relations au chapitre 3.

2.3.2. Image directe et inverse

Les relations sont un concept très utile pour manipuler commodément les définitions ensemblistes en minimisant la manipulation des quantificateurs. En particulier, toute partie A' d'un ensemble A définit une relation notée $\langle A' \rangle$ de $\mathbb{1}$ dans A par

$$\langle A' \rangle \equiv \{(*, x) \in \mathbb{1} \times A \mid x \in A'\}$$

et toute relation $R : \mathbb{1} \nrightarrow A$ définit une partie de A par $\{x \in A \mid (*, x) \in R\}$. Cette correspondance définit une bijection entre $\mathbf{Rel}(\mathbb{1}, A)$ et $\mathbb{P}A$, tout comme entre $\mathbb{P}A$ et $\mathbf{Rel}(A, \mathbb{1})$.

Définition 6. L'*image directe* d'une relation $R : A \nrightarrow B$ est la partie de B définie par la relation $\mathbf{1}_{\mathbb{1},A} \,;\, R$. L'*image inverse* de $R : A \nrightarrow B$ est la partie de A définie par la relation $\mathbf{1}_{\mathbb{1},B} \,;\, R^\circ$.

2.4. Familles d'ensembles

2.5. Relations d'équivalence et ensembles quotients

3. Ensembles ordonnés

3.1. Relations d'ordres et ensembles ordonnés

3.2. Fonctions croissantes

3.3. Suprema et infima

3.4. Treillis et ordres inductifs

3.5. Opérations sur les ensembles ordonnés

3.6. Théorèmes de point fixe

- Knaster-Tarski
- Scott
- Pataraia

3.7. Adjonctions

Motiver par l'inexistence d'inverses à l'addition dans les entiers naturels.

4. Monoïdes

4.1. Définitions de base

Définition 7. Un *monoïde* M est un triplet $(|M|, +_M, 0_M)$ où $|M|$ est un ensemble appelé *éléments de* M, $+_M : |M|^2 \to |M|$ est une application binaire associative appelée *loi* de M, et 0_M est un élément de $|M|$ neutre pour $+_M$.

On omettra les indices de la loi et de l'élément neutre lorsque le monoïde dont il est question peut être déduit du contexte. De même, pour simplifier les notations on identifira le monoïde à son ensemble support. Si x, y, z sont des éléments d'un monoïde M, l'associativité de sa loi permet d'écrire $x + y + z$ pour $(x + y) + z = x + (y + z)$ sans introduire d'ambiguïté réelle.

4.2. Éléments remarquables

Dans cette section, on fixe un monoïde $(M, +, 0)$.

4.2.1. Éléments inversibles

Soient x et y deux éléments de M. Si $x + y = 0$ on dit que x est un *inverse à gauche* de y et, réciproquement, que y est est un *inverse à droite* de x. Un élément qui a un inverse à gauche (resp. à droite) est donc *inversible à gauche* (resp. *à droite*).

Un élément inversible à gauche et à droite est simplement dit *inversible*. Un élément qui est à la fois un inverse à gauche est à droite est appelé *inverse bilatéral*, ou simplement *inverse*. Supposons que x est inversible, et soit y un de ses inverses à gauche et z l'un de ses inverses à droite. On voit que $y = y + 0 = y + x + z = 0 + z = z$ et donc tous les inverses à gauche et à droite de x sont égaux. En conséquence, un élément inversible a exactement un inverse.

Un monoïde dont tous les éléments sont inversibles est appelé *groupe*.

4.2.2. Autres éléments remarquables

Soient x, y, z des éléments d'un monoïde $(M, +, 0)$. On dit que x est :

- *absorbant à gauche* si $x + y = x$,
- *simplifiable à gauche* si $x + y = x + z$ implique $y = z$,
- *acyclique à gauche* si $x = x + y$ implique $y = 0$,

- *irréductible* si $x = y + z$ implique $y = 0$ ou $z = 0$.

Dans chaque cas, on dira que x est absorbant (respectivement simplifiable, acyclique, inversible) *à droite* lorsqu'il est absorbant (respectivement simplifiable, acyclique) à gauche pour la loi commutée. On dit qu'un élément est simplement absorbant (respectivement simplifiable, acyclique, inversible) lorsqu'il est absorbant (respectivement simplifiable, acyclique) à la fois à gauche et à droite. Dans un monoïde commutatif, les notions d'éléments absorbant (respectivement simplifiable, acyclique) à gauche et à droite se confondent.

Un élément inversible est simplifiable. Un élément simplifiable à gauche (respectivement à droite) est acyclique à gauche (respectivement à droite). Un monoïde dont l'élément neutre est irréductible est dit *rigide*.

Remarque 6. Le monoïde $(\mathbb{Z}, +, 0)$ est un groupe et donc tous ses éléments sont simplifiables à gauche et à droite, néanmoins il n'est pas rigide puisque $1 + (-1) = 0$.

4.3. Opérations sur les monoïdes

4.3.1. Préordre de décomposabilité

Soit $(M, +, 0)$ un monoïde et (x, y) une paire d'éléments de M. On dit que x *décompose* y *à gauche*, et on note $x \preceq y$, s'il existe $z \in M$ tel que $x + z = y$.

$$x \preceq y \quad \overset{def}{\iff} \quad \exists z \in M : x + z = y$$

Il s'agit d'une relation de préordre sur les éléments de M dont la réflexivité découle de l'existence de l'élément neutre et la transitivité de l'associativité de la loi. Il est clair que l'élément neutre est minimal. Il est également clair que la loi du monoïde est croissante à droite pour la décomposabilité : si $x_1 \preceq x_2$ alors $y + x_1 \preceq y + x_2$. Tout élément z absorbant à droite est maximal puisque $x + z = z$ pour tout x.

Exemple 1. La relation de décomposabilité à gauche associée au monoïde $(\mathbb{N}, +, 0)$ est la relation d'ordre habituelle sur les entiers naturels. Celle associée au monoïde $(\mathbb{N}^+, \times, 1)$ est la relation bien connue de divisibilité. La même construction appliquée aux mots sur un alphabet munis de la concaténation donne l'ordre préfixe.

Propriété 1. *Si M est un monoïde rigide dont tous les éléments sont acycliques à gauche alors la relation $\preceq$ induite est antisymétrique.*

Démonstration. Supposons $x \preceq y$ et $y \preceq x$. Il existe donc z_1 tel que $x + z_1 = y$ et z_2 tel que $y + z_2 = x$. On en déduit $x + z_1 + z_2 = y + z_2 = x$ et $y + z_2 + z_1 = x + z_1 = y$. Puisque x et y sont acycliques à gauche, on doit avoir $z_1 + z_2 = 0 = z_2 + z_1$. Puisque M est rigide, on en déduit $z_1 = 0 = z_2$ et donc $x = y$. □

Propriété 2. *Si x est un élément inversible alors x est minimal pour $\preceq$.*

Démonstration. On a nécessairement $x + x^{-1} + y = y$ et donc $x \preceq y$ pour tout y. □

Corollaire 1. *Si M est un groupe alors la relation $\preceq$ induite est codiscrète.*

Propriété 3. *Si la relation $\preceq$ est antisymétrique alors le monoïde M est rigide et …*

Démonstration. Supposons $\preceq$ antisymétrique.
Si $0 = y + z$ alors $y \preceq 0$ et donc $y = 0$ par antisymétrie. Donc 0 est irréductible.
…
□

5. Induction et coinduction

5.1. L'exemple des entiers naturels

Les ensembles et sous-ensembles définis inductivement occupent une place prépondérante en mathématiques constructives et en informatique théorique. Ils jouent le rôle de *structure libre* de l'algébriste et à celle de *syntaxe* du théoricien des langages de programmation, comme on l'expliquera dans les chapitres suivants.

Le plus connu d'entre eux est l'ensemble des entiers naturels, qu'on peut caractériser informellement comme le *plus petit ensemble clos par zéro et successeur*.

On considère l'ensemble des entiers naturels.

6. Réécriture abstraite

6.1. Systèmes de réécriture abstraits

6.2. Normalisation d'un système de réécriture

6.3. Confluence d'un système de réécriture

6.4. Réécriture infinitaire

Deuxième partie

Algèbre universelle

7. Syntaxe du premier ordre

7.1. Théories du premier ordre

7.1.1. Motivations

Quelques exemples de structures algébriques.

7.1.2. La notion de théorie du premier ordre

7.2. Termes du premier ordre et théories équationnelles

Une *signature du premier ordre* est une famille Σ d'ensembles $(\Sigma_n)_{n\in\mathbb{N}}$.

7.3. Présentation d'une théorie du premier ordre

8. Réécriture du premier ordre

8.1. Réécriture de termes du premier ordre

Radicaux, résidus.

8.2. Confluence d'un système de réécriture du premier ordre

8.3. Présentation convergente d'une théorie du premier ordre

Knuth-Bendix.

9. Syntaxe du second ordre

9.1. Théories algébriques du second ordre

9.1.1. Motivations

9.2. Théorie algébrique du second ordre

Définition 8.

9.3. Termes du second ordre

9.4. Présentation d'une théorie algébrique du second ordre

9.5. Réécriture du second ordre

9.5.1. Les systèmes de réécriture combinatoires

9.5.2. Confluence et normalisation

Troisième partie

Langages simplement typés

10. Le système T

Ce chapitre discute du système $\mathcal{T}$, formalisme calculatoire qui peut être vu comme un langage de programmation rudimentaire où l'on dispose de fonctions et de la récursion primitive sur les entiers naturels. Ce système a été proposé par Kurt Gödel [3] dans le cadre de travaux sur les fondations des mathématiques. On en donne une présentation modernisée due à William Tait [7].

10.1. Syntaxe pure

On commence par traiter des aspects purement syntaxiques du système, notamment de l'égalité des termes à renommage des variables liées près et de la substitution. Ces aspects sont communs à tous les formalismes qui mettent en jeu la notion de variable et on apprend vite à raisonner informellement à leur sujet. Il est cependant nécessaire de traiter sérieusement ces questions au moins une fois. On adopte ici une approche qui a l'avantage d'être concrète et élémentaire, au prix de raisonnements malcommodes.

10.1.1. Noms

On se donne un ensemble infini dénombrable de noms, noté $\mathbb{A}$. Ses éléments sont notés x, y, z, x', x_1, x_2, etc. On se donne également une fonction ν qui à toute partie finie A de $\mathbb{A}$ associe un élément $\nu(A)$ de $\mathbb{A}$ tel que $\nu(A) \notin A$. Le choix de ν n'influe pas sur les définitions finales. Toutefois, le lecteur ou la lectrice à l'esprit concret peut par exemple penser à $\mathbb{A}$ comme à l'ensemble des entiers naturels et poser $\nu(A) = 1 + \max A$.

10.1.2. Prétermes et présubstitutions

Définition 9. L'ensemble $\mathsf{PreTerm}_{\mathcal{T}}$ des *prétermes* et l'ensemble $\mathsf{PreSub}_{\mathcal{T}}$ des *présubstitutions* de $\mathcal{T}$ sont définis inductivement par les grammaires présentées à la figure 10.1.

En vocabulaire informatique, les prétermes sont donc les arbres de syntaxe abstraite du langage, tandis que les présubstitutions sont des listes de termes nommés. On adopte quelques raccourcis de notation pour faciliter la lecture. Tout d'abord, on écrira généralement x plutôt que $\mathtt{var}(x)$ lorsqu'aucune confusion ne peut en résulter. Ensuite, on écrira $M\, N_1\, N_2 \ldots N_k$ pour $\mathtt{app}(\ldots\ (\mathtt{app}(\mathtt{app}(M, N_1), N_2), \ldots), N_k)$. On abrègera en $\underline{n}$ le préterme $\mathtt{suc}^n(\mathtt{zero})$ contenant n successeurs imbriqués. Enfin, on écrira $x_1 \backslash M_1, \ldots, x_k \backslash M_k$ pour la présubstitution $\mathtt{id}, x_1 \backslash M_1, \ldots, x_k \backslash M_k$.

$\mathsf{PreTerm}_{\mathcal{T}} \ni M, N, P ::=$		**Prétermes**
	$\mid \mathtt{var}(x)$	Variable
	$\mid \mathtt{fun}(x.M)$	Abstraction
	$\mid \mathtt{app}(M, N)$	Application
	$\mid \mathtt{zero}$	Zéro
	$\mid \mathtt{suc}(M)$	Successeur
	$\mid \mathtt{fold}_{\mathtt{Nat}}(M, N, (x, y).P)$	Récursion primitive ($x \neq y$)
$\mathsf{PreSub}_{\mathcal{T}} \ni \sigma, \varphi ::=$		**Présubstitutions**
	$\mid \mathtt{id}$	Substitution identique
	$\mid \sigma, x \backslash M$	Liaison

FIG. 10.1. : Syntaxe de $\mathcal{T}$

On va définir l'action d'une présubstitution σ sur un préterme M. Informellement, cette action consiste à parcourir M pour remplacer chacune de ses variables x par le préterme que σ associe à x. On commence par définir cette dernière notion.

Définition 10 (Application d'une présubstitution à une variable)**.** L'application de la présubstitution σ à la variable x, notée $x[\sigma]$, est définie par récurrence sur σ.

$$
\begin{aligned}
-[=] &: \mathbb{A} \times \mathsf{PreSub}_{\mathcal{T}} \to \mathsf{PreTerm}_{\mathcal{T}} \\
x[\mathtt{id}] &= x \\
x[\sigma, y \backslash M] &= \begin{cases} M & \text{si } y = x \\ x[\sigma] & \text{sinon.} \end{cases}
\end{aligned}
$$

Définir l'action d'une présubstitution sur un préterme est plus délicat. On va commencer par la définition naïve, avant de discuter ses défauts.

Définition 11 (Application naïve d'une présubstitution à un préterme)**.** L'application naïve de la présubstitution σ au préterme M, notée $M[\sigma]_{\mathsf{n}}$, est définie par récurrence sur M.

$$
\begin{aligned}
-[=]_{\mathsf{n}} &: \mathsf{PreTerm}_{\mathcal{T}} \times \mathsf{PreSub}_{\mathcal{T}} \to \mathsf{PreTerm}_{\mathcal{T}} \\
x[\sigma]_{\mathsf{n}} &= x[\sigma] \\
\mathtt{fun}(x.M)[\sigma]_{\mathsf{n}} &= \mathtt{fun}(x.M[\sigma, x \backslash x]_{\mathsf{n}}) \\
\mathtt{app}(M, N)[\sigma]_{\mathsf{n}} &= \mathtt{app}(M[\sigma]_{\mathsf{n}}, N[\sigma]_{\mathsf{n}}) \\
\mathtt{zero}[\sigma]_{\mathsf{n}} &= \mathtt{zero} \\
\mathtt{suc}(M)[\sigma]_{\mathsf{n}} &= \mathtt{suc}(M[\sigma]_{\mathsf{n}}) \\
\mathtt{fold}_{\mathtt{Nat}}(M, N, (x, y).P)[\sigma]_{\mathsf{n}} &= \mathtt{fold}_{\mathtt{Nat}}(M[\sigma]_{\mathsf{n}}, N[\sigma]_{\mathsf{n}}, (x, y).M[\sigma, x \backslash x, y \backslash y]_{\mathsf{n}})
\end{aligned}
$$

Quelles sont les propriétés de l'application naïve ? Tout d'abord, on peut montrer que la substitution identique $\mathtt{id}$ mérite son nom : l'appliquer laisse le terme inchangé.

Propriété 4. $x[x_1\backslash x_1, \ldots, x_k\backslash x_k]_{\mathsf{n}} = x$.

Démonstration. On procède par induction sur l'entier k.

- Cas $k = 0$: on est dans le cas $x[\mathtt{id}]_{\mathsf{n}}$ qui est égal à x par définition.
- Cas $k = k' + 1$: alors soit $x = x_k$, et on conclue immédiatement car

$$x[x_1\backslash x_1, \ldots, x_k\backslash x_k]_{\mathsf{n}} = x_k = x,$$

soit $x \neq x_k$, auquel cas on a

$$x[x_1\backslash x_1, \ldots, x_{k'}\backslash x_{k'}, x_k\backslash x_k]_{\mathsf{n}} = x[\mathtt{id}, x_1\backslash x_1, \ldots, x_{k'}\backslash x_{k'}]_{\mathsf{n}} = x$$

où la deuxième égalité est l'hypothèse d'induction.

□

Propriété 5. $M[x_1\backslash x_1, \ldots, x_k\backslash x_k]_{\mathsf{n}} = M$.

Démonstration. Par induction sur le terme M. Tous les cas sont des applications immédiates de la définition de l'application naïve de présubstitution et de l'hypothèse d'induction, à l'exception du cas des variables qui fait appel à la propriété 4. □

Corollaire 2. $M[\mathtt{id}]_{\mathsf{n}} = M$.

Le corollaire 2 exprime que la substitution identique agit sur les termes comme l'application identique. Il est donc naturel de se demander s'il existe une opération sur les substitutions qui corresponde à la composition des applications naïves de celles-ci. Autrement dit, les présubstitutions σ_1 et σ_2 étant données, existe-t-il une présubstitution φ telle que pour tout préterme M, on ait $M[\varphi]_{\mathsf{n}} = M[\sigma_1]_{\mathsf{n}}[\sigma_2]_{\mathsf{n}}$? Le raisonnement ci-dessous, dû à Krivine [4], montre que la réponse est négative.

Soient x, y, z trois noms distincts, et posons $\sigma_1 = x\backslash y$ et $\sigma_2 = y\backslash x$. On a $y[x\backslash y]_{\mathsf{n}}[y\backslash x]_{\mathsf{n}} = x$ et $z[x\backslash y]_{\mathsf{n}}[y\backslash x]_{\mathsf{n}} = z$. Donc, si φ existe, elle doit être $y\backslash x$. Pourtant, on a

$$\begin{aligned}\mathtt{fun}(y.x)[x\backslash y]_{\mathsf{n}}[y\backslash x]_{\mathsf{n}} = \mathtt{fun}(y.y)[y\backslash x]_{\mathsf{n}} &= \mathtt{fun}(y.y)\\ &\neq \mathtt{fun}(y.x)[y\backslash x]_{\mathsf{n}} = \mathtt{fun}(y.x)\end{aligned}$$

et donc φ ne peut pas jouer le rôle de la composition de σ_1 et σ_2.

Dans le raisonnement ci-dessus, le problème survient à cause de la *capture* de y dans $\mathtt{fun}(y.x)[x\backslash y]$. Ce mot désigne le changement de statut de la variable y, qui apparaît *libre* dans la substitution $x\backslash y$ mais *liée* dans $\mathtt{fun}(y.x)[x\backslash y]_{\mathsf{n}} = \mathtt{fun}(y.y)$. Informellement, une variable x apparaît libre dans un terme si elle y apparaît hors de la portée d'une opération qui la lie, comme $\mathtt{fun}(x.-)$.

Définition 12 (Variables libres et liées)**.** On définit l'ensemble $\mathsf{fv}(M)$ des variables libres d'un terme par récurrence sur M comme suit.

$$\begin{aligned}
\mathsf{fv}(x) &= \{x\} \\
\mathsf{fv}(\mathtt{app}(M,N)) &= \mathsf{fv}(M) \cup \mathsf{fv}(N) \\
\mathsf{fv}(\mathtt{fun}(x.M)) &= \mathsf{fv}(M) - \{x\} \\
\mathsf{fv}(\mathtt{zero}) &= \emptyset \\
\mathsf{fv}(\mathtt{suc}(M)) &= \mathsf{fv}(M) \\
\mathsf{fv}(\mathtt{fold}_{\mathtt{Nat}}(M,N,(x,y).P)) &= \mathsf{fv}(M) \cup \mathsf{fv}(N) \cup (\mathsf{fv}(P) - \{x,y\})
\end{aligned}$$

L'ensemble $\mathsf{fv}(\sigma)$ (respectivement $\mathsf{bv}(\sigma)$) des variables *libres* (respectivement *liées*) d'une substitution σ est défini par récurrence comme suit.

$$\begin{aligned}
\mathsf{fv}(\mathtt{id}) &= \emptyset & \mathsf{bv}(\mathtt{id}) &= \emptyset \\
\mathsf{fv}(\sigma, x\backslash M) &= \mathsf{fv}(\sigma) \cup \mathsf{fv}(M) & \mathsf{bv}(\sigma, x\backslash M) &= \mathsf{bv}(\sigma) \cup \{x\}
\end{aligned}$$

Pour alléger les notations, on écrira $\nu(X_1,\dots,X_n)$ pour désigner $\nu(\mathsf{fv}(X_1) \cup \dots \cup \mathsf{fv}(X_n))$, où X_i désigne soit un terme, soit une substitution. De plus, on écrira $x_1,\dots,x_m = \nu(X_1,\dots,X_n)$ pour désigner le choix de m variables libres successives distinctes. Formellement, on a donc $x_i = \nu(X_1,\dots,X_n,x_1,\dots,x_{i-1})$. Enfin, on écrira $x_1,\dots,x_m \# X_1,\dots,X_n$ pour exprimer que $x_1,\dots,x_m \notin \mathsf{fv}(X_1) \cup \dots \cup \mathsf{fv}(X_n)$ avec $x_1,\dots,x_m$ distincts.

Remarque 7. On ne parlera **jamais** des variables liées *au sein d'un terme*, pour éviter de distinguer des termes qui ne se distinguent que par celles-ci.

On peut maintenant définir la substitution de sorte à éviter la capture. On renomme toutes les variables liées par des variables *fraîches*, c'est-à-dire non libres, dans le terme et de la substitution concernés.

Définition 13 (Application d'une présubstitution à un préterme)**.** L'application de la présubstitution σ au préterme M, notée $M[\sigma]$, est définie par récurrence sur M.

$$\begin{aligned}
-[=] &: \mathsf{PreTerm}_{\mathcal{T}} \times \mathsf{PreSub}_{\mathcal{T}} \to \mathsf{PreTerm}_{\mathcal{T}} \\
\mathtt{var}(x)[\sigma] &= x[\sigma] \\
\mathtt{fun}(x.M)[\sigma] &= \mathtt{fun}(z.M[\sigma, x\backslash z]) \text{ où } z = \nu(\mathtt{fun}(x.M),\sigma) \\
\mathtt{app}(M,N)[\sigma] &= \mathtt{app}(M[\sigma], N[\sigma]) \\
\mathtt{zero}[\sigma] &= \mathtt{zero} \\
\mathtt{suc}(M)[\sigma] &= \mathtt{suc}(M[\sigma]) \\
\mathtt{fold}_{\mathtt{Nat}}(M,N,(x,y).P)[\sigma] &= \mathtt{fold}_{\mathtt{Nat}}(M[\sigma], N[\sigma], (z_1,z_2).P[\sigma, x\backslash z_1, y\backslash z_2]) \\
&\quad \text{où } z_1,z_2 = \nu(\mathtt{fold}_{\mathtt{Nat}}(M,N,(x,y).P),\sigma)
\end{aligned}$$

On écrira qu'une substitution σ *agit* sur un préterme M pour donner le préterme $M[\sigma]$, et on nommera donc *action* de σ l'application qui envoie M sur $M[\sigma]$. On notera parfois cette application $(-)[\sigma]$.

La propriété ci-dessous exprime l'effet qu'à l'application d'une substitution sur les variables libres d'un terme. En particulier, toutes les variables libres du terme après action de la substitution sont soit libres dans la substitution, soit libres dans le terme initial et non liées par la substitution, soit les deux.

Propriété 6. $\mathsf{fv}(M[\sigma]) \cup \mathsf{fv}(\sigma) = (\mathsf{fv}(M) - \mathsf{bv}(\sigma)) \cup \mathsf{fv}(\sigma)$.

Corollaire 3. $\mathsf{fv}(M[\sigma]) \subseteq (\mathsf{fv}(M) - \mathsf{bv}(\sigma)) \cup \mathsf{fv}(\sigma)$.

On définit l'opération de composition de substitution conjecturée plus haut. Informellement, la composition $\sigma \circ \varphi$ contient les liaisons $x\backslash M$ de φ suivies de celles de σ où l'on a appliqué φ à chaque terme.

Définition 14 (Composition de substitution)**.** L'opération de *composition* associe à deux substitutions σ et φ leur *composée*, notée $\sigma \circ \varphi$, définie par récurrence sur σ comme suit.

$$\begin{aligned} - \circ = &: \mathsf{PreSub}_{\mathcal{T}} \times \mathsf{PreSub}_{\mathcal{T}} \to \mathsf{PreSub}_{\mathcal{T}} \\ \mathtt{id} \circ \varphi &= \varphi \\ (\sigma, x\backslash M) \circ \varphi &= (\sigma \circ \varphi), x\backslash M[\varphi] \end{aligned}$$

Propriété 7. $\mathsf{fv}(\sigma \circ \varphi) = (\mathsf{fv}(\sigma) - \mathsf{bv}(\varphi)) \cup \mathsf{fv}(\varphi)$ *et* $\mathsf{bv}(\sigma \circ \varphi) = \mathsf{bv}(\sigma) \cup \mathsf{bv}(\varphi)$.

Pour démontrer des propriétés plus intéressantes de la composée, on a besoin de définir une autre opération sur les substitutions. Celle-ci, très simple, concerne intuitivement à concaténer deux substitutions.

Définition 15 (Produit de substitution)**.** L'opération de *multiplication* associe à deux substitutions σ et φ leur *produit*, noté σ, φ, défini par récurrence sur φ comme suit.

$$\begin{aligned} -, = &: \mathsf{PreSub}_{\mathcal{T}} \times \mathsf{PreSub}_{\mathcal{T}} \to \mathsf{PreSub}_{\mathcal{T}} \\ \sigma, \mathtt{id} &= \sigma \\ \sigma, (\varphi, x\backslash M) &= (\sigma, \varphi), x\backslash M \end{aligned}$$

Propriété 8. $\mathsf{fv}(\sigma, \varphi) = \mathsf{fv}(\sigma) \cup \mathsf{fv}(\varphi)$ *et* $\mathsf{bv}(\sigma, \varphi) = \mathsf{bv}(\sigma) \cup \mathsf{bv}(\varphi)$.

Propriété 9. *Le produit de substitution est associatif et admet la substitution identique comme élément neutre à gauche et à droite.*

Remarque 8. En langage algébrique, la propriété 9 exprime que le produit équipe l'ensemble des substitutions d'une structure de monoïde.

10.1.3. Équivalence α et renommage des variables liées

On voudrait maintenant montrer que cette définition de la substitution remédie aux défauts de la précédente. Ce n'est pas le cas, du moins pas littéralement. Pire, on semble même avoir régressé : l'analogue du corollaire 2 échoue. Ainsi, en appliquant $\mathtt{fun}(x.x)[\mathtt{id}]$ on obtient $\mathtt{fun}(\nu(\emptyset).\nu(\emptyset))$, mais le nom $\nu(\emptyset)$ n'a aucune raison d'être le même que x.

$$\boxed{M \equiv_\alpha N} \qquad \frac{}{x \equiv_\alpha x} \qquad \frac{M_1 \equiv_\alpha M_2 \qquad N_1 \equiv_\alpha N_2}{\mathtt{app}(M_1, N_1) \equiv_\alpha \mathtt{app}(M_2, N_2)} \qquad \frac{}{\mathtt{zero} \equiv_\alpha \mathtt{zero}}$$

$$\frac{M_1[x_1\backslash y] \equiv_\alpha M_2[x_2\backslash y] \qquad y = \nu(M_1, M_2)}{\mathtt{fun}(x_1.M_1) \equiv_\alpha \mathtt{fun}(x_2.M_2)} \qquad \frac{M_1 \equiv_\alpha M_2}{\mathtt{suc}(M_1) \equiv_\alpha \mathtt{suc}(M_2)}$$

$$\frac{\begin{array}{c} M_1 \equiv_\alpha M_2 \\ N_1 \equiv_\alpha N_2 \qquad P_1[x_1\backslash z_1, y_1\backslash z_2] \equiv_\alpha P_2[x_2\backslash z_1, y_2\backslash z_2] \qquad z_1, z_2 = \nu(P_1, P_2) \end{array}}{\mathtt{fold}_{\mathtt{Nat}}(M_1, N_1, (x_1, y_1).P_1) \equiv_\alpha \mathtt{fold}_{\mathtt{Nat}}(M_2, N_2, (x_2, y_2).P_2)}$$

FIG. 10.2. : Relation d'équivalence α entre prétermes

Pour remédier à ce défaut, il est naturel d'identifier les prétermes qui ne diffèrent que par l'identité de leurs variables liées. Le cas intéressant est celui des lieurs, où l'on doit exprimer que les variables liées n'importent pas. Par exemple, deux termes $\mathtt{fun}(x.M)$ et $\mathtt{fun}(y.N)$ doivent être considérés équivalents si $M[x\backslash z]$ et $N[y\backslash z]$ sont équivalents pour tous les noms z libres ni dans M ni dans N. Par exemple, on ne veut pas identifier $\mathtt{fun}(x.\mathtt{app}(x, y))$ et $\mathtt{fun}(y.\mathtt{app}(y, x))$.

En suivant ce raisonnement, on aboutit à une définition inductive de l'égalité modulo renommage des occurences liées des variables, appelée équivalence α.

Définition 16. La relation d'*équivalence* α entre prétermes, notée $M \equiv_\alpha N$, est définie inductivement par les règles données à la figure 10.2.

Remarque 9. Il existe de nombreuses définitions possibles de l'équivalence α. L'avantage de la définition ci-dessus est son caractère algorithmique : on a en réalité défini une fonction récursive totale capable de décider l'égalité modulo renommage de deux termes. En contrepartie, il est plus difficile de démontrer qu'elle jouit des propriétés attendues.

On souhaite maintenant considérer l'ensemble des prétermes modulo la relation d'équivalence α, ce qui suppose que cette dernière soit bien une relation d'équivalence.

Théorème 1. *L'équivalence α est une relation d'équivalence.*

Si démontrer la réflexivité et la symétrie n'est pas difficile, le cas de la transitivité est nettement plus complexe. Pour l'établir, on va étendre la notion d'équivalence α aux présubstitutions.

Définition 17. Soient X et Y deux ensembles de noms. On note $X; Y \vdash \sigma \equiv_\alpha \varphi$ lorsque la paire de substitutions (σ, φ) est dans la relation d'*équivalence α modulo* X, Y. Cette relation est définie inductivement par les règles données à la figure 10.3. La relation d'*équivalence α* est définie comme la relation d'équivalence α modulo $\mathbb{A}, \mathbb{A}$.

Intuitivement, l'équivalence α de deux substitutions σ et φ modulo les ensembles de noms X et Y exprime que σ et φ sont équivalentes si l'on s'autorise à oublier n'importe

$$\boxed{X;Y \vdash \sigma \equiv_\alpha \varphi}$$

$$\textsc{Erase}\ \frac{x \notin X}{X;Y \vdash x\backslash M \equiv_\alpha \mathtt{id}} \qquad \textsc{CongProd}\ \frac{X;Y \vdash \sigma_2 \equiv_\alpha \varphi_2 \qquad X \cap \mathsf{bv}(\varphi_2);Y \cap \mathsf{bv}(\sigma_2) \vdash \sigma_1 \equiv_\alpha \varphi_1}{X;Y \vdash \sigma_1, \sigma_2 \equiv_\alpha \varphi_1, \varphi_2}$$

$$\textsc{CongComp}\ \frac{X;Y \vdash \sigma_1 \equiv_\alpha \varphi_1 \qquad X \cup \mathsf{fv}(\varphi_1);Y \cup \mathsf{fv}(\sigma_1) \vdash \sigma_2 \equiv_\alpha \varphi_2}{X;Y \vdash \sigma_1 \circ \sigma_2 \equiv_\alpha \varphi_1 \circ \varphi_2} \qquad \textsc{CongBind}\ \frac{M \equiv_\alpha N}{X;Y \vdash x\backslash M \equiv_\alpha x\backslash N}$$

$$\textsc{Trivial}\ \frac{}{X;Y \vdash x\backslash x \equiv_\alpha \mathtt{id}} \qquad \textsc{Sym}\ \frac{Y;X \vdash \varphi \equiv_\alpha \sigma}{X;Y \vdash \sigma \equiv_\alpha \varphi} \qquad \textsc{Trans}\ \frac{X;Y \vdash \sigma_1 \equiv_\alpha \sigma_2 \qquad X;Y \vdash \sigma_2 \equiv_\alpha \sigma_3}{X;Y \vdash \sigma_1 \equiv_\alpha \sigma_3}$$

FIG. 10.3. : Relation d'équivalence α entre substitutions

quelle liaison $x\backslash M$ de σ si x n'est *pas* dans X, ainsi que n'importe quelle liaison $y\backslash N$ de φ où y n'est *pas* dans Y. Pour tous X, Y, la relation binaire $X;Y \vdash (-) \equiv_\alpha (=)$ est une relation d'équivalence dont on peut prouver la réflexivité par une induction simple.

Propriété 10. *L'équivalence α entre présubstitutions est une relation d'équivalence.*

Démonstration. On prouve la réflexivité par une induction de routine en utilisant la réflexivité de l'équivalence α définie sur les prétermes. La symétrie et la transitivité sont garanties par la présence des règles SYM et TRANS. □

Propriété 11. *Soient X, X', Y, Y' des ensembles de noms. Si $X \subseteq X'$, $Y \subseteq Y'$ et $X';Y' \vdash \sigma \equiv_\alpha \varphi$ alors $X;Y \vdash \sigma \equiv_\alpha \varphi$.*

Démonstration. Induction de routine. □

Nous allons maintenant montrer que l'α-convertibilité interagit bien avec une classe particulière de substitutions, les *renommages*.

Définition 18 (Renommage)**.** Un *renommage* ρ est une substitution où les termes substitués sont tous des variables, c'est-à-dire une substitution de la forme $\rho = x_1\backslash y_1, \ldots, x_n\backslash y_n$.

Les renommages, qui seront dénotés $\rho, \rho_1, \rho_2, \rho'$ et ainsi de suite, constituent une classe de substitution élémentaires. En particulier, ils vérifient une propriété essentielle pour les démonstrations : faire agir un renommage sur un terme laisse sa taille inchangée.

Définition 19 (Taille d'un préterme)**.** La *taille* d'un préterme M, notée $\mathsf{size}(M)$, est

l'entier défini par récurrence sur M comme suit.

$$\begin{aligned}
\mathsf{size} : \mathsf{PreTerm}_{\mathcal{T}} &\to \mathbb{N} \\
\mathsf{size}(\mathtt{var}(x)) &= 1 \\
\mathsf{size}(\mathtt{app}(M, N)) &= 1 + \mathsf{size}(M) + \mathsf{size}(N) \\
\mathsf{size}(\mathtt{fun}(x.M)) &= 1 + \mathsf{size}(M) \\
\mathsf{size}(\mathtt{zero}) &= 1 \\
\mathsf{size}(\mathtt{suc}(M)) &= 1 + \mathsf{size}(M) \\
\mathsf{size}(\mathtt{fold}_{\mathtt{Nat}}(M, N, (x, y).P)) &= 1 + \mathsf{size}(M) + \mathsf{size}(N) + \mathsf{size}(P)
\end{aligned}$$

Remarque 10. La définition exacte des cas de base de size n'est pas importante pour l'usage que l'on va en faire dans les arguments d'induction. L'essentiel est que la taille d'un terme soit strictement supérieure à celle de ses sous-termes. Une fonction dotée de cette propriété est parfois appelée *mesure* ; voir l'exercice 11 pour plus de détails.

Propriété 12 (Stabilité de la taille par renommage)**.** $\mathsf{size}(M[\rho]) = \mathsf{size}(M)$

Démonstration. Par induction sur M. On démontre le cas clef des variables par une seconde induction sur ρ. Tous les autres cas sont routiniers. □

Propriété 13 (Passage au quotient de la taille)**.** *Si* $M \equiv_\alpha N$ *alors* $\mathsf{size}(M) = \mathsf{size}(N)$.

Démonstration. Par induction sur la dérivation, en utilisant la propriété 12 pour traiter l'abstraction et la récursion primitive. □

Propriété 14. $x[\rho_1][\rho_2] = x[\rho_1 \circ \rho_2]$.

Démonstration. Par une induction sur ρ_1. Le cas $\rho_1 = \mathtt{id}$ est immédiat. Dans le cas $\rho_1 = \rho_1', y\backslash z$, on doit distinguer les cas $x = y$ et $x \neq y$. □

On peut enfin énoncer les lemmes qui entraînent le théorème 1, et devront être démontrés en même temps que celui-ci.

Lemme 1 (Stabilité, renommages)**.** *Si* $M \equiv_\alpha N$ *et* $\mathsf{fv}(M); \mathsf{fv}(N) \vdash \rho_1 \equiv_\alpha \rho_2$ *alors*

$$M[\rho_1] \equiv_\alpha N[\rho_2].$$

Lemme 2 (Fonctorialité, renommages)**.** $M \equiv_\alpha M[\mathtt{id}]$ *et* $M[\rho_1][\rho_2] \equiv_\alpha M[\rho_1 \circ \rho_2]$.

Démonstration. On démontre par une seule induction forte sur la taille de M à la fois la transitivité de l'équivalence α, qui complète le théorème 1, et les lemmes 1 et 2. En conséquence des propriétés 12 et 13, l'hypothèse d'induction nous donne accès à ces résultats pour tout terme obtenu à partir d'un sous-terme de M par une suite quelconque d'équivalences α et de renommages.

On commence par démontrer un résultat élémentaire mais essentiel pour le reste de la preuve. Pour tout préterme M' de taille strictement inférieure à celle de M, et tous noms x, y, z avec y n'appartenant pas à $\mathsf{fv}(M')$, on a

$$M'[x\backslash y][y\backslash z] \equiv_\alpha M'[x\backslash y \circ y\backslash z] = M'[y\backslash z, x\backslash z] \equiv_\alpha M'[x\backslash z]. \tag{10.1}$$

C'est une conséquence de nos hypothèses d'induction : on utilise la transitivité de l'équivalence α pour chaîner les égalités, la fonctorialité pour la première équivalence et la stabilité avec la règle CongProd et l'hypothèse $y \notin \mathsf{fv}(M')$ pour la deuxième.

On démontre ensuite chaque résultat en raisonnant par cas sur M. On va détailler, pour chaque résultat, le cas des variables et surtout de l'abstraction, qui constistera en une série d'égalités chaînées à l'aide de la transitivité. Les cas du zéro, du successeur et de l'application sont routiniers, tandis que la récursion primitive est toujours une variante de l'abstraction.

1. On démontre que si $N_1 \equiv_\alpha M$ et $M \equiv_\alpha N_2$ alors $N_1 \equiv_\alpha N_2$.
   - Cas $M = x$: immédiat par inversion des hypothèses.
   - Cas $M = \mathtt{fun}(x.M')$: par inversion des hypothèses, on doit avoir, pour $i \in \{1, 2\}$, un préterme $N_i = \mathtt{fun}(x_i.N_i')$ tel que $N_i'[x_i \backslash a_i] \equiv_\alpha M'[x \backslash a_i]$ avec $a_i = \nu(M', N_i')$. Par les propriétés 12 et 13, on sait que les tailles de M' et des N_i' sont strictement inférieures à celle de M, tout comme celles de tous leurs renommages. On doit montrer $N_1'[x_1 \backslash c] \equiv_\alpha N_2'[x_2 \backslash c]$ où $c = \nu(N_1', N_2')$. On va montrer $N_i'[x_i \backslash c] \equiv_\alpha M'[x \backslash c]$ puis conclure par transitivité. On a

$$\begin{aligned} N_i'[x_i \backslash c] &\equiv_\alpha N_i'[x_i \backslash a_i][a_i \backslash c] && \text{par l'équation (10.1)} \\ &\equiv_\alpha M'[x \backslash a_i][a_i \backslash c] && \text{par stabilité} \\ &\equiv_\alpha M'[x \backslash c] && \text{par l'équation (10.1).} \end{aligned}$$

2. On démontre que si $M \equiv_\alpha N$ et $\mathsf{fv}(M); \mathsf{fv}(N) \vdash \rho_1 \equiv_\alpha \rho_2$ alors $M[\rho_1] \equiv_\alpha N[\rho_2]$.
   - Cas $M = x$: on doit démontrer que si $\{x\}; \{x\} \vdash \rho_1 \equiv_\alpha \rho_2$ alors $x[\rho_1] = x[\rho_2]$. On procède par une nouvelle induction sur la dérivation d'équivalence α. Le fait que ρ_1 et ρ_2 soient des renommages rend la preuve possible. Par exemple, pour le cas CongBind, de $M \equiv_\alpha N$ on déduira que $M = N = y$ pour un certain nom y. Les cas CongProd et CongComp requièrent chacun encore une nouvelle induction sur le nombre de liaisons.
   - Cas $M = \mathtt{fun}(x_1.M_1)$: par inversion de l'hypothèse $M \equiv_\alpha N$, on doit avoir $N = \mathtt{fun}(x_2.M_2)$ avec $M_1[x_1 \backslash y] \equiv_\alpha M_2[x_2 \backslash y]$ avec $y = \nu(M_1, M_2)$. On doit montrer $M_1[\rho_1, x_1 \backslash x_1'][x_1' \backslash z] \equiv_\alpha M_2[\rho_2, x_2 \backslash x_2'][x_2' \backslash z]$ où on a $x_i' = \nu(M_i, \rho_i)$ et $z = \nu(M_1[\rho_1, x_1 \backslash x_1'], M_2[\rho_2, x_2 \backslash x_2'])$. On a

$$\begin{aligned} M_i[\rho_i, x_i \backslash x_i'][x_i' \backslash z] &\equiv_\alpha M_i[(\rho_i, x_i \backslash x_i') \circ x_i' \backslash z] && \text{par fonctorialité} \\ &\equiv_\alpha M_i[\rho_i, x_i \backslash z] && \text{par stabilité} \\ &\equiv_\alpha M_i[x_i \backslash y][\rho_i, y \backslash z] && \text{par fonctorialité} \end{aligned}$$

   et on conclue par stabilité puisque $M_1[x_1 \backslash y] \equiv_\alpha M_2[x_2 \backslash y]$ et

$$\mathsf{fv}(M_1[x_1 \backslash y]); \mathsf{fv}(M_2[x_2 \backslash y]) \vdash \rho_1, y \backslash z \equiv_\alpha \rho_2, y \backslash z.$$

3. On démontre que $M \equiv_\alpha M[\mathtt{id}]$ par les mêmes méthodes que précédemment ; en particulier, le cas de $M = \mathtt{fun}(x.M')$ est traité en utilisant équation (10.1). En ce qui concerne $M[\rho_1][\rho_2] \equiv_\alpha M[\rho_1 \circ \rho_2]$, on utilise les mêmes techniques que précédemment, sachant que le cas des variables correspond à la propriété 14.

□

10.1.4. Termes et substitutions

Le théorème 1 assure l'existence de l'ensemble quotient des termes, c'est-à-dire des classes d'équivalence de prétermes.

Définition 20. L'ensemble $\mathsf{Term}_\mathcal{T}$ des *termes* est défini comme le quotient des prétermes par l'équivalence α, de même pour l'ensemble $\mathsf{Sub}_\mathcal{T}$ des *substitutions*.

Pour que la notion de terme et de substitution soit utile, il faut que toutes les opérations définies précédemment sur les prétermes passent au quotient, autrement dit qu'elles ne distinguent pas les termes ou substitutions α-convertibles, à la manière de la propriété 13 qui exprime que la taille passe au quotient, ou du résultat suivant.

Propriété 15 (Passage au quotient des renommages)**.** *Si* $M \equiv_\alpha N$ *et* $\rho_1 \equiv_\alpha \rho_2$ *alors* $M[\rho_1] \equiv_\alpha N[\rho_2]$.

Démonstration. C'est un corollaire immédiat du lemme 1 et de la propriété 11. □

Pour généraliser ce résultat des renommages aux substitutions arbitraires, on doit démontrer les analogues du lemme 1 et de la deuxième partie du lemme 2, exprimés sous une forme suffisamment générale pour se prêter à un argument inductif.

Lemme 3. $M[\sigma][\varphi] \equiv_\alpha M[\sigma \circ \varphi]$.

Lemme 4. *Si* $M[\rho] \equiv_\alpha N$ *et* $\mathsf{fv}(M[\rho]); \mathsf{fv}(N) \vdash \sigma \equiv_\alpha \varphi$ *alors* $M[\rho][\sigma] \equiv_\alpha N[\varphi]$.

Démonstration. On démontre simultanément les lemmes 3 et 4 par une induction sur M.

- Démonstration du lemme 4 :
  - Cas $M = x$: par inversion de l'hypothèse d'équivalence α, N est nécessairement un certain nom $y = x[\rho]$. On doit montrer $y[\sigma] \equiv_\alpha y[\varphi]$. On procède par une nouvelle induction sur la dérivation de $\{y\}; \{y\} \vdash \sigma \equiv_\alpha \varphi$.
  - Cas $M = \mathtt{fun}(x.M')$: on a $M[\rho] = \mathtt{fun}(x'.M'[\rho, x\backslash x'])$ avec $x' \# M, \rho$. Par inversion de l'hypothèse d'équivalence α, on doit avoir $N = \mathtt{fun}(y.N')$ avec $M'[\rho, x\backslash a] \equiv_\alpha M'[\rho, x\backslash x'][x'\backslash a] \equiv_\alpha N'[y\backslash a]$. On applique l'hypothèse d'induction pour déduire
    $$M'[\rho, x\backslash a][\sigma, a\backslash b] \equiv_\alpha N'[y\backslash a][\varphi, a\backslash b]$$
    et on conclue après de nouvelles utilisations des hypothèses d'induction.

□

Corollaire 4 (Stabilité)**.** *Si* $M \equiv_\alpha N$ *et* $\sigma \equiv_\alpha \varphi$ *alors* $M[\sigma] \equiv_\alpha N[\varphi]$.

Démonstration. Prendre $\mathtt{id}$ pour ρ dans l'énoncé du lemme 4. □

10.1.5. L'équivalence α en pratique

Les résultats précédents permettent de se rendre compte de la difficulté qu'il peut y avoir à traiter de façon à la fois rigoureuse et élémentaire la gestion des variables libres et liées. Ce problème est fréquemment passé sous silence dans les cours de mathématiques qui introduisent la notion de variable liée. Pourtant, des solutions satisfaisantes sont disponibles, et elles sont satisfaisantes quoique pas exemptes de défauts. On peut par exemple adopter une syntaxe moins lisible mais plus disciplinée, voir l'exercice 13, ou bien se placer dans un cadre mathématique où la notion de variable fraîche reçoit un statut à part entière, comme les ensembles nominaux [5].

Dans les faits, lorsqu'on travaille sur papier plutôt que sur machine, on adopte la convention dite *de Barendregt*, du nom de l'auteur de l'ouvrage de référence sur le λ-calcul [1]. L'idée est la suivante.

> Au moment de manipuler un terme M dans une définition ou une preuve, on s'assure par un renommage approprié de choisir un représentant de la classe d'équivalence de M dont toutes les variables liées sont distinctes des variables libres utilisées à ce point du texte.

Cette convention est difficile à rendre mathématiquement précise puisque la notion de variables libres “utilisées à ce point du texte” est informelle, sauf à introduire (beaucoup) d'outillage mathématique. Cependant, elle permet dans les faits de ne plus jamais avoir à se soucier de problèmes de capture, et en particulier de *prétendre que la substitution naïve coïncide avec la vraie substitution*. L'usage a montré qu'elle n'était pas source d'erreur, du moins tant que l'on traite de langages où la structure de la liaison est suffisamment simple. Ce sera le cas de tous les langages traités dans ces notes, et pour cette raison on adopte à partir de ce point la convention de Barendregt sans rechigner.

10.2. Types et calcul

10.2.1. Équivalence β

La relation d'équivalence α décrite précédemment exprime une forme très simple d'égalité de programmes : deux programmes qui ne diffèrent que par le nom de leurs variables liées sont essentiellement les mêmes. On va maintenant introduire une notion d'égalité plus dynamique, au sens où elle caractérise les programmes qui “s'exécutent de façon similaire” ou, plus précisément, “calculent le même résultat”.

Définition 21 (Clôture $\mathcal{T}$-contextuelle d'une relation)**.** Si R est une relation n-aire sur les termes de $\mathcal{T}$, sa *clôture $\mathcal{T}$-contextuelle*, notée $\mathcal{T}[R]$, est la relation n-aire sur les termes de $\mathcal{T}$ définie inductivement par les règles données à la figure 10.4.

Définition 22 ($\mathcal{T}$-congruence)**.** Une *$\mathcal{T}$-congruence* R est une relation d'équivalence sur les termes de $\mathcal{T}$ telle que $\mathcal{T}[R]$ soit incluse dans R.

$$\frac{(M_1,\ldots,M_n)\in R}{(M_1,\ldots,M_n)\in\mathcal{T}[R]} \qquad \frac{(M_1,\ldots,M_n)\in\mathcal{T}[R] \qquad (N_1,\ldots,N_n)\in\mathcal{T}[R]}{(\mathtt{app}(M_1,N_1),\ldots,\mathtt{app}(M_n,N_n))\in\mathcal{T}[R]}$$

$$\frac{(M_1,\ldots,M_n)\in\mathcal{T}[R]}{(\mathtt{fun}(x.M_1),\ldots,\mathtt{fun}(x.M_n))\in\mathcal{T}[R]} \qquad \frac{(M_1,\ldots,M_n)\in\mathcal{T}[R]}{(\mathtt{suc}(M_1),\ldots,\mathtt{suc}(M_n))\in\mathcal{T}[R]}$$

$$\frac{(M_1,\ldots,M_n)\in\mathcal{T}[R] \qquad (N_1,\ldots,N_n)\in\mathcal{T}[R] \qquad (P_1,\ldots,P_n)\in\mathcal{T}[R]}{(\mathtt{fold}_{\mathtt{Nat}}(M_1,N_1,(x,y).P_1),\ldots,\mathtt{fold}_{\mathtt{Nat}}(M_n,N_n,(x,y).P_n))\in\mathcal{T}[R]}$$

FIG. 10.4. : Clôture $\mathcal{T}$-contextuelle d'une relation R

Une $\mathcal{T}$-congruence est une relation d'équivalence qui autorise le raisonnement local : pour montrer que deux termes sont équivalents, il suffit de montrer qu'ils ne diffèrent que par deux sous-termes équivalents.

Propriété 16. *Soit R une $\mathcal{T}$-congruence. Si $(M,N)\in R$ alors $(P[x\backslash M],P[x\backslash N])\in R$.*

Définition 23 (β-équivalence)**.** La relation de *β-équivalence* entre termes, notée $M\equiv_\beta N$, est la plus petite $\mathcal{T}$-congruence contenant les règles suivantes.

$$\mathtt{app}(\mathtt{fun}(x.M),N)\equiv_\beta M[x\backslash N] \tag{10.2}$$
$$\mathtt{fold}_{\mathtt{Nat}}(\mathtt{zero},N,(x,y).P)\equiv_\beta N \tag{10.3}$$
$$\mathtt{fold}_{\mathtt{Nat}}(\mathtt{suc}(M),N,(x,y).P)\equiv_\beta P[x\backslash M,y\backslash\mathtt{fold}_{\mathtt{Nat}}(M,N,(x,y).P)] \tag{10.4}$$

La β-équivalence constitue l'un des objets d'étude fondamentaux de la théorie des langages de programmation. Pour expliquer cette relation, il est utile de distinguer dans la syntaxe de $\mathcal{T}$ deux familles d'opérations distinctes.

- Les *formes d'introduction* permettent de créer de nouvelles données. Elles comprennent l'abstraction, le zéro et le successeur.
- Les *formes d'élimination* permettent d'utiliser des données existantes. Elles comprennent l'application et la récursion primitive.

Les équations (10.2) à (10.4) expriment qu'une forme d'introduction imbriquée sous une forme d'élimination est équivalente à un terme plus simple. Par exemple, l'application d'une abstraction à un argument est équivalente au corps de l'abstraction où le paramètre a été substitué par l'argument, comme exprimé par l'équation (10.2).

Remarque 11. La variable n'est ni une forme d'introduction, ni une forme d'élimination. D'un point de vue théorique, la variable est une opération qu'on pourrait qualifier de *logistique*, au sens où elle n'agit pas par elle même mais fait fonctionner le reste du langage en interagissant avec la substitution.

10.2.2. Types

La syntaxe de $\mathcal{T}$ telle que nous l'avons définie n'impose aucune contrainte sur l'enchâssement des opérations. En particulier, une forme d'élimination peut rencontrer une forme d'introduction incompatible, au sens où aucune des équations (10.2) à (10.4) ne s'applique. C'est le cas par exemple d'un terme de la forme $\texttt{app}(\texttt{zero}, N)$ ou encore $\texttt{fold}_{\texttt{Nat}}(\texttt{fun}(z.M), N, (x, y).P)$. Il est raisonnable de considérer de tels termes comme entièrement dépourvus de sens.

Pour ne pas avoir à se soucier de tels termes, on va raffiner la syntaxe de $\mathcal{T}$ en classifiant les termes en fonction de la nature de leur résultat. Le lecteur ou la lectrice aura sans doute reconnu la notion de *système de types*, essentielle à de nombreux langages de programmation. L'idée est d'associer aux termes M des *types* A, auxquels on peut penser comme à des formules logiques qui classifient le résultat construit par M. Les types de $\mathcal{T}$ sont très simples : on ne peut y construire que des entiers, de type $\texttt{Nat}$, et des fonctions, de type $A \to B$, où A et B sont des types. On écrira que M *habite le type* A, et on notera $M : A$, lorsque A est un type possible pour M.

La relation $M : A$ est définie inductivement par un ensemble de règles. Essayons d'imaginer à quoi celles-ci peuvent ressembler. Les règles traitant du zéro, du successeur et de l'application semblent s'écrire d'elles-même.

$$\frac{}{\texttt{zero} : \texttt{Nat}} \qquad \frac{M : \texttt{Nat}}{\texttt{suc}(M) : \texttt{Nat}} \qquad \frac{M : A \to B \qquad N : A}{\texttt{app}(M, N) : B}$$

Les cas de l'abstraction et de la récursion primitive sont nettement moins évidents.

$$\frac{M : B \ ?!}{\texttt{fun}(x.M) : A \to B} \qquad \frac{M : \texttt{Nat} \qquad N : A \qquad P : A \ ?!}{\texttt{fold}_{\texttt{Nat}}(M, N, (x, y).P)}$$

Considérons le cas de l'abstraction, celui de la récursion primitive étant similaire. L'abstraction $\texttt{fun}(x.M)$ est une fonction et doit donc nécessairement habiter un type de la forme $A \to B$. Pour montrer que c'est le cas, il faut pouvoir montrer que son corps M habite le type B. Mais ce n'est pas tout : les occurrences à x dans M doivent être considérées comme habitant le type A. Or, notre jugement $M : A$ est incapable d'exprimer cette hypothèse. Symétriquement, on ne voit pas non plus comment traiter les variables.

$$\frac{}{x : ?!}$$

Pour régler ce problème, on est naturellement amenés à considérer un jugement de typage dit *hypothétique* : un terme M habite le type A sous l'hypothèse que ses variables libres $x_1, \ldots, x_k$ habitent les types $B_1, \ldots, B_k$, ce qu'on écrit $x_1 : B_1, \ldots, x_k : B_k$. On appelle cette liste d'hypothèses un *contexte de typage*, et on écrira

$$\Gamma \vdash M : A$$

pour exprimer que M habite le type A *sous le contexte* Γ, contexte qui prescrit les types habités par les variables libres de M. Les contextes sont soumis à une restriction importante : une même variable apparaît **au plus** une fois dans un contexte de typage donné. Ainsi, $x : \texttt{Nat}, y : \texttt{Nat} \to \texttt{Nat}, x : \texttt{Nat}$ n'est pas un contexte valide.

$\mathsf{Type}_{\mathcal{T}} \ni A, B, C ::=$		**Types**
	$\mid \mathtt{Nat}$	Type des entiers naturels
	$\mid A \to B$	Type fonctionnel
$\mathsf{Con}_{\mathcal{T}} \ni \Gamma, \Delta ::=$		**Contextes**
	$\mid \cdot$	Contexte vide
	$\mid \Gamma, x : A$	Déclaration de variable

FIG. 10.5. : Types de $\mathcal{T}$

$$\boxed{\Delta \supseteq \Gamma} \qquad \frac{}{\Delta \supseteq \cdot} \qquad \frac{\Delta \supseteq \Gamma}{\Delta, x : A \supseteq \Gamma} \qquad \frac{\Delta \supseteq \Gamma}{\Delta, x : A \supseteq \Gamma, x : A}$$

FIG. 10.6. : Affaiblissement des contextes de $\mathcal{T}$

Définition 24 (Types de $\mathcal{T}$)**.** La syntaxe des types et des contextes de $\mathcal{T}$ est définie à la figure 10.5.

Définition 25 (Affaiblissement)**.** On dit qu'un contexte Δ peut être *affaibli* en un contexte Γ, ce qu'on note $\Delta \supseteq \Gamma$, lorsqu'on peut obtenir Γ en effaçant un nombre fini de liaisons dans Δ. Formellement, il s'agit du jugement inductif défini à la figure 10.6.

Propriété 17. *Le jugement d'affaiblissement est réflexif et transitif.*

Démonstration. Par des inductions de routine. □

Définition 26 (Typage des termes de $\mathcal{T}$)**.** Le typage des termes est défini par les jugements inductifs présentés à la figure 10.7. On écrit $\mathcal{T}(\Gamma; A)$ pour l'ensemble des termes habitant le type A sous le contexte Γ, et on abrège $\cdot \vdash M : A$ en $M : A$.

La règle traitant du cas de la variable utilise le jugement auxiliaire dit d'*affaiblissement*, noté $\Gamma \supseteq \Delta$, qui exprime que le contexte Δ peut être obtenu à partir de Γ en oubliant certaines variables. Ce n'est pas la seule façon d'exprimer cette idée.

Il faut insister encore sur le fait que toute variable apparaît au plus une fois dans un contexte. C'est une supposition implicite des deux règles mettant en jeu la liaison, c'est-à-dire l'abstraction et la récursion primitive. Ainsi, la prémisse de la règle

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \mathtt{fun}(x.M) : A \to B}$$

présuppose que x n'apparaît pas dans Γ. Cependant, puisque $\mathtt{fun}(x.M)$ est un terme et donc considéré modulo équivalence α, on peut toujours renommer x en une variable qui n'apparaît pas dans Γ.

$$\boxed{\Gamma \vdash M : A} \qquad \frac{\Gamma \supseteq x : A}{\Gamma \vdash \mathtt{var}(x) : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \mathtt{fun}(x.M) : A \to B}$$

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash \mathtt{app}(M, N) : B} \qquad \frac{}{\Gamma \vdash \mathtt{zero} : \mathtt{Nat}} \qquad \frac{\Gamma \vdash M : \mathtt{Nat}}{\Gamma \vdash \mathtt{suc}(M) : \mathtt{Nat}}$$

$$\frac{\Gamma \vdash M : \mathtt{Nat} \qquad \Gamma \vdash N : A \qquad \Gamma, x : \mathtt{Nat}, y : A \vdash P : A}{\Gamma \vdash \mathtt{fold}_{\mathtt{Nat}}(M, N, (x, y).P) : A}$$

FIG. 10.7. : Typage des termes de $\mathcal{T}$

$$\boxed{\Gamma \vdash \sigma : \Delta} \qquad \frac{\Delta \supseteq \Gamma}{\Delta \vdash \mathtt{id} : \Gamma} \qquad \frac{\Gamma \vdash \sigma : \Delta \qquad \Gamma \vdash M : A}{\Gamma \vdash \sigma, x \backslash M : \Delta, x : A}$$

FIG. 10.8. : Typage des substitutions de $\mathcal{T}$

En plus de typer les termes, il est également commode de disposer d'un jugement de typage classifiant les substitutions. Une substitution σ est formée de plusieurs termes, nommés qui plus est, il est donc naturel de la classifier par un contexte de typage.

Définition 27 (Typages des substitutions de $\mathcal{T}$)**.** Le jugement de typage des substitutions est défini à la figure 10.8. On écrit $\mathcal{T}(\Delta; \Gamma)$ pour l'ensemble des substitutions habitant le contexte Γ sous le contexte Δ.

On démontre maintenant un lemme fondamental, caractéristique des systèmes de types, qui exprime la compatibilité entre le typage et la substitution.

Définition 28 (Concaténation de contextes)**.** L'opération de *concaténation* associe à deux contextes Γ et Δ leur *produit*, noté Γ, Δ, défini par récurrence sur Δ comme suit.

$$\begin{aligned} -, = : \mathsf{Con}_{\mathcal{T}} \times \mathsf{Con}_{\mathcal{T}} &\to \mathsf{Con}_{\mathcal{T}} \\ \Gamma, \cdot &= \Gamma \\ \Gamma, (\Delta, x : A) &= (\Gamma, \Delta), x : A \end{aligned}$$

Lemme 5 (Affaiblissement, termes)**.** *Si* $\Gamma \vdash M : A$ *et* $\Delta \supseteq \Gamma$ *alors* $\Delta \vdash M : A$.

Lemme 6 (Affaiblissement, substitutions)**.** *Si* $\Gamma \vdash \sigma : \Theta$ *et* $\Delta \supseteq \Gamma$ *alors* $\Delta \vdash \sigma : \Theta$.

Propriété 18. *Si* $\Delta \vdash \sigma : \Gamma$ *et* $\Gamma \supseteq x : A$ *alors* $\Delta \vdash x[\sigma] : A$.

Lemme 7 (Substitution)**.** *Si* $\Gamma \vdash M : A$ *et* $\Delta \vdash \sigma : \Gamma$ *alors* $\Delta \vdash M[\sigma] : A$.

Démonstration. Par induction sur la dérivation de typage $\Gamma \vdash M : A$. On ne détaille que les trois premiers cas.

- Cas $\dfrac{\Gamma \supseteq x : A}{\Gamma \vdash x : A}$: c'est la propriété 18.
- Cas $\dfrac{\Gamma \vdash M_1 : A \to B \qquad \Gamma \vdash M_2 : A}{\Gamma \vdash \mathtt{app}(M_1, M_2) : B}$: on a $\mathtt{app}(M_1, M_2)[\sigma] = \mathtt{app}(M_1[\sigma], M_2[\sigma])$ et donc
$$\frac{\Delta \vdash M_1[\sigma] : A \to B \qquad \Delta \vdash M_2[\sigma] : A}{\Delta \vdash \mathtt{app}(M_1[\sigma], M_2[\sigma]) : B}$$
où les deux prémisses sont les hypothèses d'induction.
- Cas $\dfrac{\Gamma, x : A \vdash M_1 : B}{\Gamma \vdash \mathtt{fun}(x.M_1) : A \to B}$: on a $\mathtt{fun}(x.M_1)[\sigma] = \mathtt{fun}(x.M_1[\sigma, x\backslash x])$ et donc
$$\frac{\Delta, x : A \vdash M_1[\sigma, x\backslash x] : B}{\Delta \vdash \mathtt{fun}(x.M_1[\sigma, x\backslash x]) : B}$$
où la prémisse est obtenue par l'hypothèse d'induction, appliquée à
$$\frac{\Delta, x : A \vdash \sigma : \Gamma \qquad \Delta, x : A \vdash x : A}{\Delta, x : A \vdash (\sigma, x\backslash x) : \Gamma, x : A}$$
où la prémisse de gauche est obtenue par le lemme 6.

□

Remarque 12. Le cas de l'abstraction dans la preuve du lemme 7 exploite la convention de Barendregt pour traiter la substitution de façon naïve.

10.2.3. Canonicité et modèle ensembliste

Canonicité

Ce chapitre s'ouvre en affirmant que les programmes écrits en $\mathcal{T}$ ont un comportement très discipliné, ce qu'exprime le théorème qui suit.

Théorème 2 (Canonicité)**.** *Si $M : \mathtt{Nat}$ alors il existe un unique entier k tel que*
$$M \equiv_\beta \underline{k}.$$

Démontrer ce théorème est plus difficile que les précédents. Il est instructif d'essayer par une induction directe sur la dérivation de typage. Très formellement, on tente donc de démontrer la propriété
$$\forall \Gamma \in \mathsf{Con}_{\mathcal{T}}, \forall A \in \mathsf{Type}_{\mathcal{T}}, \forall M \in \mathcal{T}(\Gamma; A), \Gamma = \cdot \wedge A = \mathtt{Nat} \Rightarrow \exists! k \in \mathbb{N}, M \equiv_\beta \underline{k}.$$

Tout d'abord, remarquons que la plupart des cas n'ont pas besoin d'être considérés parce qu'ils sont incompatibles avec nos hypothèses. Par exemple, la règle de la variable ne s'applique que lorsque le contexte Γ n'est pas vide, ce qui est absurde puisque $\Gamma = \cdot$ par hypothèse. De même, la règle de l'abstraction ne s'applique que lorsque le type A est un type fonctionnel, ce qui est aussi absurde puisque $A = \texttt{Nat}$ par hypothèse.

Considérons ensuite le cas du zéro et du successeur. Pour le zéro, on choisit $k = 0$, puisque $\texttt{zero} \equiv_\beta \underline{0}$ par réflexivité. En revanche, on ne voit pas comment prouver l'unicité. Pour le successeur, l'hypothèse d'induction donne un unique k' tel que $M \equiv_\beta \underline{k'}$; on choisit $k = k' + 1$, et on a bien $\texttt{suc}(M) \equiv_\beta \underline{k' + 1} = \texttt{suc}(\underline{k'})$ par congruence. Là encore l'unicité semble rester inaccessible.

Enfin, les cas des formes d'élimination (application et récursion primitive) sont hautement problématiques et nous empêchent définitivement de compléter la preuve, même restreinte à l'existence. Considérons le cas de l'application. Étant donné nos hypothèses sur Γ et A, la règle doit avoir la forme qui suit.

$$\frac{\cdot \vdash M_1 : A \to \texttt{Nat} \qquad \cdot \vdash M_2 : A}{\cdot \vdash \texttt{app}(M_1, M_2) : \texttt{Nat}}$$

Le problème est que nos hypothèses d'induction ne s'appliquent qu'à des termes de type $\texttt{Nat}$, et donc ne fournissent aucune information au sujet de M_1 et M_2. Nous ne pouvons donc pas traiter ce cas. Le même problème se pose pour la récursion primitive.

Il faudrait donc généraliser notre hypothèse d'induction pour traiter des types fonctionnels. Mais pour traiter les types fonctionnels, il faudra traiter la règle de l'abstraction, dont la prémisse implique un contexte qui n'est jamais vide. Il faut donc également généraliser l'hypothèse d'induction pour traiter des contextes non vides. Enfin, il faut trouver un argument pour démontrer l'unicité. Pour satisfaire toutes ces contraintes, on va utiliser une méthode très souple, qui est un outil standard en sémantique des langages de programmation : on va bâtir un *modèle* de $\mathcal{T}$.

Informellement, un modèle de $\mathcal{T}$ définit une interprétation de chaque type, contexte et terme typé du langage par un objet mathématique. La nature exacte des objets mathématiques considérés dépend du modèle, et en faisant varier celle-ci, on peut déduire diverses conséquences sur le langage. De plus, l'interprétation des termes doit être cohérente avec celle des types et contextes, et doit respecter certaines contraintes :

1. elle doit être *fonctorielle*, c'est-à-dire commuter avec la substitution ;
2. elle doit être *correcte*, c'est-à-dire égaliser la relation d'équivalence considérée.

Dans cette section, nos modèles seront corrects vis-à-vis de l'équivalence β, au sens où l'équivalence $M \equiv_\beta N$ entraîne l'égalité des interprétations de M et N.

Modèle syntaxique

La façon la plus grossière de construire un modèle est simplement de quotienter la syntaxe par la relation d'équivalence considérée, ici l'équivalence β. Pour cela, on a besoin de quelques définitions auxiliaires.

Définition 29. Un *environnement* définissant un contexte Γ est un élément de $\mathcal{T}(\cdot;\Gamma)$.

Définition 30. On écrit $\Gamma \vdash M \equiv_\beta N : A$ lorsque $\Gamma \vdash M : A$ et $\Gamma \vdash N : A$ avec $M \equiv_\beta N$. Cette équivalence s'étend aux substitutions de manière structurelle : deux substitutions sont équivalentes modulo β lorsqu'elles lient les mêmes variables et que les termes liés sont équivalents, comme exprimé par le jugement inductif ci-dessous.

$$\boxed{\Delta \vdash \sigma \equiv_\beta \varphi : \Gamma} \qquad \frac{\Delta \supseteq \Gamma}{\Delta \vdash \mathtt{id} \equiv_\beta \mathtt{id} : \Gamma} \qquad \frac{\Delta \vdash \sigma \equiv_\beta \varphi : \Gamma \qquad \Delta \vdash M \equiv_\beta N : A}{\Delta \vdash \sigma, x\backslash M \equiv_\beta \varphi, x\backslash N : \Gamma, x : A}$$

Lemme 8. *Si* $\Gamma \vdash M \equiv_\beta N : A$ *et* $\Delta \vdash \sigma \equiv_\beta \varphi : \Gamma$ *alors* $\Delta \vdash M[\sigma] \equiv_\beta N[\varphi] : A$.

Démonstration. Par une induction de routine, en exploitant le fait que l'équivalence β est (par définition) une $\mathcal{T}$-congruence. □

Définition 31 (Modèle syntaxique)**.** Le modèle syntaxique de $\mathcal{T}$ interprète un terme $\Gamma \vdash M : A$ par sa classe d'équivalence modulo β, qu'on note $[\![\Gamma \vdash M : A]\!]_{\mathsf{Syn}}$.

Dans ce qui suit, on écrira $[\![A]\!]_{\mathsf{Syn}}$ pour l'ensemble des termes clos de type A quotientés par l'équivalence β. En particulier, si $\vdash M : A$ alors $[\![M]\!]_{\mathsf{Syn}} \in [\![A]\!]_{\mathsf{Syn}}$. On écrira également $[\![\Gamma]\!]_{\mathsf{Syn}}$ pour l'ensemble des environnements définissant Γ quotientés par l'équivalence β telle qu'exprimée par la définition 30.

La fonctorialité du modèle correspond ici au lemme 8, et la correction est immédiate par définition. Toutefois, ce modèle n'est pas utile en tant que tel, puisqu'on ne sait pas démontrer ses propriétés facilement. On va utiliser un modèle plus utile dans ce qui suit.

Modèle ensembliste

Propriété 19. *Soit X un ensemble, f_0 un élément de X, f_s une fonction de $\mathbb{N} \times X$ dans X. Alors il existe une unique fonction $\mathit{fold}_{\mathbb{N}}(f_0, f_s)$ de $\mathbb{N}$ dans X telle que*

$$\begin{aligned} \mathit{fold}_{\mathbb{N}}(f_0, f_s)(0) &= f_0, \\ \mathit{fold}_{\mathbb{N}}(f_0, f_s)(1+n) &= f_s(n, \mathit{fold}_{\mathbb{N}}(f_0, f_s)(n)). \end{aligned}$$

Chaque type A de $\mathcal{T}$ est interprété par un ensemble $[\![A]\!]_{\mathsf{Ens}}$ défini par récurrence sur la structure de A. De même pour les contextes, interprétés par des produits cartésiens.

Définition 32 (Modèle ensembliste, interprétation des types et contextes)**.** On interprète chaque type A (respectivement contexte Γ) par un ensemble $[\![A]\!]_{\mathsf{Ens}}$ (respectivement $[\![\Gamma]\!]_{\mathsf{Ens}}$) défini par récurrence sur sa structure comme suit.

$$\begin{aligned} [\![-]\!]_{\mathsf{Ens}} &: \mathsf{Type}_{\mathcal{T}} \to \mathbf{Ens} \\ [\![\mathtt{Nat}]\!]_{\mathsf{Ens}} &= \mathbb{N} \\ [\![A \to B]\!]_{\mathsf{Ens}} &= [\![A]\!]_{\mathsf{Ens}} \to [\![B]\!]_{\mathsf{Ens}} \end{aligned} \qquad \begin{aligned} [\![-]\!]_{\mathsf{Ens}} &: \mathsf{Con}_{\mathcal{T}} \to \mathbf{Ens} \\ [\![\cdot]\!]_{\mathsf{Ens}} &= \mathbf{1} \\ [\![\Gamma, x : A]\!]_{\mathsf{Ens}} &= [\![\Gamma]\!]_{\mathsf{Ens}} \times [\![A]\!]_{\mathsf{Ens}} \end{aligned}$$

Ci-dessus, la notation $\mathbf{1}$ désigne l'ensemble à un seul élément, dénoté $()$. On dénotera v un élément quelconque d'un $[\![A]\!]_{\mathsf{Ens}}$, et E un élément quelconque d'un $[\![\Gamma]\!]_{\mathsf{Ens}}$.

$$\left[\!\!\left[\frac{}{\Delta \supseteq \cdot}\right]\!\!\right]_{\mathsf{Ens}}(E) = ()$$

$$\left[\!\!\left[\frac{\Delta \supseteq \Gamma}{\Delta, x : A \supseteq \Gamma}\right]\!\!\right]_{\mathsf{Ens}}(E, v) = [\![\Delta \supseteq \Gamma]\!]_{\mathsf{Ens}}(E)$$

$$\left[\!\!\left[\frac{\Delta \supseteq \Gamma}{\Delta, x : A \supseteq \Gamma, x : A}\right]\!\!\right]_{\mathsf{Ens}}(E, v) = ([\![\Delta \supseteq \Gamma]\!]_{\mathsf{Ens}}(E), v)$$

FIG. 10.9. : Modèle ensembliste de $\mathcal{T}$: affaiblissement

Définition 33 (Modèle ensembliste, interprétation de l'affaiblissement)**.** On interprète une dérivation d'affaiblissement $\Delta \supseteq \Gamma$ par une fonction de $[\![\Delta]\!]_{\mathsf{Ens}}$ dans $[\![\Gamma]\!]_{\mathsf{Ens}}$. La figure 10.9 présente les clauses de l'interprétation.

Définition 34 (Modèle ensembliste, interprétation des termes et des substitutions)**.** On interprète une dérivation de typage $\Gamma \vdash M : A$ par une fonction de $[\![\Gamma]\!]_{\mathsf{Ens}}$ dans $[\![A]\!]_{\mathsf{Ens}}$. Similairement, on interprète une dérivation de typage $\Delta \vdash \sigma : \Gamma$ par une fonction de $[\![\Delta]\!]_{\mathsf{Ens}}$ dans $[\![\Gamma]\!]_{\mathsf{Ens}}$. La figure 10.10 présente les clauses de l'interprétation.

Remarque 13. Dans l'interprétation des variables de la figure 10.10, on a $[\![x : A]\!]_{\mathsf{Ens}} = [\![\cdot, x : A]\!]_{\mathsf{Ens}} = \mathbf{1} \times [\![A]\!]_{\mathsf{Ens}}$, et on utilise la deuxième projection $\pi_2 : \mathbf{1} \times [\![A]\!]_{\mathsf{Ens}} \to [\![A]\!]_{\mathsf{Ens}}$.

Lemme 9 (Fonctorialité)**.** $[\![\Delta \vdash M[\sigma] : A]\!]_{\mathsf{Ens}} = [\![\Gamma \vdash M : A]\!]_{\mathsf{Ens}} \circ [\![\Delta \vdash \sigma : \Gamma]\!]_{\mathsf{Ens}}$.

Démonstration. Par une induction de routine sur la dérivation de $\Gamma \vdash M : A$. □

Théorème 3 (Correction)**.** *Si* $\Gamma \vdash M \equiv_\beta N : A$ *alors* $[\![\Gamma \vdash M : A]\!]_{\mathsf{Ens}} = [\![\Gamma \vdash N : A]\!]_{\mathsf{Ens}}$.

Démonstration. Par induction sur la dérivation de $M \equiv_\beta N$. Les cas de la réflexivité, symétrie et transitivité sont immédiats par application des hypothèses d'induction, tout comme ceux correspondant à la clôture contextuelle. Les équations (10.3) et (10.4) sont conséquences immédiates de la propriété 19. Reste l'équation (10.2), pour laquelle on a

$$\begin{aligned}
&[\![\Gamma \vdash \mathtt{app}(\mathtt{fun}(x.M), N) : B]\!]_{\mathsf{Ens}}(E) \\
&= [\![\Gamma, x : A \vdash M : B]\!]_{\mathsf{Ens}}(E, [\![\Gamma \vdash N : A]\!]_{\mathsf{Ens}}(E)) && \text{par définition} \\
&= [\![\Gamma, x : A \vdash M : B]\!]_{\mathsf{Ens}}([\![\Gamma \vdash (\mathtt{id}, x\backslash N) : (\Gamma, x : A)]\!]_{\mathsf{Ens}}(E)) && \text{par définition} \\
&= [\![\Gamma \vdash M[x\backslash N] : B]\!]_{\mathsf{Ens}}(E) && \text{par le lemme 9.}
\end{aligned}$$

□

Corollaire 5. *Si* $\Delta \vdash \sigma \equiv_\beta \varphi : \Gamma$ *alors* $[\![\Delta \vdash \sigma : \Gamma]\!]_{\mathsf{Ens}} = [\![\Delta \vdash \varphi : \Gamma]\!]_{\mathsf{Ens}}$.

$$[\![-]\!]_{\mathsf{Ens}} : \mathcal{T}(\Gamma; A) \to [\![\Gamma]\!]_{\mathsf{Ens}} \to [\![A]\!]_{\mathsf{Ens}}$$

$$\left[\!\!\left[\frac{\Gamma \supseteq x : A}{\Gamma \vdash x : A} \right]\!\!\right]_{\mathsf{Ens}} (E) = \pi_2([\![\Gamma \supseteq x : A]\!]_{\mathsf{Ens}}(E))$$

$$\left[\!\!\left[\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash \mathtt{app}(M, N) : B} \right]\!\!\right]_{\mathsf{Ens}} (E) = [\![\Gamma \vdash M : A \to B]\!]_{\mathsf{Ens}}(E)(v) \quad \text{où } v = [\![\Gamma \vdash N : A]\!]_{\mathsf{Ens}}(E)$$

$$\left[\!\!\left[\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \mathtt{fun}(x.M) : A \to B} \right]\!\!\right]_{\mathsf{Ens}} (E) = v \mapsto [\![\Gamma, x : A \vdash M : B]\!]_{\mathsf{Ens}}(E, v)$$

$$\left[\!\!\left[\frac{}{\Gamma \vdash \mathtt{zero} : \mathtt{Nat}} \right]\!\!\right]_{\mathsf{Ens}} (E) = 0$$

$$\left[\!\!\left[\frac{\Gamma \vdash M : \mathtt{Nat}}{\Gamma \vdash \mathtt{suc}(M) : \mathtt{Nat}} \right]\!\!\right]_{\mathsf{Ens}} (E) = 1 + [\![\Gamma \vdash M : \mathtt{Nat}]\!]_{\mathsf{Ens}}(E)$$

$$\left[\!\!\left[\frac{\begin{array}{c}\Gamma \vdash M : \mathtt{Nat} \\ \Gamma \vdash N : A \qquad \Gamma, x : \mathtt{Nat}, y : A \vdash P : A\end{array}}{\Gamma \vdash \mathtt{fold}_{\mathtt{Nat}}(M, N, (x, y).P) : A} \right]\!\!\right]_{\mathsf{Ens}} (E) = \mathit{fold}_{\mathbb{N}}(f_0, f_s)([\![M]\!]_{\mathsf{Ens}}(E)) \quad \text{où } f_0 = [\![N]\!]_{\mathsf{Ens}}(E),\ f_s(m, w) = [\![P]\!]_{\mathsf{Ens}}((E, m), w)$$

$$[\![-]\!]_{\mathsf{Ens}} : \mathcal{T}(\Delta; \Gamma) \to [\![\Delta]\!]_{\mathsf{Ens}} \to [\![\Gamma]\!]_{\mathsf{Ens}}$$

$$\left[\!\!\left[\frac{\Delta \supseteq \Gamma}{\Delta \vdash \mathtt{id} : \Gamma} \right]\!\!\right]_{\mathsf{Ens}} (E) = [\![\Delta \supseteq \Gamma]\!]_{\mathsf{Ens}}(E)$$

$$\left[\!\!\left[\frac{\Delta \vdash \sigma : \Gamma \qquad \Delta \vdash M : A}{\Delta \vdash (\sigma, x \backslash M) : (\Gamma, x : A)} \right]\!\!\right]_{\mathsf{Ens}} (E) = ([\![\sigma]\!]_{\mathsf{Ens}}(E), [\![M]\!]_{\mathsf{Ens}}(E))$$

FIG. 10.10. : Modèle ensembliste de $\mathcal{T}$: termes et substitutions

Adéquation

Il reste à prouver que le modèle ensembliste associe à tout terme clos M de type `Nat` un entier $[\![M]\!]_{\mathsf{Ens}}$ tel que $\underline{[\![M]\!]_{\mathsf{Ens}}} \equiv_\beta M$. Une preuve directe échoue pour les mêmes raisons que celles expliquées au début de cette section. Pour y remédier, on va généraliser notre énoncé via un outil standard appelé *relation logique*.

De façon générale, une relation logique est une relation définie par récurrence sur les types d'un langage de programmation. Ici, il s'agira de relier les éléments du modèle ensembliste et des classes d'équivalence de terme modulo β, i.e., des éléments du modèle syntaxique. Le point clef est la définition au type des fonctions, qui va être conçue pour fournir une propriété assez forte pour permettre le raisonnement par induction.

Définition 35. On définit la relation $\sqsubseteq_A \subseteq [\![A]\!]_{\mathsf{Ens}} \times [\![A]\!]_{\mathsf{Syn}}$ par récurrence sur A.

$$\begin{aligned} \sqsubseteq_{\mathtt{Nat}} &= \{(n, M) \mid [\![\underline{n}]\!]_{\mathsf{Syn}} = M\} \\ \sqsubseteq_{A\to B} &= \{(f, M) \mid \forall v \in [\![A]\!]_{\mathsf{Ens}}, \forall N \in [\![A]\!]_{\mathsf{Syn}}, v \sqsubseteq_A N \Rightarrow f(v) \sqsubseteq_B \mathtt{app}(M, N)\} \end{aligned}$$

Dans la première clause ci-dessus, l'égalité $[\![\underline{n}]\!]_{\mathsf{Syn}} = M$ concerne des éléments de $[\![A]\!]_{\mathsf{Syn}}$, qui sont des classes d'équivalence de termes clos modulo équivalence β. Autrement dit, cette égalité exprime que tout terme dans la classe d'équivalence M est β-équivalent à $\underline{n}$. Dans la seconde clause, l'opération $\mathtt{app}(M, N)$ est bien définie sur les classes d'équivalence puisque l'équivalence β est une $\mathcal{T}$-congruence.

On écrira qu'un élément v de $[\![A]\!]_{\mathsf{Ens}}$ *réalise* un terme clos M de type A modulo β, ce qu'on dénote $v \sqsubseteq_A M$, lorsque $(v, M) \in \sqsubseteq_A$.

La relation d'adéquation s'étend aux environnements liaison-à-liaison.

Définition 36. On définit la relation $\sqsubseteq_\Gamma \subseteq [\![\Gamma]\!]_{\mathsf{Ens}} \times [\![\Gamma]\!]_{\mathsf{Syn}}$ par récurrence sur Γ.

$$\begin{aligned} \sqsubseteq_\cdot &= \{((), \mathtt{id})\} \\ \sqsubseteq_{\Gamma, x:A} &= \{((E, v), (\sigma, x \backslash M)) \mid E \sqsubseteq_\Gamma \sigma \text{ et } v \sqsubseteq_A M\} \end{aligned}$$

Lemme 10. *Si $E \sqsubseteq_\Delta \delta$ alors $[\![\Delta \supseteq \Gamma]\!]_{\mathsf{Ens}}(E) \sqsubseteq_\Gamma \delta|_\Gamma$, où $\delta|_\Gamma$ désigne la restriction de δ aux liaisons dont la variable apparaît dans Γ.*

Définition 37. On écrira que M *réalise* A *sous* Γ, et on notera $\Gamma \models M : A$, lorsque pour tous $E \in [\![\Gamma]\!]_{\mathsf{Ens}}$ et $\sigma \in [\![\Gamma]\!]_{\mathsf{Syn}}$ tels que $E \sqsubseteq_\Gamma \sigma$ on a $[\![M]\!]_{\mathsf{Ens}}(E) \sqsubseteq_A [\![M]\!]_{\mathsf{Syn}}[\sigma]$.

Le lemme qui suit est le résultat principal de cette section, duquel découlent plusieurs corollaires très utiles. C'est un énoncé typique lorsqu'on travaille avec une relation logique : il exprime que les termes bien typés respectent la relation.

Lemme 11 (Adéquation)**.** *Si $\Gamma \vdash M : A$ alors $\Gamma \models M : A$.*

Démonstration. Par induction sur la dérivation de typage.

- Cas $\dfrac{\Gamma \supseteq x : A}{\Gamma \vdash x : A}$: par le lemme 10 on a $\pi_2([\![\Gamma \supseteq x : A]\!]_{\mathsf{Ens}}(E)) \sqsubseteq_A x[\delta|_{x:A}]$. On conclut immédiatement puisque $x[\delta|_{x:A}] = x[\delta]$.

- Cas $\dfrac{\Gamma \vdash M_1 : A \to B \qquad \Gamma \vdash M_2 : A}{\Gamma \vdash \mathtt{app}(M_1, M_2) : B}$: on a $E \in [\![\Gamma]\!]_{\mathsf{Ens}}$ et $\gamma \in [\![\Gamma]\!]_{\mathsf{Syn}}$ tels que $E \sqsubseteq_\Gamma \gamma$. On doit montrer

$$\begin{aligned} & [\![\mathtt{app}(M_1, M_2)]\!]_{\mathsf{Ens}}(E) \sqsubseteq_B [\![\mathtt{app}(M_1, M_2)]\!]_{\mathsf{Syn}}[\gamma] \\ \Leftrightarrow\ & [\![M_1]\!]_{\mathsf{Ens}}(E)([\![M_2]\!]_{\mathsf{Ens}}(E)) \sqsubseteq_B \mathtt{app}([\![M_1]\!]_{\mathsf{Syn}}[\gamma], [\![M_2]\!]_{\mathsf{Syn}}[\gamma]). \end{aligned}$$

  Par l'hypothèse d'induction sur M_2, on sait que $[\![M_2]\!]_{\mathsf{Ens}}(E) \sqsubseteq_A [\![M_2]\!]_{\mathsf{Syn}}[\gamma]$. On conclut immédiatement par l'hypothèse d'induction sur M_1 et la définition de $\sqsubseteq_{A\to B}$.

- Cas $\dfrac{\Gamma, x : A \vdash M_1 : B}{\Gamma \vdash \mathtt{fun}(x.M_1) : A \to B}$: on a $E \in [\![\Gamma]\!]_{\mathsf{Ens}}$ et $\gamma \in [\![\Gamma]\!]_{\mathsf{Syn}}$ tels que $E \sqsubseteq_\Gamma \gamma$. Soit $v \in [\![A]\!]_{\mathsf{Ens}}$ et $N \in [\![A]\!]_{\mathsf{Syn}}$ tels que $v \sqsubseteq_A N$. On doit montrer

$$\begin{aligned} & [\![\mathtt{fun}(x.M_1)]\!]_{\mathsf{Ens}}(E)(v) \sqsubseteq_B \mathtt{app}([\![\mathtt{fun}(x.M_1)]\!]_{\mathsf{Syn}}[\gamma], N) \\ \Leftrightarrow\ & [\![M_1]\!]_{\mathsf{Ens}}(E, v) \sqsubseteq_B \mathtt{app}(\mathtt{fun}(x.[\![M_1]\!]_{\mathsf{Syn}}[\gamma, x\backslash x]), N) \\ \Leftrightarrow\ & [\![M_1]\!]_{\mathsf{Ens}}(E, v) \sqsubseteq_B [\![M_1]\!]_{\mathsf{Syn}}[\gamma, x\backslash N]. \end{aligned}$$

  Comme $(E, v) \sqsubseteq_{\Gamma, x:A} \gamma, x\backslash N$ on conclue par l'hypothèse d'induction sur M_1.

Les cas du zéro, du successeur et de la récursion primitive sont gérés similairement en utilisant la propriété 19. □

Corollaire 6. *Si* $[\![M : \mathtt{Nat}]\!]_{\mathsf{Ens}} = [\![N : \mathtt{Nat}]\!]_{\mathsf{Ens}}$ *alors* $M \equiv_\beta N$.

Démonstration. On a $M \equiv_\beta [\![M]\!]_{\mathsf{Ens}} = [\![N]\!]_{\mathsf{Ens}} \equiv_\beta N$ par le lemme 11. □

Applications du modèle ensembliste

Théorème 4 (Injectivité des constructeurs, termes clos)**.** *Si* $\mathtt{suc}(M) \equiv_\beta \mathtt{suc}(N) : \mathtt{Nat}$ *alors* $M \equiv_\beta N : \mathtt{Nat}$. *De plus,* $\mathtt{suc}(M) \not\equiv_\beta \mathtt{zero} : \mathtt{Nat}$.

Démonstration. Pour la première propriété, observons que $1 + [\![M]\!]_{\mathsf{Ens}} = 1 + [\![N]\!]_{\mathsf{Ens}}$, et donc $[\![M]\!]_{\mathsf{Ens}} = [\![N]\!]_{\mathsf{Ens}}$ par injectivité du successeur. On conclut par le corollaire 6. La deuxième propriété se démontre de façon analogue. □

Corollaire 7. *Si* $\underline{i} \equiv_\beta \underline{j}$ *alors* $i = j$.

Démonstration. Par induction sur i puis par cas sur j, en utilisant le théorème 4. □

Théorème 5 (Canonicité)**.** *Si* $M : \mathtt{Nat}$ *alors il existe un unique entier* k *tel que*

$$M \equiv_\beta \underline{k}.$$

Démonstration. L'existence découle directement du lemme 11 et de la définition de la relation logique, en prenant $k = [\![M : \mathtt{Nat}]\!]_{\mathsf{Ens}}$. Pour l'unicité, supposons qu'on ait i tel que $M \equiv_\beta \underline{i}$. Par transitivité $\underline{i} \equiv_\beta \underline{k} : \mathtt{Nat}$, et on conclue $i = k$ par le corollaire 7. □

Remarques au sujet du modèle ensembliste

On termine cette section par deux remarques au sujet du modèle ensembliste, l'une positive et l'autre critique.

Extensionalité. Le modèle ensembliste fournit un outil utile à l'étude de l'équivalence β. En particulier, on a vu que lorsque deux termes clos de type $\texttt{Nat}$ ont la même interprétation, alors ils sont équivalents modulo β. Cependant, ce résultat ne s'étend pas au delà du type $\texttt{Nat}$.

Considérons par exemple les termes clos de type $(\texttt{Nat} \to \texttt{Nat}) \to \texttt{Nat} \to \texttt{Nat}$ que sont

$$M_1 = \texttt{fun}(f.f) \qquad \text{et} \qquad M_2 = \texttt{fun}(f.\texttt{fun}(x.\texttt{app}(f, x))).$$

En calculant leurs interprétations respectives dans le modèle ensembliste, on trouve

$$[\![M_1]\!]_{\mathsf{Ens}}() = f \mapsto f \qquad \text{et} \qquad [\![M_2]\!]_{\mathsf{Ens}}() = f \mapsto x \mapsto f(x)$$

qui sont identiques, puisque deux fonctions mathématiques sont égales si et seulement si elles envoient les mêmes entrées dans les mêmes sorties. Pour autant, M_1 et M_2 ne sont pas équivalents modulo β, ce qu'on peut montrer en utilisant les méthodes qu'on verra à la section 10.3.

Plutôt que de considérer ce décalage entre $\mathcal{T}$ et le modèle ensembliste comme un défaut de ce dernier, on peut y voir le signe d'un manque dans la théorie équationnelle choisie. Une façon d'y remédier serait d'ajouter une règle d'équivalence qui stipule que si M est un terme de type $A \to B$, alors M devrait être considéré comme équivalent à $\texttt{fun}(x.\texttt{app}(M, x))$. Ce principe appartient à une famille d'équations appelées *équivalences η*, ou encore *équivalence extensionnelle*.

Définissabilité. On attend d'un modèle qu'il nous éclaire sur la structure du langage interprété en explicitant certains phénomènes difficilement visibles au niveau de la syntaxe. C'est pourquoi le modèle syntaxique est peu intéressant. En échange, celui-ci est très économe au sens où l'interprétation d'un type ne contient pas d'éléments extérieurs à la syntaxe.

Définition 38. Soit $[\![-]\!]$ la fonction d'interprétation d'un modèle quelconque de $\mathcal{T}$ et A un type. Un élément v de $[\![A]\!]$ est *définissable* s'il est dans l'image de la fonction d'interprétation, autrement dit s'il existe un terme $\vdash M : A$ tel que $v = [\![\cdot \vdash M : A]\!]$.

Il est évident que tous types du modèle syntaxique ne contiennent que des éléments définissables. Cette propriété n'est pas vraie du modèle ensembliste.

Propriété 20. *La fonction $F \in (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ telle que*

$$F(f) = \begin{cases} 1 & \textit{si } f \textit{ est la fonction constante } 0 \\ 0 & \textit{sinon} \end{cases}$$

n'est pas définissable.

$$\boxed{\mathsf{Nf}_{\mathcal{T}}} \qquad \frac{v \in \mathsf{Nf}_{\mathcal{T}}}{\mathtt{fun}(x.v) \in \mathsf{Nf}_{\mathcal{T}}} \qquad \frac{}{\mathtt{zero} \in \mathsf{Nf}_{\mathcal{T}}} \qquad \frac{v \in \mathsf{Nf}_{\mathcal{T}}}{\mathtt{suc}(v) \in \mathsf{Nf}_{\mathcal{T}}} \qquad \frac{e \in \mathsf{Ne}_{\mathcal{T}}}{e \in \mathsf{Nf}_{\mathcal{T}}}$$

$$\boxed{\mathsf{Ne}_{\mathcal{T}}} \qquad \frac{}{\mathtt{var}(x) \in \mathsf{Ne}_{\mathcal{T}}} \qquad \frac{e \in \mathsf{Ne}_{\mathcal{T}} \qquad v \in \mathsf{Nf}_{\mathcal{T}}}{\mathtt{app}(e, v) \in \mathsf{Ne}_{\mathcal{T}}} \qquad \frac{e \in \mathsf{Ne}_{\mathcal{T}} \qquad v \in \mathsf{Nf}_{\mathcal{T}} \qquad w \in \mathsf{Nf}_{\mathcal{T}}}{\mathtt{fold}_{\mathtt{Nat}}(e, v, (x, y).w) \in \mathsf{Ne}_{\mathcal{T}}}$$

FIG. 10.11. : Termes normaux et neutres de $\mathcal{T}$.

Très informellement, F n'est pas définissable dans $\mathcal{T}$, ni dans aucun autre langage de programmation, puisqu'une hypothétique implémentation devrait tester son entrée sur un nombre infini d'arguments avant de répondre oui ou non. On introduira au chapitre suivant les outils techniques qui permettent de transformer cette intuition en raisonnement rigoureux.

Remarque 14. L'échec de la définissabilité n'est pas restreint aux fonctions d'ordre supérieur. Le lecteur ou la lectrice qui connaît un peu de théorie de la calculabilité sait que la fonction $g : \mathbb{N} \to \mathbb{N}$ telle que

$$g(n) = \begin{cases} 1 & \text{si } n \text{ est le code d'une machine de Turing qui termine sur toute entrée} \\ 0 & \text{sinon} \end{cases}$$

ne peut pas non plus être définissable.

10.3. Réduction β et normalisation

10.3.1. Énoncé du théorème de normalisation

Le théorème 5 exprime que tout terme clos est équivalent à un résultat entier. Notre objectif est maintenant de généraliser ce théorème aux termes ouverts de types quelconques. Pour cela, on introduit les termes *normaux*, qui généralisent le rôle que jouaient les termes de la forme $\underline{k}$ dans l'énoncé du théorème de canonicité.

Définition 39 (Termes normaux et neutres)**.** L'ensemble des termes *normaux* et l'ensemble des termes *neutres*, respectivement dénotés $\mathsf{Nf}_{\mathcal{T}}$ et $\mathsf{Ne}_{\mathcal{T}}$, sont définis inductivement à la figure 10.11. On désigne par $\mathsf{Nf}_{\mathcal{T}}(\Gamma; A)$ (respectivement $\mathsf{Ne}_{\mathcal{T}}(\Gamma; A)$) l'ensemble des termes normaux (respectivement neutres) de type A sous Γ.

Informellement, un terme normal est une succession de formes d'introduction éventuellement suivie d'un terme neutre. Un terme neutre est une succession de formes d'élimination ultimement “bloquée” par la présence d'une variable. Comme attendu, un terme normal clos de type $\mathtt{Nat}$ est nécessairement de la forme $\underline{k}$ pour $k \in \mathbb{N}$.

Propriété 21. *Si M et N sont deux termes normaux β-équivalents, alors $M = N$.*

Démonstration. Par induction sur la dérivation de l'équivalence. Il est clair qu'aucune des équations (10.2) à (10.4) ne s'applique, la dérivation doit donc être terminée par des applications de l'axiome de réflexivité. □

Théorème 6 (Normalisation)**.** *Pour tout terme $\Gamma \vdash M : A$ il existe un unique terme normal $\Gamma \vdash M' : A$ tel que $M \equiv_\beta M'$.*

On dira que cet unique M' est la *forme normale* du terme M.

10.3.2. Réduction β

Pour démontrer le théorème 6, on va utiliser une méthode de preuve classique dont le caractère implémentatoire est plus apparent que celui des preuves utilisant des modèles. L'idée est de définir une relation qui décrit la réécriture progressive d'un terme vers sa forme normale, où le calcul ne peut plus progresser. Cette relation, appelée *réduction β*, est obtenue en orientant les équations (10.2) à (10.4).

Définition 40. La *réduction β atomique*, notée $\rhd_\beta$, est la relation binaire sur les termes définie par les trois règles suivantes.

$$\mathtt{app}(\mathtt{fun}(x.M), N) \rhd_\beta M[x\backslash N] \tag{10.5}$$

$$\mathtt{fold}_{\mathtt{Nat}}(\mathtt{zero}, N, (x,y).P) \rhd_\beta N \tag{10.6}$$

$$\mathtt{fold}_{\mathtt{Nat}}(\mathtt{suc}(M), N, (x,y).P) \rhd_\beta P[x\backslash M, y\backslash \mathtt{fold}_{\mathtt{Nat}}(M, N, (x,y).P)] \tag{10.7}$$

La *réduction β*, notée $\to_\beta$, est définie inductivement par les règles de la figure 10.12. Cette relation s'étend inductivement aux substitutions via les règles données ci-dessous.

$$\boxed{\sigma \to_\beta \sigma'} \qquad \frac{M \to_\beta M'}{\sigma, x\backslash M \to_\beta \sigma, x\backslash M'} \qquad \frac{\sigma \to_\beta \sigma'}{\sigma, x\backslash M \to_\beta \sigma', x\backslash M}$$

Propriété 22. *Si $M \to_\beta M'$ alors $M[\sigma] \to_\beta M'[\sigma]$.*

Démonstration. Par induction sur la dérivation de $M \to_\beta M'$. □

Propriété 23. *Si $\sigma \to_\beta \sigma'$ alors $M[\sigma] \to_\beta^* M[\sigma']$.*

Démonstration. Induction de routine sur M. □

Lemme 12 (Réduction du sujet)**.** *Si $\Gamma \vdash M : A$ et $M \to_\beta M'$ alors $\Gamma \vdash M' : A$.*

Démonstration. Par induction sur la dérivation de $M \to_\beta M'$. On démontre les cas correspondant à la réduction β atomique en utilisant le lemme 7, les autres suivent par induction. □

Remarque 15. Le lemme 12 exprime que le typage est préservé par la réduction. En revanche, il n'est pas vrai que si $M \to_\beta M'$ et $\Gamma \vdash M' : A$ alors $\Gamma \vdash M : A$.

Lemme 13 (Sûreté)**.** *Pour tout terme $\Gamma \vdash M : A$, M est normal si et seulement s'il n'existe pas de terme M' tel que $M \to_\beta M'$.*

$$\boxed{M \to_\beta M'} \qquad \frac{M \rhd_\beta M'}{M \to_\beta M'} \qquad \frac{M \to_\beta M'}{\mathtt{fun}(x.M) \to_\beta \mathtt{fun}(x.M')}$$

$$\frac{M \to_\beta M'}{\mathtt{app}(M,N) \to_\beta \mathtt{app}(M',N)} \qquad \frac{N \to_\beta N'}{\mathtt{app}(M,N) \to_\beta \mathtt{app}(M,N')} \qquad \frac{M \to_\beta M'}{\mathtt{suc}(M) \to_\beta \mathtt{suc}(M')}$$

$$\frac{M \to_\beta M'}{\mathtt{fold}_{\mathtt{Nat}}(M,N,(x,y).P) \to_\beta \mathtt{fold}_{\mathtt{Nat}}(M',N,(x,y).P)}$$

$$\frac{N \to_\beta N'}{\mathtt{fold}_{\mathtt{Nat}}(M,N,(x,y).P) \to_\beta \mathtt{fold}_{\mathtt{Nat}}(M,N',(x,y).P)}$$

$$\frac{P \to_\beta P'}{\mathtt{fold}_{\mathtt{Nat}}(M,N,(x,y).P) \to_\beta \mathtt{fold}_{\mathtt{Nat}}(M,N,(x,y).P')}$$

FIG. 10.12. : Réduction β pour $\mathcal{T}$

Démonstration. La direction de gauche à droite se montre comme pour la propriété 21. Pour la direction de droite à gauche, on montre par étude de cas que tous les cas où M n'est pas normal sont mal typés. □

Pour calculer la forme normale d'un terme, notre approche va être d'appliquer des étapes de réduction β quelconques jusqu'à l'obtention de la forme normale. Il va falloir montrer que ce procédé termine (existence d'une forme normale), et que la forme normale trouvée ne dépend pas des choix de réductions à effectuer. On va préciser cette idée en introduisant un peu de théorie de la réécriture abstraite.

Un peu de théorie de la réécriture

Généralités au sujet des relations. On rappelle qu'une relation R de A dans B est un sous-ensemble de $A \times B$, ce qu'on note $R : A \nrightarrow B$. On écrira parfois aRb pour signifier que la paire (a, b) appartient à R.

Définition 41 (Identité)**.** La *relation identité* sur un ensemble A est la relation

$$Id_A : A \nrightarrow A \equiv \{(a, a) \mid a \in A\}.$$

Définition 42 (Composée)**.** Étant données deux relations $R : A \nrightarrow B$ et $S : B \nrightarrow C$, la *composée de R et S* est la relation

$$R; S : A \nrightarrow C = \{(a, c) \mid \exists b \in B, (a, b) \in R \text{ et } (b, c) \in S\}.$$

Étant donnée une relation $R : A \nrightarrow A$, on définit $R^n : A \nrightarrow A$ par récurrence sur n en posant $R^0 = Id_A$ et $R^{n+1} = R^n; R$.

Définition 43 (Réciproque)**.** La *réciproque de* $R : A \nrightarrow B$ est la relation

$$R^{op} : B \nrightarrow A \equiv \{(b, a) \mid (a, b) \in R\}.$$

Définition 44 (Clôtures)**.** Soit $R : A \nrightarrow A$.

- La *clôture réflexive* de R est $R^{=} \equiv R \cup Id_A$.
- La *clôture symétrique* de R est $R^{s} \equiv R \cup R^{op}$.
- La *clôture transitive* de R est $R^{t} \equiv \bigcup_{n \geq 1} R^n$.
- La *clôture réflexive-transitive* de R est $R^{*} \equiv \bigcup_{n \geq 0} R^n = R^{=} \cup R^{t}$.

Les clôtures de R définies ci-dessus sont donc les plus petites relations incluant R et respectivement réflexive, symétrique, transitive et réflexive-transitive.

Propriété 24. *La clôture transitive d'une relation symétrique est symétrique.*

La clôture symétrique d'une relation transitive n'est pas nécessairement transitive. Par exemple, la relation $R = \{(a_1, a_2), (a_1, a_3)\}$ est transitive mais sa clôture symétrique $R^s = \{(a_1, a_2), (a_1, a_3), (a_2, a_1), (a_3, a_1)\}$ ne l'est pas puisqu'elle ne contient ni (a_3, a_2) ni (a_2, a_3). La propriété qui suit généralise ce résultat.

Propriété 25. $R^{*s} \subseteq R^{s*}$.

Définition 45. Soit $R : A \nrightarrow A$. On définit $R^{\leq n}$ comme $(R^{=})^n$.

Propriété 26. $R^{*} = \bigcup_{n \geq 0} R^{\leq n}$.

Démonstration. L'inclusion de R^* dans $\bigcup_{n \geq 0} R^{\leq n}$ découle du fait que R^n soit incluse dans $R^{\leq n}$ pour tout n. Pour l'inclusion inverse, on démontre facilement que $R^{\leq n}$ est incluse dans R^* par induction sur n. □

Systèmes de réécriture. L'étude des systèmes de réécriture peut être motivée par une situation générale.

Il arrive fréquemment en mathématiques et en informatique qu'on définisse un ensemble X comme le quotient A/E d'un ensemble A par une relation d'équivalence E. Par exemple, les éléments du modèle syntaxique de $\mathcal{T}$ sont définis comme le quotient des termes du langage par l'équivalence β. Pour manipuler algorithmiquement l'ensemble X, il faut être capable de décider la relation d'équivalence E, c'est-à-dire disposer d'une procédure effective qui permette de répondre exactement à la question “le couple (a_1, a_2) appartient-il à la relation E ?” pour tout couple (a_1, a_2) de A.

L'approche par réécriture consiste à répondre à cette question via la définition d'une relation R de l'ensemble A dans lui-même telle que $R^{s*} = E$. Lorsque la relation d'équivalence E a été définie inductivement, la relation R peut souvent être obtenue en orientant les axiomes de E dans une direction bien choisie. On voit R comme une relation qui décrit comment “réécrire” les éléments de A, assimilés à des objets syntaxiques.

Si R vérifie certaines bonnes propriétés, on dispose alors d'une procédure de décision pour $a_1 E a_2$ qui consiste à appliquer la relation R librement et autant que possible à a_1 et a_2 jusqu'à l'obtention d'éléments a'_1 et a'_2 qu'il n'est plus possible de réécrire. Ceux-ci sont équivalents pour E si et seulement si $a_1 R^{s*} a_2$, et donc si et seulement si $a_1 E a_2$.

La théorie de la réécriture abstraite fournit un cadre général pour étudier de telles situations. De façon peut-être étonnante, à ce niveau la définition d'un système de réécriture ne nécessite aucune hypothèse sur l'ensemble sous-jacent.

Définition 46. Un *système de réécriture* est une paire (A, R) où A est un ensemble et R une relation de A dans A.

Par abus de langage, on tend à appeler système de réécriture la relation R, l'ensemble A pouvant être déduit du contexte.

Pour pouvoir appliquer la procédure de décision décrite plus haut, on va vouloir que R vérifie deux propriétés essentielles : la *normalisation forte* et la *propriété de Church-Rosser*. Intuitivement, la première assure la terminaison des calculs et donc l'existence de résultats, la seconde l'unicité des résultats.

Définition 47 (Éléments normaux, normalisation)**.** Un élément a de A est *normal* s'il n'existe pas de a' tel que aRa'. Un élément a_0 de A est *normalisable* s'il existe un terme normal a tel que $a_0 R^* a$. On dit alors que a est une *forme normale de* a_0. Un système de réécriture abstrait est *normalisant* si tous les éléments de A sont normalisables.

Remarque 16. Cette définition est cohérente avec la terminologie introduite en début de section pour le cas particulier considéré : le lemme 13 exprime que les formes normales du système de réécriture $(\mathsf{Term}_{\mathcal{T}}, \to_\beta)$ sont bien les termes normaux au sens de la figure 10.11.

Définition 48 (Normalisation forte)**.** Un élément a_0 de A est *fortement normalisable* s'il existe un entier naturel $B(a_0)$ tel que pour toute séquence de réécriture $a_0 R a_1 R \dots R a_n$ on a $n \leq B(a_0)$. Un système de réécriture est *fortement normalisant* si tous les éléments de A sont fortement normalisables.

Il est clair que tout système de réécriture fortement normalisant est normalisant. La réciproque n'est pas vraie puisqu'un système de réécriture normalisant peut présenter des séquences infinies de réduction. L'absence de telle séquence garantie la terminaison de la réécriture et ce quels que soient les choix effectués à chaque étape.

Propriété 27. *Un élément a de A est fortement normalisable si et seulement si tous les termes a' tels que aRa' sont fortement normalisables.*

Passons maintenant à la propriété de Church-Rosser. Si l'on pense à la relation d'équivalence E comme à la clôture symétrique-réflexive-transitive de R, cette propriété assure que deux éléments équivalents partagent une forme normale.

Définition 49 (Church-Rosser)**.** Un système de réécriture R a la *propriété de Church-Rosser* si $a_1 R^{s*} a_2$ implique qu'il existe a tel que $a_1 R^* a$ et $a_2 R^* a$.

La propriété de Church-Rosser est peu agréable à manipuler, et on lui préfère donc une propriété équivalente et plus intuitive, la *confluence*.

Définition 50. Soient P et Q des relations d'un ensemble A dans lui-même. On dit que P est *Q-confluente* si pour tout triplet (a, a_1, a_2) d'éléments de A tel que aPa_1 et aPa_2, alors il existe un a' dans A tel que a_1Qa' et a_2Qa'.

Définition 51. Soit (A, R) un système de réécriture abstrait. Ce système :

- est *déterministe* si R est Id_A-confluente,
- a la *propriété du diamant* si R est R-confluente,
- est *localement confluent* si R est R^*-confluente,
- est *globalement confluent* ou simplement *confluent* si R^* est R^*-confluente.

Remarque 17. Un système de réécriture abstrait peut être confluent et normalisant sans être fortement normalisant.

Propriété 28. *Les trois propriétés suivantes sont équivalentes :*

1. *R est confluent,*
2. *R^* est localement confluent,*
3. *R^* a la propriété du diamant.*

Propriété 29. *Un système de réécriture a la propriété de Church-Rosser si et seulement s'il est confluent.*

Démonstration. Supposons que R ait la propriété de Church-Rosser. Soient a, a_1, a_2 tels que aR^*a_1 et aR^*a_2. On a donc $a_1R^{*s}a$ et $aR^{*s}a_2$. Par la propriété 25 on a $a_1R^{s*}a$ et $aR^{s*}a_2$, donc $a_1R^{s*}a_2$ et donc, par Church-Rosser, il existe a' tel que a_1R^*a' et a_2R^*a'. Donc R est confluente.

Supposons que R soit confluente. Pour démontrer que R a la propriété de Church-Rosser, il suffit de démontrer que pour tout entier n, si $a_1(R^s)^n a_2$ alors il existe a tel que a_1R^*a et a_2R^*a. On procède par induction sur n.

- Cas $n = 0$: immédiat puisque $a_1 = a_2$ et on choisit donc $a = a_1 = a_2$.
- Cas $n = n' + 1$: on a a'_2 tel que $a_1(R^s)^{n'}a'_2$ et $a'_2R^sa_2$. L'hypothèse d'induction nous donne un certain a' tel que a_1R^*a' et a'_2R^*a'. On raisonne par cas sur $a'_2R^sa_2$.
  - Cas $a_2Ra'_2$: on a alors $a_2Ra'_2R^*a'$, donc a_2Ra' et on choisit $a = a'$.
  - Cas a'_2Ra_2 : on a a'_2R^*a' et a'_2Ra_2 donc, puisque R est confluent, il existe a tel que $a'R^*a$ et a_2R^*a. C'est bien le a désiré puisque $a_1R^*a'R^*a$.

□

Définition 52 (Convergence)**.** Un système de réécriture est dit *convergent* s'il est confluent et fortement normalisant.

Le théorème suivant exprime l'essence de l'approche.

Théorème 7. *Si R est un système de réécriture convergent, tout élément a de A dispose d'une unique forme normale.*

Démonstration. La normalisation assure que l'ensemble des formes normales de a n'est pas vide. La confluence assure que si a_1 et a_2 sont des formes normales de a, il existe a' tel que $a_1 R^* a'$ et $a_2 R^* a'$. Mais puisque a_1 et a_2 sont normaux, on a $a_1 = a' = a_2$. □

Dans les faits, démontrer la confluence d'un système de réécriture peut être difficile. Pour ce faire, on utilisera fréquemment le lemme 14 ci-dessous.

Propriété 30. *Si R a la propriété du diamant alors R^* l'a également.*

Démonstration. Supposons que R ait la propriété du diamant. On prouve la propriété suivante : pour tous m, n, si $aR^n a_1$ et $aR^m a_2$ alors il existe a' tel que $a_1 R^m a'$ et $a_2 R^n a'$. Il est clair que le cas particulier $m = n$ entraîne que R^* a la propriété du diamant.

On procède par induction forte sur le maximum de m et n, puis par cas.

- Si $n = 0$, on a $a = a_1$ et on choisit $a' = a_2$. On a bien $a_2 R^0 a_2$ et $a_1 = aR^m a_2$.
- Le cas $m = 0$ est symétrique.
- Si $n = n' + 1$ et $m = m' + 1$, on a a'_1 et a'_2 tels que $aR^{n'} a'_1 R a_1$ et $aR^{m'} a'_2 R a_2$. En appliquant une première fois l'hypothèse d'induction, on obtient a'' tel que $a'_1 R^{m'} a''$ et $a'_2 R^{n'} a''$. En appliquant une deuxième l'hypothèse d'induction, on obtient a''_1 tel que $a_1 R^{m'} a''_1$ et $a'' R^1 a''_1$. En appliquant une troisième fois l'hypothèse d'induction, on obtient a''_2 tel que $a_2 R^{n'} a''_2$ et $a'' R^1 a''_2$. On utilise le fait que R a la propriété du diamant pour obtenir a' tel que $a''_1 R a'$ et $a''_2 R a'$. On a bien $a_1 R^{m'} a''_1 R a'$ et $a_2 R^{n'} a''_2 R a'$ comme attendu.

□

Lemme 14. *Si R et T sont des systèmes de réécritures tels que $R \subseteq T \subseteq R^*$ et que T a la propriété du diamant, alors R est confluent.*

Démonstration. Par la propriété 30, T^* a la propriété du diamant. Mais on a $R^* \subseteq T^* \subseteq R^{**} = R^*$, donc $T^* = R^*$ et R est confluent. □

Confluence de la réduction β

On va maintenant appliquer le lemme 14 pour démontrer la confluence de la réduction β. Pour cela, on va définir la *réduction β parallèle* comme un système de réécriture intermédiaire entre la réduction β et sa clôture transitive.

Définition 53 (Réduction β parallèle)**.** La relation de *réduction β parallèle* de $\mathcal{T}$, notée $\rightrightarrows_\beta$, est la relation définie inductivement à la figure 10.13.

$$\boxed{M \rightrightarrows_\beta M'} \qquad \frac{}{\mathtt{var}(x) \rightrightarrows_\beta \mathtt{var}(x)} \qquad \frac{M \rightrightarrows_\beta M'}{\mathtt{fun}(x.M) \rightrightarrows_\beta \mathtt{fun}(x.M')}$$

$$\frac{M \rightrightarrows_\beta M' \qquad N \rightrightarrows_\beta N'}{\mathtt{app}(M,N) \rightrightarrows_\beta \mathtt{app}(M',N')} \qquad \frac{M \rightrightarrows_\beta M' \qquad N \rightrightarrows_\beta N'}{\mathtt{app}(\mathtt{fun}(x.M),N) \rightrightarrows_\beta M'[x\backslash N']} \qquad \frac{}{\mathtt{zero} \rightrightarrows_\beta \mathtt{zero}}$$

$$\frac{M \rightrightarrows_\beta M'}{\mathtt{suc}(M) \rightrightarrows_\beta \mathtt{suc}(M')} \qquad \frac{M \rightrightarrows_\beta M' \qquad N \rightrightarrows_\beta N' \qquad P \rightrightarrows_\beta P'}{\mathtt{fold}_{\mathtt{Nat}}(M,N,(x,y).P) \rightrightarrows_\beta \mathtt{fold}_{\mathtt{Nat}}(M',N',(x,y).P')}$$

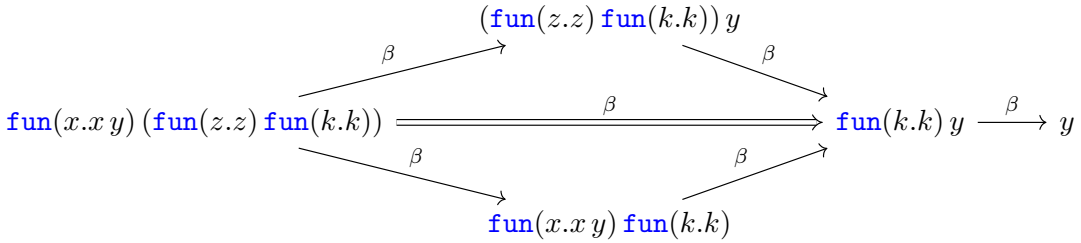
$$\frac{N \rightrightarrows_\beta N'}{\mathtt{fold}_{\mathtt{Nat}}(\mathtt{zero},N,(x,y).P) \rightrightarrows_\beta N'}$$

$$\frac{M \rightrightarrows_\beta M' \qquad N \rightrightarrows_\beta N' \qquad P \rightrightarrows_\beta P'}{\mathtt{fold}_{\mathtt{Nat}}(\mathtt{suc}(M),N,(x,y).P) \rightrightarrows_\beta P'[x\backslash M', y\backslash \mathtt{fold}_{\mathtt{Nat}}(M',N',(x,y).P')]}$$

$$\boxed{\sigma \rightrightarrows_\beta \sigma'} \qquad \frac{}{\mathtt{id} \rightrightarrows_\beta \mathtt{id}} \qquad \frac{\sigma \rightrightarrows_\beta \sigma' \qquad M \rightrightarrows_\beta M'}{\sigma, x\backslash M \rightrightarrows_\beta \sigma', x\backslash M'}$$

FIG. 10.13. : Réduction parallèle de $\mathcal{T}$

Intuitivement, une étape de réduction β parallèle $M \rightrightarrows_\beta M'$ applique simultanément un certain nombre des réductions β atomiques partant du terme M. En revanche, elle ne permet pas d'appliquer des réductions β qui n'auraient pas déjà été présentes dans le terme de départ. Le graphe de réduction



illustre ce phénomène : depuis $\mathtt{fun}(x.x\,y)\,(\mathtt{fun}(z.z)\,\mathtt{fun}(k.k))$ on peut atteindre $\mathtt{fun}(k.k)\,y$ en une seule réduction parallèle, mais pas y.

On démontre maintenant les hypothèses du lemme 14.

Propriété 31. *$\rightrightarrows_\beta$ est une $\mathcal{T}$-congruence.*

Propriété 32. *Si $M \to_\beta N$ alors $M \rightrightarrows_\beta N$.*

Démonstration. Puisque la réduction parallèle est une $\mathcal{T}$-congruence, il suffit de traiter les équations (10.2) à (10.4). C'est immédiat puisque la réduction parallèle est réflexive. □

Propriété 33. *Si $M \rightrightarrows_\beta N$ alors $M \to_\beta^* N$.*

Démonstration. Par induction sur $M \rightrightarrows_\beta N$ en utilisant le fait que $\to_\beta^*$ est une $\mathcal{T}$-congruence. □

Propriété 34. *Si $M \rightrightarrows_\beta M'$ et $\sigma \rightrightarrows_\beta \sigma'$ alors $M[\sigma] \rightrightarrows_\beta M'[\sigma']$.*

Démonstration. Par une induction de routine sur M. □

On veut maintenant démontrer que $\rightrightarrows_\beta$ a la propriété du diamant. Pour ce faire, on emploie la méthode de Takahashi [8].

Définition 54. Le *contracté parallèle maximal* d'un terme M, noté $c(M)$, est un terme défini par récurrence sur M comme suit.

$$
\begin{aligned}
c(x) &= x \\
c(\mathtt{fun}(x.M_1)) &= \mathtt{fun}(x.c(M_1)) \\
c(\mathtt{app}(M_1, M_2)) &= \begin{cases} M_1'[x \backslash c(M_2)] \\ \quad \text{si } c(M_1) \text{ est de la forme } \mathtt{fun}(x.M_1'), \\ \mathtt{app}(c(M_1), c(M_2)) \\ \quad \text{sinon.} \end{cases} \\
c(\mathtt{zero}) &= \mathtt{zero} \\
c(\mathtt{suc}(M_1)) &= \mathtt{suc}(c(M_1)) \\
c(\mathtt{fold}_{\mathtt{Nat}}(M_1, M_2, (x,y).M_3)) &= \begin{cases} c(M_2) \\ \quad \text{si } c(M_1) \text{ est de la forme } \mathtt{zero}, \\ c(M_3)[x \backslash M_1', y \backslash \mathtt{fold}_{\mathtt{Nat}}(M_1', c(M_2), (x,y).c(M_3))] \\ \quad \text{si } c(M_1) \text{ est de la forme } \mathtt{suc}(M_1'), \\ \mathtt{fold}_{\mathtt{Nat}}(c(M_1), c(M_2), (x,y).c(M_3)) \\ \quad \text{sinon.} \end{cases}
\end{aligned}
$$

Propriété 35. *Si $M \rightrightarrows_\beta N$ alors $N \rightrightarrows_\beta c(M)$.*

Démonstration. Par induction sur $M \rightrightarrows_\beta N$. Comme précédemment, les cas correspondant aux règles de congruence sont immédiats. Les cas correspondant aux règles qui implémentent la réduction β atomique utilisent la propriété 34. □

Corollaire 8. *La β réduction parallèle a la propriété du diamant.*

Démonstration. Si $M \rightrightarrows_\beta N_1$ et $M \rightrightarrows_\beta N_2$ alors $N_1 \rightrightarrows_\beta c(M)$ et $N_2 \rightrightarrows_\beta c(M)$. □

Théorème 8. *$\to_\beta$ est confluent.*

Démonstration. On a $\to_\beta \subseteq \rightrightarrows_\beta \subseteq \to_\beta^*$ par les propriétés 32 et 33, et on sait que $\rightrightarrows_\beta$ a la propriété du diamant par le corollaire 8. On conclut par le lemme 14. □

Normalisation forte de la réduction β

On donne ici la preuve de normalisation forte dûe à Krivine [4]. Cette preuve s'étend facilement à des langages beaucoup plus riches, mais demande un certain nombre de définitions et propriétés préliminaires.

Normalisations

Définition 55. Soit M_0 un terme. Une *normalisation* de M_0 est une séquence $M_1, \ldots, M_n$ de termes avec $M_0 \to_\beta M_1 \to_\beta \ldots \to_\beta M_n$ et M_n normal.

Propriété 36. *Un terme fortement normalisable n'a qu'un nombre fini de normalisations.*

Démonstration. Soit M_0 un terme fortement normalisable. On raisonne par l'absurde. Supposons qu'il existe un nombre infini de normalisations de M_0. Considérons l'arbre dont M_0 forme la racine, et dont les branches sont données par les normalisations de M_0. Comme, pour tout terme M, il n'existe qu'un nombre fini de M' tels que $M \to_\beta M'$, on a construit un arbre infini à branchement fini. Par le lemme de König, cet arbre dispose d'une branche infinie, ce qui entre en contradiction avec le fait que M_0 soit fortement normalisable. □

Définition 56. Le *poids* d'un terme fortement normalisable, noté $\mathsf{w}(M)$, est la somme des longeurs de ses normalisations.

Remarque 18. La notion de poids n'est définie que pour les termes fortement normalisables, puisque la propriété 36 garantie la finitude du nombre de normalisations.

Propriété 37. *Si M est fortement normalisable et $M \to_\beta M'$ alors M' est fortement normalisable et $\mathsf{w}(M') < \mathsf{w}(M)$.*

Démonstration. On sait que M' est fortement normalisable par la propriété 27, ce qui donne le droit d'écrire $\mathsf{w}(M')$. Toute normalisation de M' de longueur n donne lieu à une normalisation de M de longueur $n+1$, donc $\mathsf{w}(M') < \mathsf{w}(M)$. □

Ensembles de termes

Définition 57. Soient $\mathcal{X}$, $\mathcal{Y}$ et $\mathcal{Z}$ des ensembles de termes. On définit les ensembles de termes $\mathtt{fun}(x.\mathcal{X})$, $\mathtt{app}(X, \mathcal{Y})$, $\mathtt{suc}(\mathcal{X})$, $\mathtt{fold}_{\mathtt{Nat}}(\mathcal{X}, \mathcal{Y}, (x,y).\mathcal{Z})$ et $\mathcal{X} \to \mathcal{Y}$ comme suit.

$$
\begin{aligned}
\mathtt{fun}(x.\mathcal{X}) &= \{\mathtt{fun}(x.M) \mid M \in \mathcal{X}\} \\
\mathtt{app}(\mathcal{X}, \mathcal{Y}) &= \{\mathtt{app}(M, N) \mid M \in \mathcal{X}, N \in \mathcal{Y}\} \\
\mathtt{suc}(\mathcal{X}) &= \{\mathtt{suc}(M) \mid M \in \mathcal{X}, N \in \mathcal{Y}\} \\
\mathtt{fold}_{\mathtt{Nat}}(\mathcal{X}, \mathcal{Y}, (x,y).\mathcal{Z}) &= \{\mathtt{fold}_{\mathtt{Nat}}(M, N, (x,y).P) \mid M \in \mathcal{X}, N \in \mathcal{Y}, P \in \mathcal{Z}\} \\
\mathcal{X} \to \mathcal{Y} &= \{M \in \mathsf{Term}_{\mathcal{T}} \mid \forall N \in \mathcal{X}, \mathtt{app}(M, N) \in \mathcal{Y}\}
\end{aligned}
$$

Définition 58. Soit $\mathcal{X}$ un ensemble de termes. L'ensemble des termes $\mathcal{X}$-*neutres*, dénoté $\mathsf{Ne}_{\mathcal{T}}(\mathcal{X})$, est défini inductivement par les règles suivantes.

$$\frac{}{\mathtt{var}(x) \in \mathsf{Ne}_{\mathcal{T}}(\mathcal{X})} \qquad \frac{M \in \mathsf{Ne}_{\mathcal{T}}(\mathcal{X}) \qquad N \in \mathcal{X}}{\mathtt{app}(M,N) \in \mathsf{Ne}_{\mathcal{T}}(\mathcal{X})} \qquad \frac{M \in \mathsf{Ne}_{\mathcal{T}}(\mathcal{X}) \qquad N,P \in \mathcal{X}}{\mathtt{fold}_{\mathtt{Nat}}(M,N,(x,y).P) \in \mathsf{Ne}_{\mathcal{T}}(\mathcal{X})}$$

Propriété 38. *Soient $\mathcal{X},\mathcal{Y},\mathcal{Z},\mathcal{X}',\mathcal{Y}',\mathcal{Z}'$ des ensembles de termes. Si $\mathcal{X} \subseteq \mathcal{X}'$, $\mathcal{Y} \subseteq \mathcal{Y}'$ et $\mathcal{Z} \subseteq \mathcal{Z}'$ alors on a les inclusions suivantes.*

$$\begin{gathered}
\mathtt{fun}(x.\mathcal{X}) \subseteq \mathtt{fun}(x.\mathcal{X}') \\
\mathtt{app}(\mathcal{X},\mathcal{Y}) \subseteq \mathtt{fun}(\mathcal{X}'.\mathcal{Y}') \\
\mathtt{suc}(\mathcal{X}) \subseteq \mathtt{suc}(\mathcal{X}') \\
\mathtt{fold}_{\mathtt{Nat}}(\mathcal{X},\mathcal{Y},(x,y).\mathcal{Z}) \subseteq \mathtt{fold}_{\mathtt{Nat}}(\mathcal{X}',\mathcal{Y}',(x,y).\mathcal{Z}') \\
\mathcal{X}' \to \mathcal{Y} \subseteq \mathcal{X} \to \mathcal{Y}' \\
\mathsf{Ne}_{\mathcal{T}}(\mathcal{X}) \subseteq \mathsf{Ne}_{\mathcal{T}}(\mathcal{X}')
\end{gathered}$$

On désigne par $\mathsf{No}_{\mathcal{T}}$ l'ensemble des termes fortement normalisables. Dans cette section, on dira qu'un terme est *neutre* lorsqu'il est $\mathsf{No}_{\mathcal{T}}$-neutre. On note $\mathsf{Ne}_{\mathcal{T}}$ l'ensemble $\mathsf{Ne}_{\mathcal{T}}(\mathsf{No}_{\mathcal{T}})$ des termes neutres en ce sens.

Remarque 19. Les termes neutres au sens de la section précédente correspondent à l'ensemble $\mathsf{Ne}_{\mathcal{T}}(\mathsf{Nf}_{\mathcal{T}})$, qui est un strict sous-ensemble dans $\mathsf{Ne}_{\mathcal{T}}(\mathsf{No}_{\mathcal{T}})$ puisque $\mathsf{Nf}_{\mathcal{T}} \subseteq \mathsf{No}_{\mathcal{T}}$.

Propriété 39. $\mathsf{No}_{\mathcal{T}}; \to_\beta \subseteq \mathsf{No}_{\mathcal{T}}$.

Démonstration. C'est une reformulation concise de la propriété 27. □

Propriété 40. $\mathtt{fun}(x.\mathsf{No}_{\mathcal{T}}) \subseteq \mathsf{No}_{\mathcal{T}}$ *et* $\mathtt{suc}(\mathsf{No}_{\mathcal{T}}) \subseteq \mathsf{No}_{\mathcal{T}}$.

Démonstration. On doit montrer que $M = \mathtt{fun}(x.N)$ (resp. $\mathtt{suc}(M)$) est fortemente normalisable si N fortement normalisable. On procède par induction sur $\mathsf{w}(N)$. Par la propriété 27, il suffit de prouver que tout terme M' tel que $M \to_\beta M'$ est fortement normalisable. La seule possibilité est $M' = \mathtt{fun}(x.N')$ (resp. $\mathtt{suc}(N')$) avec $N \rightrightarrows N'$. Or, $\mathsf{w}(N') < \mathsf{w}(N)$ par la propriété 37, donc on conclue immédiatement par l'hypothèse d'induction. □

Propriété 41. $\mathsf{Ne}_{\mathcal{T}}; \to_\beta \subseteq \mathsf{Ne}_{\mathcal{T}}$.

Démonstration. Soit $M \to_\beta M'$ avec M neutre. On procède par induction sur M. Dans le cas des variables, aucune réduction n'est possible. Dans le cas d'une application $\mathtt{app}(M_1,M_2)$, la réduction se produit forcément dans M_1 ou dans M_2 puisque M_1 est neutre et ne peut donc pas être une forme d'introduction et a fortiori une abstraction. On peut alors conclure par induction ou par $\mathcal{T}$-congruence de la réduction β le cas échéant. Le cas de la récursion primitive est similaire. □

Propriété 42. $\mathtt{app}(\mathsf{Ne}_{\mathcal{T}} \cap \mathsf{No}_{\mathcal{T}}, \mathsf{No}_{\mathcal{T}}) \subseteq \mathsf{No}_{\mathcal{T}}$.

Démonstration. On doit montrer que si M_1 est neutre et fortement normalisable et M_2 fortement normalisable alors $\texttt{app}(M_1, M_2)$ est fortement normalisable. On raisonne par induction sur $\mathsf{w}(M_1) + \mathsf{w}(M_2)$. Par la propriété 27, il suffit de prouver que tout terme P tel que $\texttt{app}(M_1, M_2) \to_\beta P$ est fortement normalisable. Cette réduction est nécessairement la réduction d'un des deux sous-termes M_i en un certain M_i' puisque M_1 est neutre et ne peut donc pas être une abstraction. Par les propriétés 37, 39 et 41, on sait que le poids de M_i' est strictement inférieur à celui de M_i, et que celui-ci est toujours fortement normalisable et éventuellement neutre s'il s'agit de M_1. On peut donc conclure en applicant l'hypothèse d'induction. □

Propriété 43. $\texttt{fold}_{\texttt{Nat}}(\mathsf{Ne}_\mathcal{T} \cap \mathsf{No}_\mathcal{T}, \mathsf{No}_\mathcal{T}, (x, y).\mathsf{No}_\mathcal{T}) \subseteq \mathsf{No}_\mathcal{T}$.

Démonstration. Essentiellement identique à celle de la propriété 42. □

Propriété 44. $\mathsf{Ne}_\mathcal{T} \subseteq \mathsf{No}_\mathcal{T}$.

Démonstration. Soit M un terme neutre. On montre qu'il est fortement normalisable par induction. Le cas des variables est immédiat puisqu'elles sont normales. Dans les deux autres cas, l'application de l'hypothèse d'induction permet de conclure par les propriétés 42 et 43. □

Propriété 45. $\mathsf{Ne}_\mathcal{T} \subseteq \mathsf{No}_\mathcal{T} \to \mathsf{Ne}_\mathcal{T}$.

Propriété 46. $\mathsf{Ne}_\mathcal{T} \to \mathsf{No}_\mathcal{T} \subseteq \mathsf{No}_\mathcal{T}$.

Démonstration. Supposons que M_0 soit un terme tel que pour tout N neutre, $\texttt{app}(M_0, N)$ soit fortemente normalisable. On procède par l'absurde. Supposons que M_0 ne soit pas fortement normalisable, c'est-à-dire qu'il existe une séquence infinie

$$M_0 \to_\beta M_1 \to_\beta M_2 \to_\beta \dots$$

de réductions. Mais alors, pour n'importe quelle variable x, on a la séquence infinie

$$\texttt{app}(M_0, x) \to_\beta \texttt{app}(M_1, x) \to_\beta \texttt{app}(M_2, x) \to_\beta \dots$$

alors que $\texttt{app}(M_0, x)$ est fortement normalisant puisque x est neutre. □

Réductions On s'intéresse maintenant à un cas particulier de la réduction β.

Définition 59 (Réduction de tête faible)**.** La *réduction de tête faible*, notée $\to_{wh}$, est définie inductivement par les règles de la figure 10.14.

Propriété 47. *Si $M \to_{wh} M'$ alors $M \to_\beta M'$.*

Intuitivement, la réduction de tête faible est plus proche d'une implémentation sur machine que la réduction β. On s'interdit notamment de réduire sous les abstractions, et on fixe un unique choix de sous-terme à réduire dans chaque terme.

$$\boxed{M \to_\beta M'} \qquad \frac{M \rhd_\beta M'}{M \to_{wh} M'} \qquad \frac{M \to_{wh} M'}{\mathtt{app}(M,N) \to_{wh} \mathtt{app}(M',N)}$$

$$\frac{M \to_{wh} M'}{\mathtt{fold}_{\mathtt{Nat}}(M,N,(x,y).P) \to_{wh} \mathtt{fold}_{\mathtt{Nat}}(M',N,(x,y).P)}$$

FIG. 10.14. : Réduction de tête faible pour $\mathcal{T}$

Remarque 20. On s'est également interdit de réduire sous les successeurs. Il s'agit d'un choix discutable, qui n'est pas le seul possible.

Propriété 48. *La relation $\to_{wh}$ est déterministe.*

Dans les énoncés ci-dessous, on note $M \twoheadrightarrow_\beta N$ lorsque N est obtenu par réduction β d'un sous-terme strict de M, autrement dit lorsque $M \to_\beta N$ et $M \not\rhd_\beta N$.

Propriété 49. *Si $M \to_\beta M'$ alors soit $M \rhd_\beta M'$ soit $M \twoheadrightarrow_\beta M'$.*

Théorème 9 (Standardisation faible)**.** *Si $M \to_{wh} M'$ et $M \twoheadrightarrow_\beta N$ alors il existe N' tel que $N \to_{wh} N'$ et $M' \to_\beta^* N'$.*

Démonstration. Par induction sur la dérivation de $M \to_{wh} M'$, puis par cas sur le sous-terme réécrit par $\twoheadrightarrow_\beta$ dans M. On ne détaille que deux cas représentatifs.

- Cas $M = \mathtt{app}(\mathtt{fun}(x.M_1), M_2) \to_{wh} M_1[x\backslash M_2] = M'$: le sous-terme réécrit est soit M_1, soit M_2.
  - Cas $N = \mathtt{app}(\mathtt{fun}(x.M_1'), M_2)$ et $M_1 \to_\beta M_1'$: on choisit $N' = M_1'[x\backslash M_2]$. Clairement $N \to_{wh} N'$. On a $M_1[x\backslash M_2] \to_\beta M_1[x\backslash M_2']$ par la propriété 22.
  - Cas $N = \mathtt{app}(\mathtt{fun}(x.M_1), M_2')$ et $M_2 \to_\beta M_2'$: on choisit $N' = M_1[x\backslash M_2']$. Clairement $N \to_{wh} N'$. On a $M_1[x\backslash M_2] \to_\beta^* M_1[x\backslash M_2']$ par la propriété 23.
- Cas $M = \mathtt{app}(M_1, M_2) \to_{wh} \mathtt{app}(M_1', M_2) = M'$ avec $M_1 \to_{wh} M_1'$: le sous-terme réécrit est soit M_1, soit M_2.
  - Cas $N = \mathtt{app}(M_1, M_2')$ avec $M_2 \to_\beta M_2'$: on choisit $N' = \mathtt{app}(M_1, M_2')$. Clairement $N \to_{wh} N'$ et $M' = \mathtt{app}(M_1', M_2) \to_\beta N'$.
  - Cas $N = \mathtt{app}(M_1'', M_2)$ avec $M_1 \to_\beta M_1''$: on applique l'hypothèse d'induction et on obtient M_1''' tel que $M_1' \to_\beta^* M_1'''$ et $M_1'' \to_{wh} M_1'''$. On choisit $N' = \mathtt{app}(M_1''', M_2)$. Clairement $N \to_{wh} N'$ et $M' = \mathtt{app}(M_1'', M_2) \to_\beta N'$.

□

Ensembles saturés et candidats

Définition 60. Un ensemble $\mathcal{X}$ de termes est *saturé* lorsque pour tout M' dans $\mathcal{X}$ et tout M tel que $M \to_{wh} M'$, si tous les sous-termes immédiats de M sont dans $\mathcal{X}$ alors M est dans $\mathcal{X}$.

Définition 61. Un $\mathcal{T}$-*candidat* est un ensemble saturé $\mathcal{X}$ tel que $\mathsf{Ne}_\mathcal{T} \subseteq \mathcal{X} \subseteq \mathsf{No}_\mathcal{T}$.

On va maintenant construire un $\mathcal{T}$-candidat pour interpréter chaque type. Pour ce faire, on va montrer que l'ensemble $\mathsf{No}_\mathcal{T}$ lui-même constitue un $\mathcal{T}$-candidat, et que l'ensemble $\mathcal{X} \to \mathcal{Y}$ est un $\mathcal{T}$-candidat si $\mathcal{X}$ et $\mathcal{Y}$ le sont.

Lemme 15. *L'ensemble* $\mathsf{No}_\mathcal{T}$ *est saturé.*

Démonstration. Soient M et M' deux termes tels que $M \to_{wh} M'$ avec M' et tous les sous-termes immédiats de M fortement normalisables. On montre $M \in \mathsf{No}_\mathcal{T}$ par induction sur la somme des poids de M' et des sous-termes immédiats de M. Par la propriété 27, il suffit de démontrer que tout N tel que $M \to_\beta N$ est fortement normalisable. On utilise la propriété 49 pour raisonner par cas.

- Cas $M \rhd_\beta N$: alors $M \to_{wh} N$ et donc, par la propriété 48, $N = M'$ est fortement normalisable.
- Cas $M \to\!\!\!\!\triangleright_\beta N$: par le théorème 9, on a un certain N' tel que $N \to_{wh} N'$ et $M' \to_\beta^* N'$. On a donc $\mathsf{w}(N) < \mathsf{w}(N')$ et $\mathsf{w}(M') \leq \mathsf{w}(N')$. On conclut par application de l'hypothèse d'induction.

□

Lemme 16. $\mathsf{No}_\mathcal{T}$ *est un* $\mathcal{T}$*-candidat.*

Démonstration. Par le lemme 15 et la propriété 44. □

Propriété 50. *Si* $\mathsf{Ne}_\mathcal{T} \subseteq \mathcal{X} \subseteq \mathsf{No}_\mathcal{T}$ *et* $\mathsf{Ne}_\mathcal{T} \subseteq \mathcal{Y} \subseteq \mathsf{No}_\mathcal{T}$ *alors* $\mathsf{Ne}_\mathcal{T} \subseteq \mathcal{X} \to \mathcal{Y} \subseteq \mathsf{No}_\mathcal{T}$.

Démonstration. On a $\mathsf{Ne}_\mathcal{T} \subseteq \mathsf{No}_\mathcal{T} \to \mathsf{Ne}_\mathcal{T} \subseteq \mathcal{X} \to \mathcal{Y}$ par les propriétés 38 et 45 et $\mathcal{X} \to \mathcal{Y} \subseteq \mathsf{Ne}_\mathcal{T} \to \mathsf{No}_\mathcal{T} \subseteq \mathsf{No}_\mathcal{T}$ par les propriétés 38 et 46. □

Propriété 51. *Si* $\mathcal{Y}$ *est saturé et* $\mathcal{X}$ *inclu dans* $\mathsf{No}_\mathcal{T}$*, alors* $\mathcal{X} \to \mathcal{Y}$ *est saturé.*

Démonstration. Soient M et M' deux termes tels que $M \to_{wh} M'$ avec $M' \in \mathcal{X} \to \mathcal{Y}$ et tous les sous-termes immédiats de M fortement normalisables. On doit montrer que M appartient à $\mathcal{X} \to \mathcal{Y}$, c'est-à-dire que pour tout $N \in \mathcal{X}$, on a $\mathtt{app}(M, N) \in \mathcal{Y}$. Mais $\mathtt{app}(M, N) \to_{wh} \mathtt{app}(M', N) \in \mathcal{Y}$, et puisque $\mathcal{Y}$ est saturé on en déduit $\mathtt{app}(M, N) \in Y$. □

Lemme 17. *Si* $\mathcal{X}$ *et* $\mathcal{Y}$ *sont des* $\mathcal{T}$*-candidats, alors* $\mathcal{X} \to \mathcal{Y}$ *est un* $\mathcal{T}$*-candidat.*

Démonstration. L'inclusion $\mathsf{Ne}_{\mathcal{T}} \subseteq \mathcal{X} \to \mathcal{Y} \subseteq \mathsf{No}_{\mathcal{T}}$ est conséquence de la propriété 50.

La saturation découle de la propriété 51. On doit maintenant montrer que $\mathsf{Ne}_{\mathcal{T}} \subseteq X \to Y \subseteq \mathsf{No}_{\mathcal{T}}$, en sachant que $\mathsf{Ne}_{\mathcal{T}} \subseteq X \subseteq \mathsf{No}_{\mathcal{T}}$ et $\mathsf{Ne}_{\mathcal{T}} \subseteq Y \subseteq \mathsf{No}_{\mathcal{T}}$ puisque X et Y sont des $\mathcal{T}$-candidats. On a $\mathsf{Ne}_{\mathcal{T}} \subseteq \mathsf{No}_{\mathcal{T}} \to \mathsf{Ne}_{\mathcal{T}} \subseteq X \to Y$ par les propriétés 38 et 45 et $X \to Y \subseteq \mathsf{Ne}_{\mathcal{T}} \to \mathsf{No}_{\mathcal{T}} \subseteq \mathsf{No}_{\mathcal{T}}$ par les propriétés 38 et 46. □

On prouve enfin le théorème de normalisation via la même stratégie qu'à la section 10.2.3. On définit une *relation logique* unaire, c'est-à-dire qu'on interprète chaque type par un ensemble de termes. Plus précisément, on interprète chaque type A par un $\mathcal{T}$-candidat $|A|$ comme suit.

$$|\mathtt{Nat}| = \mathsf{No}_{\mathcal{T}}$$
$$|A \to B| = |A| \to |B|$$

On étend cette relation logique aux contextes, interprétés comme des ensembles de substitutions, de façon complètement standard.

$$|\cdot| = \{\mathtt{id}\}$$
$$|\Gamma, x : A| = \{\sigma, x\backslash M \mid \sigma \in |\Gamma|, M \in |A|\}$$

Lemme 18. *Soit X un $\mathcal{T}$-candidat. Soient M_1 fortement normalisable et $M_2 \in X$. Soit M_3 tel que pour tout N fortement normalisable et tout P appartenant à X, le terme $M_3[x\backslash N, y\backslash P]$ appartient à X. Alors $\mathtt{fold}_{\mathtt{Nat}}(M_1, M_2, (x, y).M_3)$ appartient à X.*

Lemme 19 (Adéquation)**.** *Soit $\Gamma \vdash M : A$ un terme. Si $\gamma \in |\Gamma|$ alors $M[\gamma] \in |A|$.*

Démonstration. Par induction sur la dérivation de typage.

- Cas $\dfrac{\Gamma \supseteq x : A}{\Gamma \vdash \mathtt{var}(x) : A}$: par une induction de routine sur la dérivation de $\Gamma \supseteq x : A$.

- Cas $\dfrac{\Gamma, x : A_1 \vdash M_1 : A_2}{\Gamma \vdash \mathtt{fun}(x.M_1) : A_1 \to A_2}$: on doit montrer que $\mathtt{app}(\mathtt{fun}(x.M_1)[\sigma], M_2) = \mathtt{app}(\mathtt{fun}(x.M_1[\sigma, x\backslash x]), M_2) \in |A_2|$ pour tout $M_2 \in |A_1|$. Puisque $|A_1|$ est un $\mathcal{T}$-candidat, M_2 est fortement normalisable. Donc

  $$\mathtt{app}(\mathtt{fun}(x.M_1[\sigma, x\backslash x]), M_2) \to_{wh} M_1[\sigma, x\backslash M_2].$$

  On a $\sigma, x\backslash M_2 \in |\Gamma, x : A_1|$ et donc $M_1[\sigma, x\backslash M_2] \in |A_2|$ par application de l'hypothèse d'induction. On conclut $M_1[\sigma, x\backslash M_2] \in |A_2|$ par saturation de $|A_2|$.

- Cas $\dfrac{\Gamma \vdash M_1 : A_1 \to A \qquad \Gamma \vdash M_2 : A_1}{\Gamma \vdash \mathtt{app}(M_1, M_2) : A}$: immédiat par application des hypothèses d'induction et définition de $|A_1| \to |A|$.

- Cas $\dfrac{}{\Gamma \vdash \mathtt{zero} : \mathtt{Nat}}$: immédiat puisque $\mathtt{zero}[\sigma] = \mathtt{zero}$ est normal et a fortiori fortement normalisable.

- Cas $\dfrac{\Gamma \vdash M_1 : \mathtt{Nat}}{\Gamma \vdash \mathtt{suc}(M_1) : \mathtt{Nat}}$: par application de l'hypothèse d'induction, on a $M_1[\sigma] \in |\mathtt{Nat}| = \mathsf{No}_{\mathcal{T}}$. On conclut par la propriété 40.
- Cas $\dfrac{\Gamma \vdash M_1 : \mathtt{Nat} \qquad \Gamma \vdash M_2 : A \qquad \Gamma, x : \mathtt{Nat}, y : A \vdash M_3 : A}{\Gamma \vdash \mathtt{fold}_{\mathtt{Nat}}(M_1, M_2, (x, y).M_3) : A}$: on commence par déduire des hypothèse d'induction que $M_1[\sigma] \in \mathsf{No}_{\mathcal{T}}$, que $M_2[\sigma] \in |A|$ et que pour tout N fortement normalisable et tout $P \in |A|$, on a $M_3[\sigma, x\backslash x, y\backslash y][x\backslash N, y\backslash P] \in |A|$. Ce sont exactement les hypothèses du lemme 18, qui permet de conclure.

□

Théorème 10 (Normalisation forte)**.** *Si $\Gamma \vdash M : A$ alors M est fortement normalisant.*

Démonstration. Soit σ la substitution $x_1\backslash x_1, \ldots, x_n\backslash x_n$, où les x_i sont les variables de Γ. Toute variable x_i est neutre, donc appartient à $|A_i|$, et donc $\sigma \in |\Gamma|$. Par le lemme 19, on a $M[\sigma] \in |A| \subseteq \mathsf{No}_{\mathcal{T}}$. Donc $M = M[\sigma]$ est fortement normalisable. □

Exercices

* **Ex. 1 ---** Démontrer que la propriété $\mathsf{fv}(M[\sigma]) = (\mathsf{fv}(M) - \mathsf{bv}(\sigma)) \cup \mathsf{fv}(\sigma)$ est fausse.

* **Ex. 2 ---** Démontrer que la propriété $\mathsf{size}(M[\sigma]) = \mathsf{size}(M)$ est fausse.

* **Ex. 3 ---** Grouper les termes suivants par classe d'α-équivalence.

$\mathtt{fun}(x.x)$ $\mathtt{fun}(x.\mathtt{suc}(\mathtt{app}(x,y)))$ $\mathtt{fun}(z.\mathtt{suc}(\mathtt{app}(z,y)))$ $\mathtt{fun}(y.y)$

$\mathtt{fold}_{\mathtt{Nat}}(\mathtt{suc}(y),\mathtt{suc}(x),(x,y).y)$ $\mathtt{fun}(z.x)$ $\mathtt{fun}(y.\mathtt{zero})$ $\mathtt{fun}(y.\mathtt{suc}(\mathtt{app}(y,y)))$

$\mathtt{fold}_{\mathtt{Nat}}(\mathtt{suc}(y),x,(x,y).y)$ $\mathtt{fun}(x.\mathtt{zero})$ $\mathtt{fold}_{\mathtt{Nat}}(\mathtt{suc}(y),x,(z_1,z_2).z_2)$

* **Ex. 4 ---** Reformuler la β-équivalence comme un jugement inductif.

* **Ex. 5 ---** Démontrer que $\Gamma \vdash M' : A$ et $M \to_\beta M'$ n'entraîne pas $\Gamma \vdash M : A$.

* **Ex. 6 ---** Démontrer la direction ``seulement si'' du lemme 13.

* **Ex. 7 ---** Donner un exemple d'un système de réécriture abstrait qui soit confluent et normalisant sans être fortement normalisant.

* **Ex. 8 ---** Donner un exemple d'un système de réécriture abstrait qui soit confluent et normalisant sans avoir l'unicité des formes normales.

* **Ex. 9 ---** Rédiger la preuve de la propriété 38.

** Exercice 10 Substitution et liaison

Démontrer les propriétés 6 à 8.

* Exercice 11 Mesures

La relation de *sous-terme immédiat*, notée $M <_i N$, est définie inductivement par les sept axiomes exprimés concisément ci-dessous.

$$\begin{aligned} M, N &<_i \mathtt{app}(M,N) \\ M &<_i \mathtt{fun}(x.M), \mathtt{suc}(M) \\ M, N, P &<_i \mathtt{fold}_{\mathtt{Nat}}(M, N, (x,y).P) \end{aligned}$$

La relation de *sous-terme*, notée $M < N$, est la clôture transitive de la relation de sous-terme immédiat. Une *mesure* est une fonction $d : \mathsf{PreTerm}_{\mathcal{T}} \to \mathbb{N}$ telle que pour tous termes M et N tels que $M < N$, on ait $d(M) < d(N)$.

1. Démontrer que la fonction size est une mesure.
2. Donner un autre exemple de mesure que la fonction size.
3. Démontrer que la relation de sous-terme est irréflexive : $M \not< M$ pour tout M.

4. Soit $\mathbf{P}$ un ensemble de prétermes qui satisfait la propriété suivante.

$$\frac{\forall N, N < M \Rightarrow N \in \mathbf{P}}{M \in \mathbf{P}}$$

   Démontrer que $\mathbf{P}$ contient tous les prétermes.

5. Soit d une mesure. Un ensemble de prétermes $\mathbf{P}$ est dit *d-inductif* s'il satisfait la propriété suivante.

$$\frac{\forall N, (\forall M' <_i M, d(N) \leq d(M')) \Rightarrow N \in \mathbf{P}}{M \in \mathbf{P}}$$

   Démontrer que tout ensemble d-inductif qui contient les variables et `zero` contient tous les prétermes.

Dans les deux exercices qui suivent, on s'intéresse uniquement au sous-ensemble $\mathsf{PreTerm}_\lambda$ de $\mathsf{PreTerm}_\mathcal{T}$ qui correspond aux termes du langage appelé *λ-calcul pur*. Il s'agit du fragment du système T sans les constructions permettant de manipuler les entiers naturels, à savoir zéro, successeur et récursion primitive.

$$\mathsf{PreTerm}_\lambda \ni M, N, P ::= \mathtt{var}(x) \mid \mathtt{fun}(x.M) \mid \mathtt{app}(M, N)$$

L'extension des résultats de ces exercices au système T tout entier ne pose aucune difficulté, mais ne demande aucune idée supplémentaire.

* Exercice 12 Une définition alternative de l'α-équivalence

Le but de cet exercice est d'étudier une définition alternative de l'α-équivalence, moins algorithmique que celle donnée à la figure 10.2, mais mathématiquement plus plaisante.

Pour écrire cette définition, on introduit la notion suivante. Si $\mathbf{P}$ est une propriété des noms, alors on écrira que $\mathbf{P}$ *est vérifiée pour tout nom suffisamment frais*, noté $И x, \mathbf{P}(x)$, lorsqu'il existe un ensemble fini de noms A tel que $\mathbf{P}(x)$ est vérifiée pour tout nom x qui n'appartient pas à A. La définition formelle de $И x, \mathbf{P}(x)$ est donc

$$И x, \mathbf{P}(x) \quad \Leftrightarrow \quad \exists A \subseteq_{fin} \mathbb{A}, \forall x \in \mathbb{A}, x \notin A \Rightarrow \mathbf{P}(x).$$

La définition alternative de l'α-équivalence, que l'on notera $M \equiv'_\alpha N$, est définie inductivement par les règles suivantes.

$$\boxed{M \equiv'_\alpha N} \qquad \frac{}{x \equiv'_\alpha x} \qquad \frac{M_1 \equiv'_\alpha M_2 \qquad N_1 \equiv'_\alpha N_2}{\mathtt{app}(M_1, N_1) \equiv'_\alpha \mathtt{app}(M_2, N_2)} \qquad \frac{И y, M_1[x_1 \backslash y] \equiv'_\alpha M_2[x_2 \backslash y]}{\mathtt{fun}(x_1.M_1) \equiv'_\alpha \mathtt{fun}(x_2.M_2)}$$

Le but de l'exercice est de démontrer que $\equiv_\alpha$ et $\equiv'_\alpha$ coïncident.

1. Démontrer la transitivité de $\equiv'_\alpha$.

2. Démontrer que si $M_1 \equiv_\alpha M_2$ alors $M_1 \equiv'_\alpha M_2$.
3. Démontrer que si $M_1 \equiv'_\alpha M_2$ alors $M_1 \equiv_\alpha M_2$.

** Exercice 13 Syntaxe anonyme et indices de de Bruijn

La définition de l'α-équivalence donnée à la figure 10.2 peut être vue comme celle d'une fonction récursive qui parcourt deux termes et décide si oui ou non ils sont α-équivalents. Cette fonction est soit coûteuse en temps ou en espace, puisqu'elle exige, lors de la vérification de $\texttt{fun}(x_1.M_1) \equiv_\alpha \texttt{fun}(x_2.M_2)$, soit de parcourir M_1 et M_2 pour calculer l'ensemble de leurs variables libres, soit de stocker ces deux ensembles dans la représentation du terme.

Le but de cet exercice est de remplacer cette définition par une représentation où les occurrences de variables ne sont plus associées à des noms mais à des entiers qui désigne le lieur pertinent. Une telle approche est dite *anonyme* et permet de se dispenser totalement de la notion d'α-équivalence, par opposition à la représentation avec nom qui est dite *nominale*. Les *termes anonymes*, dûs au mathématicien hollandais Nicolaas Govert de Bruijn, obéissent à la grammaire suivante.

$$\mathsf{ATerm}_\lambda \ni M, N, P ::= \texttt{var}^{\mathsf{db}}(n) \mid \texttt{app}^{\mathsf{db}}(M, N) \mid \texttt{fun}^{\mathsf{db}}(M) \qquad (n \in \mathbb{N})$$

L'entier 0 dans $\texttt{var}^{\mathsf{db}}(0)$ signifie que cette occurence de variable désigne la variable liée par le dernier lieur au dessus de cette occurence. L'entier 1 dans $\texttt{var}^{\mathsf{db}}(1)$ désigne l'avant-dernier lieur, et ainsi de suite. Par exemple, le terme $\texttt{fun}^{\mathsf{db}}(\texttt{var}^{\mathsf{db}}(0))$ correspond au terme $\texttt{fun}(x.x)$ dans la syntaxe avec noms. On appelle ces entiers des *indices de de Bruijn*. Les occurences libres sont représentées par des indices supérieurs au nombre de lieurs englobants ; par exemple $\texttt{fun}^{\mathsf{db}}(\texttt{app}^{\mathsf{db}}(\texttt{var}^{\mathsf{db}}(0), \texttt{var}^{\mathsf{db}}(1)))$ a une variable libre.

Il est commode de définir un jugement inductif $k \vdash M$ qui exprime que le terme anonyme M dispose d'*au plus* k variables libres.

$$\textsc{dBVar}\ \frac{n < k}{k \vdash \texttt{var}^{\mathsf{db}}(n)} \qquad \textsc{dBApp}\ \frac{k \vdash M \qquad k \vdash N}{k \vdash \texttt{app}^{\mathsf{db}}(M, N)} \qquad \textsc{dBAbs}\ \frac{k+1 \vdash M}{k \vdash \texttt{fun}^{\mathsf{db}}(M)}$$

Le but de l'exercice est de définir deux fonctions

$$\mathsf{name} : \mathsf{Term}^0_\lambda \to \mathsf{ATerm}^0_\lambda \qquad \text{et} \qquad \mathsf{db} : \mathsf{ATerm}^0_\lambda \to \mathsf{Term}^0_\lambda,$$

où Term^0_λ et ATerm^0_λ désignent les termes clos nominaux et anonymes, et de montrer que ces deux fonctions soient inverses l'une de l'autre.

1. Donner le terme anonyme qui correspond chacun des termes avec noms suivants.

$$\texttt{fun}(x.\texttt{fun}(y.\texttt{app}(x, y))) \qquad \texttt{fun}(y.\texttt{fun}(x.\texttt{app}(y, x))) \qquad \texttt{fun}(y.\texttt{fun}(x.\texttt{app}(x, y)))$$

2. Démontrer que si $k \vdash M$ alors $k + 1 \vdash M$.

3. Définir une fonction

$$\mathsf{name} : \mathsf{ATerm}_\lambda \times \mathbb{A}^* \to \mathsf{PreTerm}_\lambda$$

telle que si $\mathsf{db}(M, \varepsilon) = \mathsf{db}(M)$ pour tout M clos, où $\mathbb{A}^*$ désigne l'ensemble des listes finies de noms.

4. Définir une fonction

$$\mathsf{db} : \mathsf{PreTerm}_\lambda \times (\mathbb{A} \to \mathbb{N}) \to \mathsf{ATerm}_\lambda$$

telle que $\mathsf{db}(M, (x \mapsto 0)) = \mathsf{db}(M)$ pour tout M clos, où la notation $x \mapsto 0$ désigne la fonction constante 0.

5. Démontrer que pour tous M et N clos, si $M \equiv_\alpha N$ alors $\mathsf{db}(M) = \mathsf{db}(N)$.
6. Démontrer que pour tout M clos, $\mathsf{name}(\mathsf{db}(M)) \equiv_\alpha M$.
7. En déduire que pour tous termes nominaux clos M et N, on a $M \equiv_\alpha N$ si et seulement si $\mathsf{db}(M) = \mathsf{db}(N)$.

* Exercice 14 Programmation en $\mathcal{T}$

1. Définir les termes clos $M_+, M_\times, M_{pow}$: $\mathtt{Nat} \to \mathtt{Nat} \to \mathtt{Nat}$ tels que, pour tout couple d'entiers p, q, on ait

$$M_+ \ \underline{p} \ \underline{q} \equiv_\beta \underline{p+q}$$
$$M_\times \ \underline{p} \ \underline{q} \equiv_\beta \underline{pq}$$
$$M_{pow} \ \underline{p} \ \underline{q} \equiv_\beta \underline{q^p}.$$

2. Définir un terme clos M_- : $\mathtt{Nat} \to \mathtt{Nat} \to \mathtt{Nat}$ tel que, pour tout couple d'entiers p, q, on ait

$$M_- \ \underline{p} \ \underline{q} \equiv_\beta \underline{p-q}.$$

3. Définir un terme M_{ack} tel que, pour tout couple d'entiers p, q, on ait

$$M_{ack} \ \underline{p} \ \underline{q} \equiv_\beta \underline{A(p,q)}$$

où A désigne la fonction d'Ackermann, définie mathématiquement comme suit.

$$A(0, q) = q + 1$$
$$A(1 + p, 0) = A(p, 1)$$
$$A(1 + p, 1 + q) = A(p, A(1 + p, q))$$

* Exercice 15 Une traduction de $\mathcal{T}$ en OCaml

Proposer une traduction de $\mathcal{T}$ en OCaml. Cette traduction doit prendre la forme de deux fonctions mathématiques, toutes deux notées $[\![-]\!]_{\mathsf{OCaml}}$. La première associe à tout

type A de $\mathcal{T}$ un type $[\![A]\!]_{\mathsf{OCaml}}$ d'OCaml. La seconde associe à tout terme $x_1 : B_1, \ldots, x_n : B_n \vdash M : A$ de $\mathcal{T}$ une expression $[\![M]\!]_{\mathsf{OCaml}}$ écrite en OCaml tel que

$$x_1 : [\![B_1]\!]_{\mathsf{OCaml}}, \ldots, x_n : [\![B_n]\!]_{\mathsf{OCaml}} \vdash [\![M]\!]_{\mathsf{OCaml}} : [\![A]\!]_{\mathsf{OCaml}}.$$

Votre traduction peut utiliser du code OCaml prédéfini que vous prendrez soin de décrire. De plus, votre traduction est soumise aux contraintes qui suivent.

- Elle doit être *correcte* : si $M \equiv_\beta \underline{n}$ alors l'expression $[\![M]\!]_{\mathsf{OCaml}}$ doit s'évaluer en une représentation de l'entier n en OCaml. La représentation exacte de n dépendra de votre traduction du type `Nat`.
- Elle doit être *fonctorielle* : la traduction d'un terme est définie uniformément à partir de celle de ses sous-termes. Autrement dit, on doit avoir

$$[\![M[x\backslash N]]\!]_{\mathsf{OCaml}} = [\![M]\!]_{\mathsf{OCaml}}[x\backslash [\![N]\!]_{\mathsf{OCaml}}].$$

- Elle doit terminer en temps *linéaire* en la taille du terme. Cette contrainte exprime que votre traduction correspond à un compilateur plutôt qu'à un interprète.

* Exercice 16 Extension de $\mathcal{T}$ avec booléens

Dans cet exercice, on se propose d'étendre $\mathcal{T}$ avec des booléens. On ajoute à la syntaxe les deux formes d'introduction et la forme d'élimination que vous connaissez bien.

$\mathsf{PreTerm}_{\mathcal{T}} \ni M, N, P ::=$		**Prétermes**
	$\mid \ldots$	
	$\mid$ `true`	Vrai
	$\mid$ `false`	Faux
	$\mid$ `if`(M, N, P)	Conditionnelle
$\mathsf{Type}_{\mathcal{T}} \ni A, B, C ::=$		**Types**
	$\mid \ldots$	
	$\mid$ `Bool`	Type booléen

1. Proposer les équation β exprimant la simplification de la conditionnelle lorsque son premier sous-terme est une forme d'introduction booléenne.
2. Proposer des règles de typage des nouvelles constructions. Étendre la preuve du lemme 7 aux nouvelles règles.
3. Définir une fonction de traduction $(\!|-|\!)$ de votre extension de $\mathcal{T}$ dans le langage de départ. Elle doit respecter les mêmes contraintes que celle de l'exercice 15.
4. Démontrer que votre traduction préserve l'équivalence β.
5. Étendre le modèle ensembliste pour traiter les nouvelles constructions.

* Exercice 17 Extension de $\mathcal{T}$ avec produits cartésiens

Dans cet exercice, on se propose d'étendre $\mathcal{T}$ avec des produits cartésiens, c'est-à-dire des paires. On ajoute à la syntaxe les formes d'introduction et d'élimination suivantes.

$\mathsf{PreTerm}_{\mathcal{T}} \ni M, N, P ::=$		**Prétermes**
	$\mid \ldots$	
	$\mid \mathtt{pair}(M, N)$	Paires
	$\mid \mathtt{fst}(M)$	Première projection
	$\mid \mathtt{snd}(M)$	Deuxième projection
$\mathsf{Type}_{\mathcal{T}} \ni A, B, C ::=$		**Types**
	$\mid \ldots$	
	$\mid A \times B$	Type produit

1. Proposer les équation β exprimant la simplification des projections lorsque leur sous-terme est une forme d'introduction booléenne.
2. Proposer des règles de typage des nouvelles constructions. Étendre la preuve du lemme 7 aux nouvelles règles.
3. Étendre le modèle ensembliste pour traiter les nouvelles constructions. Étendre la relation logique et la preuve du lemme 11 pour en déduire un théorème de canonicité étendu.

* Exercice 18 Exemples de systèmes de réécriture abstraite

Définir des systèmes de réécriture $R_1, \ldots, R_9$ tels que :

- R_1 est fortement normalisant mais pas fini,
- R_2 est fini mais pas normalisant,
- R_3 est fortement normalisant et confluent,
- R_4 est fortement normalisant mais pas confluent,
- R_5 n'est ni confluent ni normalisant,
- R_6 est confluent mais pas normalisant,
- R_7 est confluent et normalisant mais pas fortement normalisant,
- R_8 est normalisant mais ni confluent ni fortement normalisant,
- R_9 est localement confluent mais pas confluent.

** Exercice 19 Confluence locale et globale

Démontrer le lemme suivant.

Lemme 20. *Soit R un système de réécriture fortement normalisant. Alors, R est confluent si et seulement si il est localement confluent.*

11. Le langage PCF

Dans ce chapitre, on s'intéresse à un langage fonctionnel toujours très idéalisé mais plus proche des langages de programmation réels comme Haskell. Ce langage, PCF, pour *Programming with Computation Functions*, remplace la récursion primitive de $\mathcal{T}$ par la récursion générale, c'est-à-dire la possibilité de définir des fonctions récursives arbitraires. En échange d'une facilité de programmation accrue, on perd la normalisation : les termes bien typés de PCF peuvent diverger.

11.1. Syntaxe et sémantique opérationnelle

11.1.1. Syntaxe

On passe rapidement sur les définitions élémentaire de la syntaxe de $\mathcal{PCF}$ dans cette section dans la mesure où leur traitement est parfaitement identique à celui développé pour $\mathcal{T}$ en section 10.1. En particulier, tous les termes sont immédiatement considérés à renommage des variables liées près, et on ne détaille pas le traitement de la substitution.

Définition 62. L'ensemble $\mathsf{Term}_{\mathcal{PCF}}$ des *termes* et l'ensemble $\mathsf{Sub}_{\mathcal{PCF}}$ des *substitutions* de $\mathcal{PCF}$ sont définis inductivement par les grammaires présentées à la figure 11.1.

Les termes de $\mathcal{PCF}$ sont donc les mêmes que ceux de $\mathcal{T}$, à ceci près que la récursion primitive a été remplacée par la récursion générale $\texttt{fix}(M)$ et une construction de test. De plus, tout entier naturel n est représenté dans la syntaxe comme le terme $\texttt{lit}(n)$, ce qui va faciliter l'expression de la réduction. On abrègera parfois $\texttt{lit}(n)$ en n lorsqu'il est clair qu'on souhaite parler d'un terme de $\mathcal{PCF}$ plutôt que d'un entier naturel.

L'ensemble des types de PCF est identique à celui du système T, de même pour l'ensemble des contextes de typage. Toutefois, on les dénote $\mathsf{Type}_{\mathcal{PCF}}$ et $\mathsf{Con}_{\mathcal{PCF}}$ dans ce chapitre, pour insister sur le fait qu'on étudie $\mathcal{PCF}$ plutôt que $\mathcal{T}$. Le jugement d'affaiblissement de $\mathcal{PCF}$ est hérité de $\mathcal{T}$ puisque $\mathsf{Con}_{\mathcal{PCF}}$ et $\mathsf{Con}_{\mathcal{T}}$ sont identiques.

Définition 63 (Typage de $\mathcal{PCF}$)**.** Les jugements de typage des termes et de typage des substitutions sont définis inductivement à la figure 11.2.

On écrit $\mathcal{PCF}(\Gamma; A)$ pour l'ensemble des termes habitant le type A sous le contexte de typage Γ, et on abrège $\cdot \vdash M : A$ en $M : A$.

Lemme 21 (Substitution)**.** *Si* $\Gamma \vdash M : A$ *et* $\Delta \vdash \sigma : \Gamma$ *alors* $\Delta \vdash M[\sigma] : A$.

Démonstration. Induction de routine sur la dérivation de typage. □

$\mathsf{Term}_{\mathcal{PCF}} \ni M, N, P ::=$		**Termes**
	$\mid \mathtt{var}(x)$	Variable
	$\mid \mathtt{fun}(x.M)$	Abstraction
	$\mid \mathtt{app}(M, N)$	Application
	$\mid \mathtt{lit}(n)$	Entier littéral ($n \in \mathbb{N}$)
	$\mid \mathtt{suc}(M)$	Successeur
	$\mid \mathtt{match}_{\mathtt{Nat}}(M, N, x.P)$	Conditionnelle
	$\mid \mathtt{fix}(M)$	Récursion générale
$\mathsf{Sub}_{\mathcal{PCF}} \ni \sigma, \varphi ::=$		**Substitutions**
	$\mid \mathtt{id}$	Substitution identique
	$\mid \sigma, x \backslash M$	Liaison

FIG. 11.1. : Syntaxe de $\mathcal{PCF}$

$$\boxed{\Gamma \vdash M : A} \qquad \frac{\Gamma \supseteq x : A}{\Gamma \vdash \mathtt{var}(x) : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \mathtt{fun}(x.M) : A \to B}$$

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash \mathtt{app}(M, N) : B} \qquad \frac{}{\Gamma \vdash \mathtt{lit}(n) : \mathtt{Nat}} \qquad \frac{\Gamma \vdash M : \mathtt{Nat}}{\Gamma \vdash \mathtt{suc}(M) : \mathtt{Nat}}$$

$$\frac{\Gamma \vdash M : \mathtt{Nat} \qquad \Gamma \vdash N : A \qquad \Gamma, x : \mathtt{Nat} \vdash P : A}{\Gamma \vdash \mathtt{match}_{\mathtt{Nat}}(M, N, x.P) : A} \qquad \frac{\Gamma \vdash M : A \to A}{\Gamma \vdash \mathtt{fix}(M) : A}$$

$$\boxed{\Gamma \vdash \sigma : \Delta} \qquad \frac{\Delta \supseteq \Gamma}{\Delta \vdash \mathtt{id} : \Gamma} \qquad \frac{\Gamma \vdash \sigma : \Delta \qquad \Gamma \vdash M : A}{\Gamma \vdash \sigma, x \backslash M : \Delta, x : A}$$

FIG. 11.2. : Typage des termes et substitutions de $\mathcal{PCF}$

$$\frac{(M_1,\ldots,M_n)\in R}{(M_1,\ldots,M_n)\in\mathcal{PCF}[R]} \qquad \frac{(M_1,\ldots,M_n)\in\mathcal{PCF}[R] \qquad (N_1,\ldots,N_n)\in\mathcal{PCF}[R]}{(\mathtt{app}(M_1,N_1),\ldots,\mathtt{app}(M_n,N_n))\in\mathcal{PCF}[R]}$$

$$\frac{(M_1,\ldots,M_n)\in\mathcal{PCF}[R]}{(\mathtt{fun}(x.M_1),\ldots,\mathtt{fun}(x.M_n))\in\mathcal{PCF}[R]} \qquad \frac{(M_1,\ldots,M_n)\in\mathcal{PCF}[R]}{(\mathtt{suc}(M_1),\ldots,\mathtt{suc}(M_n))\in\mathcal{PCF}[R]}$$

$$\frac{(M_1,\ldots,M_n)\in\mathcal{PCF}[R] \qquad (N_1,\ldots,N_n)\in\mathcal{PCF}[R] \qquad (P_1,\ldots,P_n)\in\mathcal{PCF}[R]}{(\mathtt{match}_{\mathtt{Nat}}(M_1,N_1,x.P_1),\ldots,\mathtt{match}_{\mathtt{Nat}}(M_n,N_n,x.P_n))\in\mathcal{PCF}[R]}$$

$$\frac{(M_1,\ldots,M_n)\in\mathcal{PCF}[R]}{(\mathtt{fix}(M_1),\ldots,\mathtt{fix}(M_n))\in\mathcal{PCF}[R]}$$

FIG. 11.3. : Clôture $\mathcal{PCF}$-contextuelle d'une relation R

11.1.2. Clôture contextuelle

Définition 64 (Clôture $\mathcal{PCF}$-contextuelle d'une relation)**.** Si R est une relation n-aire sur les termes de $\mathcal{PCF}$, sa *clôture $\mathcal{PCF}$-contextuelle*, notée $\mathcal{PCF}[R]$, est la relation n-aire sur les termes de $\mathcal{PCF}$ définie inductivement par les règles donnée à la figure 11.3.

Définition 65 ($\mathcal{PCF}$-congruence)**.** Une *$\mathcal{PCF}$-congruence* R est une relation d'équivalence sur les termes de $\mathcal{PCF}$ telle que $\mathcal{PCF}[R]$ soit incluse dans R.

On va maintenant démontrer quelques propriétés utiles concernant la clôture $\mathcal{PCF}$-contextuelle vue comme une opération sur les relations. Dans ce qui suit, R et S désignent des relations quelconques de l'ensemble $\mathsf{Term}_{\mathcal{PCF}}$ dans lui-même.

Propriété 52. *$\mathcal{PCF}[Id_{\mathsf{Term}_{\mathcal{PCF}}}] = Id_{\mathsf{Term}_{\mathcal{PCF}}}$ et $\mathcal{PCF}[R;S] \subseteq \mathcal{PCF}[R];\mathcal{PCF}[S]$.*

Propriété 53. *Si $R \subseteq S$ alors $\mathcal{PCF}[R] \subseteq \mathcal{PCF}[S]$.*

Propriété 54. *$R \subseteq \mathcal{PCF}[R]$ et $\mathcal{PCF}[\mathcal{PCF}[R]] = \mathcal{PCF}[R]$.*

Propriété 55. *$\mathcal{PCF}[R^{op}] = \mathcal{PCF}[R]^{op}$.*

Propriété 56. *La clôture contextuelle d'une relation réflexive (respesctivement symétrique, transitive) est réflexive (respectivement symétrique, transitive).*

Lemme 22. *La plus petite $\mathcal{PCF}$-congruence contenant R est $\mathcal{PCF}[R^{s*}]$.*

Démonstration. On sait que R^{s*} est une relation d'équivalence par la propriété 24. La propriété 56 implique que $\mathcal{PCF}[R^{s*}]$ est également une relation d'équivalence, c'est donc une $\mathcal{T}$-congruence.

Supposons maintenant que S est une $\mathcal{PCF}$-congruence qui contient R. On veut montrer qu'elle contient aussi $\mathcal{PCF}[R^{s*}]$. On a $\mathcal{PCF}[R^{s*}] \subseteq \mathcal{PCF}[S^{s*}] \subseteq \mathcal{PCF}[S^*] \subseteq \mathcal{PCF}[S] \subseteq S$ où la première étape découle de la propriété 53 et les autres du fait que S est une $\mathcal{PCF}$-congruence. □

$\mathsf{Kon}_{\mathcal{PCF}} \ni K ::=$		**Contextes généraux**
	$\mid \square$	Trou
	$\mid \mathtt{fun}(x.K)$	Abstraction
	$\mid \mathtt{app}(K, N)$	
	$\mid \mathtt{app}(M, K)$	Application
	$\mid \mathtt{suc}(K)$	Successeur
	$\mid \mathtt{match}_{\mathtt{Nat}}(K, N, x.P)$	
	$\mid \mathtt{match}_{\mathtt{Nat}}(M, K, x.P)$	
	$\mid \mathtt{match}_{\mathtt{Nat}}(M, N, x.K)$	Conditionnelle
	$\mid \mathtt{fix}(K)$	Récursion générale
$\mathsf{WHKon}_{\mathcal{PCF}} \ni E ::=$		**Contextes de tête faibles**
	$\mid \square$	Trou
	$\mid \mathtt{app}(E, N)$	Application
	$\mid \mathtt{suc}(E)$	Successeur
	$\mid \mathtt{match}_{\mathtt{Nat}}(E, N, x.P)$	Conditionnelle

FIG. 11.4. : Contextes d'évaluation de $\mathcal{PCF}$

11.1.3. Contextes d'évaluation

On a vu au chapitre précédent deux relations permettant de réduire les termes de $\mathcal{T}$ par réécriture progressive, la réduction β et la réduction de tête faible. La première permet la réduction β atomique d'un sous-terme arbitraire. La seconde restreint fortement la première en interdisant par exemple de réduire dans un sous-terme argument ou sous une abstraction. Plus généralement, chaque ensemble de choix de sous-termes où la réduction atomique est autorisée donne lieu à une relation de réduction.

Cette observation suggère d'approcher la réécriture de termes directement via l'ensemble des sous-termes où la réduction β atomique est autorisée. Une formalisation intuitive et commode de cette notion est la notion de *contexte d'évaluation*. Les contextes d'évaluation, qui ne doivent pas être confondus avec les contextes de typage, sont des termes équipés d'un unique sous-terme distingué. L'emplacement de ce sous-terme est marqué par le symbole $\square$, qu'on appelle souvent le “trou”, de façon imagée.

Définition 66 (Contextes d'évaluation)**.** L'ensemble $\mathsf{Kon}_{\mathcal{PCF}}$ des *contextes généraux* est défini inductivement par la grammaire présentée à la figure 11.4, tout comme le sous-ensemble $\mathsf{WHKon}_{\mathcal{PCF}} \subseteq \mathsf{Kon}_{\mathcal{PCF}}$ des *contextes de tête faible*.

Définition 67. Étant donné un contexte général K et un terme M, on note $K[\square \backslash M]$ pour le terme obtenu en substituant l'unique trou de K par M. On écrit que K est *de type A sous Γ avec un trou de type $\Delta \vdash B$*, et on note $\square : (\Delta \vdash B); \Gamma \vdash K : A$, lorsque

pour tout terme M tel que $\Delta \vdash M : B$, on a $\Gamma \vdash K[\Box\backslash M] : A$.

Lemme 23. *Si* $\Gamma \vdash K[\Box\backslash M] : A$ *alors il existe* Δ *et* B *tels que* $\Delta \vdash M : B$ *et* $\Box : (\Delta \vdash B); \Gamma \vdash K : A$.

Démonstration. Par une induction de routine sur K. □

Tout contexte de tête faible est un contexte général, on s'autorisera donc à écrire $E[\Box\backslash M]$ et $\Box : (\Delta \vdash B); \Gamma \vdash E : A$ avec E un contexte de tête faible.

Remarque 21. Le lemme 23 concernant $K[\Box\backslash M]$ n'est vrai que parce que le trou apparaît exactement une fois dans K. En particulier, il n'est pas vrai pour la substitution ordinaire $N[x\backslash M]$ où x peut apparaître zéro ou plus d'une fois dans N.

11.1.4. Réductions

Définition 68 (β-réduction atomique)**.** La *réduction* β *atomique*, notée $\rhd_\beta$, est la relation binaire sur les termes définie par les quatre règles suivantes.

$$\mathtt{app}(\mathtt{fun}(x.M), N) \rhd_\beta M[x\backslash N] \tag{11.1}$$
$$\mathtt{match}_{\mathtt{Nat}}(\mathtt{lit}(0), N, x.P) \rhd_\beta N \tag{11.2}$$
$$\mathtt{match}_{\mathtt{Nat}}(\mathtt{lit}(n+1), N, x.P) \rhd_\beta P[x\backslash \mathtt{lit}(n)] \tag{11.3}$$
$$\mathtt{fix}(M) \rhd_\beta \mathtt{app}(M, \mathtt{fix}(M)) \tag{11.4}$$

Propriété 57. $\rhd_\beta$ *est déterministe.*

Définition 69 (Réduction β)**.** Le terme M se réduit en N modulo β, ce qui est noté $M \to_\beta N$, s'il existe un contexte général K et deux termes M_0 et N_0 tels que $M = K[M_0]$ et $N = K[N_0]$ avec $M_0 \rhd_\beta N_0$.

Théorème 11. $\to_\beta$ *est confluente.*

Démonstration. La méthode de Tait, Martin-Löf et Takahashi employée au chapitre précédent s'adapte sans difficulté à $\mathcal{PCF}$. □

Corollaire 9. $\to_\beta$ *vérifie la propriété de Church-Rosser, c'est-à-dire,* $\to_\beta^{s*} = {\to_\beta}^*; \to_\beta^{op*}$.

Définition 70 (Équivalence β)**.** La relation d'*équivalence* β entre termes, notée $M \equiv_\beta N$, est la clôture symétrique réflexive-transitive $\to_\beta^{s*}$ de la réduction β.

Propriété 58. $\mathcal{PCF}[{\to_\beta}^*] \subseteq {\to_\beta}^*$.

Propriété 59. $\to_\beta \subseteq \mathcal{PCF}[\rhd_\beta^=]$.

Propriété 60. *L'équivalence* β *est la plus petite* $\mathcal{PCF}$*-congruence contenant* $\rhd_\beta$.

Démonstration. Par le lemme 22 et le corollaire 9, il suffit de démontrer que les relations ${\to_\beta}^*;{\to_\beta}^{op*}$ et $\mathcal{PCF}[\rhd_\beta^{s*}]$ sont identiques. On montre l'inclusion dans les deux sens.

Pour $\mathcal{PCF}[\rhd_\beta^{s*}] \subseteq {\to_\beta}^*;{\to_\beta}^{op*}$, on remarque tout d'abord que $\rhd_\beta \subseteq \to_\beta$ et donc $\rhd_\beta^{s*} \subseteq \to_\beta^{s*} = {\to_\beta}^*;{\to_\beta}^{op*}$ où la dernière équation est conséquence du corollaire 9. Donc

$$\begin{aligned}
\mathcal{PCF}[\rhd_\beta^{s*}] &\subseteq \mathcal{PCF}[{\to_\beta}^*;{\to_\beta}^{op*}] \\
&= \mathcal{PCF}[{\to_\beta}^*];\mathcal{PCF}[{\to_\beta}^{op*}] && \text{par la propriété 52} \\
&= \mathcal{PCF}[{\to_\beta}^*];\mathcal{PCF}[{\to_\beta}^*]^{op} && \text{par la propriété 55} \\
&= {\to_\beta}^*;{\to_\beta}^{*op} && \text{par la propriété 58.}
\end{aligned}$$

Pour ${\to_\beta}^*;{\to_\beta}^{op*} \subseteq \mathcal{PCF}[\rhd_\beta^{s*}]$, on a

$$\begin{aligned}
{\to_\beta}^*;{\to_\beta}^{op*} &\subseteq \mathcal{PCF}[\rhd_\beta^{=}]^*;\mathcal{PCF}[\rhd_\beta^{=}]^{op*} && \text{par la propriété 59} \\
&\subseteq \mathcal{PCF}[\rhd_\beta^{s*}]^*;\mathcal{PCF}[\rhd_\beta^{s*}]^{op*} \\
&= \mathcal{PCF}[\rhd_\beta^{s*}];\mathcal{PCF}[\rhd_\beta^{s*}] && \text{par la propriété 56} \\
&= \mathcal{PCF}[\rhd_\beta^{s*}] && \text{par la propriété 56.}
\end{aligned}$$

□

On a donc montré que l'équivalence β, définie comme la plus petite relation d'équivalence qui contient la réduction β, coïncide avec la plus petite $\mathcal{PCF}$-congruence qui contient la réduction β atomique. On aurait bien sûr pu intervertir théorème et définition en définissant directement $\equiv_\beta$ comme la plus petite $\mathcal{PCF}$-congruence qui contient la réduction β atomique, comme au chapitre 10.

Le fait que la relation d'équivalence engendrée par la réduction β soit une $\mathcal{PCF}$-congruence est intrinsèquement lié à la possibilité de réduire un sous-terme arbitraire. Toutefois, cette définition reflète mal les implémentations de langages fonctionnels, où l'on s'interdit de réduire le corps d'une fonction dont on ne connaît pas l'argument, et où a jamais le choix de la prochaine réduction à effectuer. Pour pallier ce défaut, on introduit la *réduction de tête faible*, plus proche d'un mécanisme d'exécution.

Définition 71 (Réduction de tête faible)**.** Le terme M se réduit en N par réduction de tête faible, ce qui est noté $M \to_{wh} N$, s'il existe un contexte de tête faible E et deux termes M_0 et N_0 tels que $M = E[M_0]$ et $N = E[N_0]$ avec $M_0 \rhd_\beta N_0$.

Remarque 22. La réduction de tête faible est une stratégie d'évaluation proche de la mécanique employée dans les langages fonctionnels non-stricts comme Haskell.

Définition 72. On appelle *radical* un terme M_0 tel qu'il existe N_0 avec $M_0 \rhd_\beta N_0$. Une *décomposition de tête faible* d'un terme M est une paire (E, M_0) avec E un contexte de tête faible et M_0 un radical tels que $M = E[\Box \backslash M_0]$.

Propriété 61. *Tout terme admet au plus une décomposition de tête faible.*

Corollaire 10. $\to_{wh}$ *est déterministe.*

On termine cette sous-section avec les résultats habituels sur le typage, qui expriment que celui-ci est un invariant du calcul.

Lemme 24 (Réduction du sujet, $\rhd_\beta$)**.** *Si* $\Gamma \vdash M : A$ *et* $M \rhd_\beta N$ *alors* $\Gamma \vdash N : A$.

Démonstration. On raisonne par cas sur $M \rhd_\beta N$ puis on inverse l'hypothèse de typage. Les cas des équations (11.1) et (11.3) sont réglés par le lemme 21. □

Lemme 25 (Réduction du sujet, $\to_\beta$)**.** *Si* $\Gamma \vdash M : A$ *et* $M \to_\beta N$ *alors* $\Gamma \vdash N : A$.

Démonstration. Par hypothèse, il existe K, M_0 et N_0 tels que $M = K[\Box \backslash M_0]$ et $N = K[\Box \backslash N_0]$. Par le lemme 23, on a Δ et B tel que $\Delta \vdash M_0 : B$ et $\Box : (\Delta \vdash B); \Gamma \vdash K : A$. Par le lemme 24 on a $\Delta \vdash N_0 : B$, et donc $\Gamma \vdash K[\Box \backslash N_0] : A$. □

Corollaire 11 (Réduction du sujet, $\to_{wh}$)**.** *Si* $\Gamma \vdash M : A$ *et* $M \to_{wh} N$ *alors* $\Gamma \vdash N : A$.

Définition 73. Une *valeur* V est un terme de $\mathcal{PCF}$ qui obéit à la grammaire suivante.

$$\mathsf{Val}_{\mathcal{PCF}} \ni V, W ::= \mathtt{lit}(n) \mid \mathtt{fun}(x.M)$$

Lemme 26 (Progrès)**.** *Un terme* $M : A$ *est* $\to_{wh}$*-normal si et seulement c'est une valeur.*

Le théorème qui clôt cette section exprime qu'un terme clos bien typé soit diverge, soit termine sur une valeur du bon type. En termes informelle, il assure que l'exécution d'un programme, assimilée à la réduction de tête faible, ne peut pas terminer sur autre chose qu'une valeur du type attendu. Ce résultat a joué un rôle historiquement important dans l'étude des langages de programmation, mais est essentiellement une conséquence immédiate de la réduction du sujet (lemme 25).

Définition 74. Un terme M_0 *diverge*, ce qu'on note $M_0 \uparrow$, lorsqu'il existe une séquence infinie $(M_i)_{i \geq 0}$ telle que $M_i \to_{wh} M_{i+1}$.

Théorème 12 (Sûreté)**.** *Si* $M : A$ *alors soit* $M \uparrow$ *soit il existe* $V : A$ *tel que* $M \to_{wh}^* V$*. De plus, une telle valeur* V*, si elle existe, est unique.*

Démonstration. Puisque $\to_{wh}$ est déterministe, si M diverge alors il n'a pas de forme normale pour $\to_{wh}$ et donc il n'existe pas de tel V. Supposons que M ait une forme normale. Le corollaire 10 assure que celle-ci est unique. De plus, celle-ci est forcément de type A, par le lemme 25, et il s'agit forcément d'une valeur par le lemme 26. □

Corollaire 12. *Si* $M : \mathtt{Nat}$ *alors soit* M *diverge soit il existe un unique entier* n *tel que* $M \to_{wh}^* \mathtt{lit}(n)$.

11.1.5. Programmer en $\mathcal{PCF}$

Il est utile d'écrire quelques programmes en $\mathcal{PCF}$ pour se forger des intuitions au sujet du langage. En plus des conventions notationnelles pour l'écriture de termes introduites au chapitre 10, on adopte les raccourcis suivants.

$$\texttt{let}(M, x.N) \equiv \texttt{app}(\texttt{fun}(x.N), M)$$
$$\texttt{letrec}(x.(M, N)) \equiv \texttt{let}(\texttt{fix}(\texttt{fun}(x.M)), x.N)$$

On vérifie qu'on a bien les règles de typage dérivées qui suivent.

$$\frac{\Gamma \vdash M : A \qquad \Gamma, x : A \vdash N : B}{\Gamma \vdash \texttt{let}(M, x.N) : B} \qquad \frac{\Gamma, x : A \vdash M : A \qquad \Gamma, x : A \vdash N : B}{\Gamma \vdash \texttt{letrec}(x.(M, N)) : B}$$

Ces définitions vérifient les réductions de tête faible

$$\texttt{let}(M, x.N) \to_{wh} N[x \backslash M] \qquad \texttt{letrec}(x.(M, N)) \to_{wh} N[x \backslash \texttt{fix}(\texttt{fun}(x.M))]$$

et la liaison récursive valide en particulier l'équation β

$$\texttt{letrec}(x.(M, N)) \equiv_\beta N[x \backslash \texttt{letrec}(x.(M, M))].$$

On peut les utiliser pour définir simplement la fonction addition.

$$\texttt{letrec}(add.(\texttt{fun}(x.\texttt{fun}(y.\texttt{match}_{\texttt{Nat}}(x, y, z.\texttt{suc}(add\ z\ y)))), add\ \texttt{lit}(2)\ \texttt{lit}(3)))$$

Remarque 23. Du point de vue de l'utilisation, $\mathcal{PCF}$ peut être vu comme un très petit sous-ensemble du langage fonctionnel Haskell. On peut écrire des fonctions d'ordre supérieur, et ces fonctions sont partielles, au sens où elles peuvent diverger pour certaines valeurs de leur argument. De plus, comme en Haskell, les fonctions n'évaluent pas nécessairement leur argument. Ainsi, on a

$$\texttt{app}(\texttt{fun}(x.\texttt{lit}(0)), \Omega) \to_{wh} \texttt{lit}(0)$$

où Ω est le terme divergent $\texttt{fix}(\texttt{fun}(x.x))$.

Haskell, comme $\mathcal{PCF}$, est parfois dit “pur'', au sens où les programmes du langage n'ont pas d'effets de bord observables. Cette terminologie suppose que la partialité ne soit pas observable.

Une grande différence entre Haskell et $\mathcal{PCF}$ du point de vue de l'exécution réside dans la possibilité de partager les calculs. En $\mathcal{PCF}$, la réduction du terme

$$\texttt{fun}(x.add\ x\ x)\ (add\ \texttt{lit}(2)\ \texttt{lit}(2))$$

vers une valeur réduira deux fois le sous-terme $add\ \texttt{lit}(2)\ \texttt{lit}(2)$ vers sa valeur $\texttt{lit}(4)$. En Haskell, le calcul sera fait une seule fois via une mécanique complexe de mémorisation des résultats intermédiaires.

11.2. L'équivalence de programmes

11.2.1. L'équivalence contextuelle

On a vu au chapitre 10 une seule relation d'équivalence sur les termes, l'équivalence β. Celle-ci a du sens dès lors que la réduction β est confluente, ce qui est vrai de $\mathcal{T}$ comme de $\mathcal{PCF}$. Toutefois, il est assez clair en $\mathcal{PCF}$ que l'équivalence β n'est pas suffisante. Par exemple, les termes $M_1 = \texttt{fix}(\texttt{fun}(x.x))$ et $M_2 = \texttt{fix}(\texttt{fun}(x.\texttt{suc}(x)))$ divergent tous les deux, mais n'ont clairement pas de réduit commun, et donc ne peuvent pas être équivalents modulo β. Cet exemple suggère de chercher une définition plus générale de l'équivalence de termes.

Pour concevoir une notion générale d'équivalence de programmes, on peut commencer par raisonner au sujet des valeurs (au sens de la définition 73). L'équivalence de deux valeurs qui sont des abstractions est un problème essentiellement aussi compliqué que celui de deux termes arbitraires. En revanche, l'équivalence de deux valeurs qui sont des entiers littéraux est simple : deux valeurs $\texttt{lit}(n)$ et $\texttt{lit}(m)$ sont égales lorsque $n = m$. On peut partir de cette observation pour définir une forme d'équivalence pour les termes clos de type entier. Deux tels termes seront équivalents lorsqu'ils calculent tous les deux le même entier au sens de la réduction de tête faible.

Définition 75 (Équiconvergence au type Nat)**.** Deux termes M et N sont *équiconvergents au type* Nat, ce qu'on note $M \simeq_{\texttt{Nat}} N$, s'ils sont clos de type Nat et que

$$\forall n \in \mathbb{N}, M \to_{wh}^{*} \texttt{lit}(n) \Leftrightarrow N \to_{wh}^{*} \texttt{lit}(n).$$

Remarquons en particulier que le théorème 12 nous assure qu'un terme clos de type entier soit diverge, soit converge vers un entier. Donc la définition ci-dessus implique que M diverge si et seulement si N diverge.

On peut maintenant utiliser la notion de contexte général pour étendre la relation $\simeq_{\texttt{Nat}}$ à des types arbitraires. L'idée est que deux termes M et N de même type sont *contextuellement équivalents* lorsqu'il n'existe aucun contexte général K tel que $K[\Box\backslash M]$ et $K[\Box\backslash N]$ ne s'évaluent pas vers le même entier.

Définition 76 (Équivalence contextuelle)**.** Deux termes $\Gamma \vdash M : A$ et $\Gamma \vdash N : A$ sont *contextuellement équivalents*, ce qu'on note $\Gamma \vdash M \cong N : A$, lorsque

$$\forall K \in \mathsf{Kon}_{\mathcal{PCF}}, \Box : (\Gamma \vdash A); \cdot \vdash K : \texttt{Nat} \Rightarrow K[\Box\backslash M] \simeq_{\texttt{Nat}} K[\Box\backslash N].$$

Une intuition utile est de penser aux contextes K comme à des tests qui permettent d'observer le comportement de M et N sous la forme très simple d'un entier final. En ce sens, les termes M et N sont équivalents lorsqu'ils passent tous les tests avec le même résultat ou, autrement dit, si aucun test ne permet de les distinguer. Pour cette raison, on parle aussi d'équivalence *observationnelle*.

11.2.2. Le préordre contextuel

L'équivalence contextuelle entre M et N capture la possibilité d'*échanger* M et N dans un programme sans changer le résultat final. On peut aussi s'intéresser à une notion plus

fine, où on s'autorise à remplacer M par N mais pas l'inverse. Il suffit de revisiter les définitions précédentes pour remplacer les équivalences par des implications.

Définition 77 (Raffinement au type $\texttt{Nat}$)**.** Le terme M *raffine* le terme N *au type* $\texttt{Nat}$, ce qu'on note $M \preceq_{\texttt{Nat}} N$, s'ils sont clos de type $\texttt{Nat}$ et que

$$\forall n \in \mathbb{N}, M \to_{wh}^{*} \texttt{lit}(n) \Rightarrow N \to_{wh}^{*} \texttt{lit}(n).$$

Définition 78 (Préordre contextuel)**.** Un terme $\Gamma \vdash M : A$ *approxime contextuellement* un terme $\Gamma \vdash N : A$, ce qu'on note $\Gamma \vdash M \lesssim N : A$, lorsque

$$\forall K \in \mathsf{Kon}_{\mathcal{PCF}}, \Box : (\Gamma \vdash A); \cdot \vdash K : \texttt{Nat} \Rightarrow K[\Box \backslash M] \preceq_{\texttt{Nat}} K[\Box \backslash N].$$

L'approximation contextuelle capture bien l'idée que remplacer M par N dans un programme peut simplement conduire celui-ci à passer de la divergence à la convergence. Si l'on considère que seule la convergence peut être vraiment observée, c'est une notion raisonnable. En particulier, une compilateur pourrait raisonnablement s'autoriser à transformer M en N, mais pas l'inverse.

On peut considérer que l'approximation contextuelle est une notion plus fondamentale que l'équivalence contextuelle, puisqu'on peut définir facilement la seconde à partir de la première mais pas l'inverse.

Propriété 62. $\Gamma \vdash M \cong N : A$ *si et seulement si* $\Gamma \vdash M \lesssim N : A$ *et* $\Gamma \vdash N \lesssim M : A$.

En des termes plus abstraits, l'équivalence contextuelle est la relation d'équivalence induite par le préordre que constitue la relation d'approximation contextuelle.

La relation d'approximation contextuelle $\Gamma \vdash M \lesssim N : A$ est une notion complexe à appréhender, en particulier lorsque A est un type d'ordre élevé. Au type $\texttt{Nat}$, elle est assez simple, puisqu'on peut montrer qu'elle coïncide avec la relation $\preceq_{\texttt{Nat}}$. Au type $\texttt{Nat} \to \texttt{Nat}$, la situation est déjà plus subtile puisqu'un terme de ce type peut soit diverger immédiatement, soit converger vers une abstraction qui elle-même diverge lorsqu'appliquée à n'importe quel argument, soit diverge pour certaines entrées et converge pour d'autres, soit toujours converger. Le type $(\texttt{Nat} \to \texttt{Nat}) \to \texttt{Nat}$ est encore plus compliqué, ce qui témoigne de la combinatoire explosive de la relation d'approximation.

On peut cependant observer ou deviner quelques faits vrais de chacune des relations $\Gamma \vdash - \lesssim = : A$, et ce même si à ce stade nous ne disposons pas les outils mathématiques qui permettent de les démontrer. Tout d'abord, il s'agit clairement d'une relation réflexive et transitive. Ensuite, le terme $\Omega \equiv \texttt{fix}(\texttt{fun}(x.x))$ est minimal, tout comme $\texttt{fix}(\texttt{fun}(x.\texttt{suc}(x)))$, et la relation n'est donc pas antisymétrique. De plus, les deux termes

$$M_0 \equiv \texttt{fun}(x.\texttt{match}_{\texttt{Nat}}(x, \Omega, y.\texttt{lit}(3))) \qquad M_1 \equiv \texttt{fun}(x.\texttt{match}_{\texttt{Nat}}(x, \texttt{lit}(2), y.\Omega))$$

sont incomparables, au sens où l'on a ni $M_0 \lesssim M_1 : \texttt{Nat} \to \texttt{Nat}$ ni $M_1 \lesssim M_0 : \texttt{Nat} \to \texttt{Nat}$. Enfin, si $M : A \to B$ et que $N_0 \lesssim N_1 : A$, on devrait avoir $M\, N_0 \lesssim M\, N_1$, ce qui reflète que M est incapable de tester si son argument diverge.

11.2.3. L'attrait des modèles

La définition de l'équivalence et de l'approximation contextuelles, en plus d'être assez intuitive, ont l'avantage d'être faciles à adapter à d'autres langages de programmation. En revanche, elles sont difficiles à utiliser directement. S'il est facile de démontrer que deux termes M et N ne sont *pas* équivalents, puisqu'il suffit d'exhiber un contexte K qui les distingue, démontrer exige de montrer qu'il n'existe aucun K de la sorte.

Une approche naïve pour montrer $\Gamma \vdash M \cong N : A$ consisterait à procéder par induction sur les contextes. C'est malheureusement une approche trop frustre et très éloignée de la façon qu'on a de raisonner sur les programmes, même informellement.

À la place, on va réutiliser une approche vue pour $\mathcal{T}$ et bâtir un modèle de $\mathcal{PCF}$, c'est-à-dire une fonction d'interprétation $[\![-]\!]$ des types et des termes de $\mathcal{PCF}$ dans une famille d'objets mathématiques. Cette interprétation aura la propriété habituelle d'être un invariant du calcul, au sens où

$$M \to_{wh} N \quad \Rightarrow \quad [\![M]\!] = [\![N]\!]. \tag{11.5}$$

On va de plus montrer qu'elle aura la propriété suivante, beaucoup plus délicate.

$$[\![M]\!] = [\![N]\!] \quad \Rightarrow \quad \Gamma \vdash M \cong N : A. \tag{11.6}$$

Le modèle est donc un invariant du calcul, mais aussi un outil pour raisonner sur l'équivalence. Pour montrer l'équivalence de deux termes, il suffit donc en théorie de calculer leurs interprétations et de constater que celles-ci sont les mêmes.

Remarque 24. Le terme “calculer” ci-dessus est à prendre au sens mathématique plutôt qu'informatique, le calcul en question étant un raisonnement équationnel arbitrairement complexe. L'équivalence de termes $\mathcal{PCF}$ n'est pas décidable et ne se prête donc pas à un calcul algorithmique exact.

11.3. Un modèle dans les ensembles ordonnés inductifs

11.3.1. Intuitions

Un modèle de $\mathcal{PCF}$ doit autoriser certains principes de raisonnement s'il doit respecter la propriété équation (11.6). En particulier, on voudrait qu'il nous autorise à penser aux termes de types fonctionnels comme à des fonctions mathématiques, qui ont pour caractéristique d'être *extensionnelles*, c'est-à-dire carctérisées uniquement par leur comportement entrée/sortie. Deux fonctions sont égales lorsqu'elles envoient des arguments égaux sur des résultats égaux. Cela suggère de partir du modèle ensembliste de $\mathcal{T}$ vu en section 10.2.3.

Essayons d'adopter naïvement la sémantique ensembliste. On interprète comme précédemment le type des entiers par l'ensemble des entiers et le type des fonctions par l'ensemble des fonctions de l'interprétation du domaine dans l'interprétation du codomaine. Interpréter la plupart des constructions ne pose pas de difficulté, étant soit déjà présente en $\mathcal{T}$, soit une simplification d'une construction existante (la conditionnelle,

simplification de la récursion primitive). C'est bien entendu la récursion générale qui va poser problème.

Considérons un terme clos $M : \mathtt{Nat} \to \mathtt{Nat}$, et posons $M_0 = \mathtt{fix}(M)$. Si l'on adopte exactement les même définitions que dans le modèle ensembliste de $\mathcal{T}$, l'interprétation de M doit être une fonction de l'ensemble à un élément dans l'ensemble des fonctions entre entiers, c'est-à-dire essentiellement une fonction $f : \mathbb{N} \to \mathbb{N}$, et l'interprétation de M_0 un entier $x_0 \in \mathbb{N}$. Mais on doit avoir $M_0 \to_{wh} \mathtt{app}(M, M_0)$ et donc, puisque l'interprétation de M_0 est invariante par réduction, $x_0 = f(x_0)$. Autrement dit, il faut que toute fonction des entiers dans les entiers définissable par un terme de $\mathcal{PCF}$ a un point fixe. Or, tel quel, c'est absurde puisque par exemple le terme

$$\mathtt{fun}(x.\mathtt{match}_{\mathtt{Nat}}(x, \mathtt{lit}(1), y.\mathtt{lit}(0)))$$

correspond à la fonction qui envoie 0 dans 1 et $n+1$ dans 0. Encore plus simple, le terme $\mathtt{fun}(x.\mathtt{suc}(x))$ serait interprété par la fonction successeur dont un point fixe serait un entier n tel que $n = n + 1$.

Pour échapper à cette difficulté, on peut commencer par remarquer que notre notion de modèle traite de l'équivalence contextuelle mais pas de l'approximation. La relation $\Gamma \vdash - \lesssim = : A$ est clairement réflexive et transitive, ce qui suggère que l'objet mathématique $[\![A]\!]$ devrait être un ensemble muni d'une relation d'ordre, et d'exiger l'analogue de la propriété 11.6 pour cette relation :

$$[\![M]\!] \leq [\![N]\!] \quad \Rightarrow \quad \Gamma \vdash M \lesssim N : A. \tag{11.7}$$

Soit X l'ensemble ordonné interprétant un type A quelconque. Peut-on deviner les propriétés de X qui sont indépendantes de A ? Tout d'abord, puisque le terme Ω est minimal pour l'approximation contextuelle à tous les types, X doit contenir un élément minimal $\bot$ qui lui servira d'inteprétation. Ensuite, toute fonction $f : X \to X$ doit avoir un point fixe à partir du moment où elle est définissable en $\mathcal{PCF}$. Il est clair qu'une telle fonction doit au minimum être monotone, puisqu'il est impossible de tester la divergence d'un argument. De plus, si $f = [\![M]\!]$ a plusieurs point fixe, on doit disposer d'un choisir un bien précis pour interpréter $\mathtt{fix}(M)$. Appelons y_0 ce choix. On doit avoir

$$\begin{aligned}
\bot &\leq y_0 && \text{car } \bot \text{ est minimal} \\
f(\bot) &\leq f(y_0) = y_0 && \text{car } f \text{ est monotone et } y_0 \text{ un point fixe de } f \\
f(f(\bot)) &\leq f(y_0) = y_0 \\
f^3(\bot) &\leq f(y_0) = y_0 \\
&\dots
\end{aligned}$$

et donc $f^n(\bot) \leq y_0$ pour tout $n \in \mathbb{N}$. Cette observation suggère qu'un choix canonique pour y_0 pourrait être la *limite* de la suite $(f^n(\bot))_{n\in\mathbb{N}}$, à supposer que celle-ci soit bien un point fixe de f. Ici, une limite est à prendre au sens de la théorie de l'ordre, c'est-à-dire un supremum.

11.3.2. Un peu de théorie des ordres

Définitions de base

Définition 79. Une *relation de préordre* sur un ensemble S est une relation $R : S \nrightarrow S$ réflexive et transitive. Une *relation d'ordre* sur S est une relation de préordre sur S qui est antisymétrique.

On parlera parfois de *préordre* ou d'*ordre* pour désigner une relation de préordre ou d'ordre.

Définition 80. Un *ensemble préordonné* X est une paire $(|X|, \leq_X)$ formée d'un ensemble $|X|$ et d'une relation de préordre $\leq_X$ sur $|X|$. L'ensemble est *ordonné* lorsque $\leq_X$ est une relation d'ordre.

Dans le reste de cette sous-section, X, Y et Z désignent des ensembles préordonnés. On écrira $\leq$ plutôt que $\leq_X$ lorsque X peut être déduit du contexte. Par abus de langage, on a tendance à identifier un ensemble préordonné ou ordonné et son ensemble d'éléments, et on parlera d'un élément de X ou d'une partie de X pour désigner un élément ou une partie de $|X|$.

Définition 81 (Fonction monotone)**.** Une fonction $f : X \to Y$ est *monotone* si pour tous $x, x' \in X$ tels que $x \leq_X x'$ on a $f(x) \leq_Y f(x')$.

Les fonctions monotones sont aussi appelées des fonctions *croissantes*.

Remarque 25. La littérature, en particulier la littérature francophone, utilise parfois le terme “monotone” pour désigner une fonction soit croissante, soit décroissante. On se conforme ici à l'usage en vigueur dans la littérature des langages de programmation et répandu en théorie des ordres [2].

Propriété 63. *La fonction identité $id : X \to X$ est monotone. La composition $gf : X \to Z$ de $f : X \to Y$ et $g : Y \to Z$ monotones aest monotone.*

Constructions sur les ensembles préordonnés et ordonnés

Relation d'ordre induite par un préordre Étant donnée une relation de préordre, on peut se demander comment lui associer une relation d'ordre de façon canonique. Il existe en général de nombreuses solutions, et on cherche une solution canonique, c'est-à-dire caractérisée comme la meilleure solution à un certain problème donné. Considérons la définition suivante.

Définition 82. Soit X un ensemble préordonné. On définit l'ensemble ordonné des *composantes fortement connexes* de X, dénoté $\mathsf{CC}\, X$, comme suit.

- Les éléments de $\mathsf{CC}\, X$ sont les classes d'équivalence d'éléments de $|X|$ pour la relation d'équivalence $\sim_X$ définie comme $\leq_X \cap \leq_X^{op}$, autrement dit $x \sim x'$ lorsque $x \leq_X x'$ et $x' \leq_X x$.

- La relation d'ordre $\leq_{\mathsf{CC}\,X}$ entre ces classes d'équivalences est héritée de X, autrement dit $[x_1]_\sim \leq [x_2]_\sim$ lorsque $x_1 \leq x_2$, où $[x]_\sim$ désigne la classe d'équivalence de x pour $\sim$.

Cette construction est canonique au sens où $\mathsf{CC}\,X$ est, informellement, le plus petit ensemble ordonné dans lequel X s'envoie par une fonction monotone. Il s'agit là de la *propriété universelle* de $\mathsf{CC}\,X$, qu'on exprime rigoureusement ci-dessous.

Propriété 64. *Soit X un ensemble préordonné et Y un ensemble ordonné. Soit $f : X \to Y$ une fonction monotone. Il existe une unique fonction monotone $f^\# : \mathsf{CC}\,X \to Y$ telle que $f = \iota; f^\#$, où $\iota : X \to \mathsf{CC}\,X$ est la fonction qui envoie x dans $[x]_\sim$.*

Corollaire 13. *L'ensemble des fonctions monotones de X dans Y, où Y est un ordre, est en bijection avec l'ensemble des fonctions monotones de $\mathsf{CC}\,X$ dans Y.*

Ordre dual. La construction suivante est aussi simple qu'utile. Elle permet notamment de ne pas dupliquer certains définitions, comme on le verra ultérieurement.

Définition 83. L'ensemble préordonné *dual* de X, noté X^{op}, a le même ensemble d'éléments que X et la relation de préordre donnée par $\leq_{X^{op}} = \leq_X{}^s$.

Propriété 65. *Si $f : X \to Y$ est une fonction monotone alors f est aussi une fonction monotone de X^{op} dans Y^{op}.*

On écrit f^{op} lorsqu'on veut insister sur le fait qu'on voit f comme une fonction de X^{op} dans Y^{op}, même si formellement f et f^{op} sont la même fonction.

Constructions libres. On se pose la question de munir un ensemble S quelconque d'une relation de préordre. Il existe deux choix canoniques : on peut décider de munir S de la plus petite relation de préordre possible, ou bien de la plus grande possible. Le premier choix correspond au préordre *discret*, le second au préordre *codiscret*, parfois aussia appelé *chaotique*.

Définition 84. Soit S un ensemble. La relation d'ordre *discrète* (resp. *codiscrète*) sur S est la relation identité sur S (resp. la relation totale sur S, c'est-à-dire $S \times S$). On écrit $\mathsf{Disc}\,S$ (resp. $\mathsf{CoDisc}\,S$) pour l'ensemble préordonné dont les éléments sont ceux de l'ensemble S et la relation de préordre est la relation discrète (resp. codiscrète).

La différence entre ces deux constructions est essentiellement résumée par les propriétés universelles suivantes.

Propriété 66. *L'ensemble des fonctions de S dans $|X|$ est en bijection avec l'ensemble des fonctions monotones de $\mathsf{Disc}\,S$ dans X. L'ensemble des fonctions de $|X|$ dans S est en bijection avec l'ensemble des fonctions monotones de X dans $\mathsf{CoDisc}\,S$.*

On remarque que la relation de préordre discrète est une relation d'ordre. Ce n'est pas vrai de la relation codiscrète, loin d'être antisymétrique. De plus, $\mathsf{CC}(\mathsf{CoDisc}\,S)$ est l'ensemble ordonné trivial a un seul élément ! Cet exemple montre l'intérêt des préordres.

Soulèvement. On voudra fréquemment s'assurer qu'un ensemble ordonne possède un élément *minimal*, c'est-à-dire plus petit que tous les autres.

Définition 85. L'ensemble préordonné $\uparrow X$ a pour ensemble d'éléments $|\uparrow X| = \langle\,\rangle \uplus \{\langle x\rangle \mid x \in |X|\}$ et pour relation de préordre la plus petite relation de préordre telle que $\langle\,\rangle \leq_{\uparrow X} a$ pour tout $a \in |\uparrow X|$ et $\langle x\rangle \leq_{\uparrow X} \langle x'\rangle$ pour tous $x, x' \in |X|$.

Propriété 67. *Si X est un ensemble ordonné alors $\uparrow X$ l'est aussi.*

Propriété 68. *La fonction $\langle -\rangle : X \to \uparrow X$ qui envoie x dans $\langle x\rangle$ est monotone.*

Propriété 69. *Soit $f : X \to Y$ une fonction monotone. La fonction $\langle f\rangle : \uparrow X \to \uparrow Y$ qui envoie $\langle\,\rangle$ sur $\langle\,\rangle$ et $\langle x\rangle$ sur $\langle f(x)\rangle$ est monotone.*

Produits cartésiens.

Définition 86 (Préordre produit)**.** Le produit cartésien de X et Y, noté $X \times Y$, a pour ensemble d'élément $|X| \times |Y|$, et pour relation d'ordre la relation telle que $(x, y) \leq_{X\times Y} (x', y')$ si et seulement si $x \leq_X y$ et $y \leq_Y y'$.

Propriété 70. *Si X et Y sont des ensembles ordonnés alors $X \times Y$ l'est aussi.*

Propriété 71. *La fonction π_1 (resp. π_2) de $|X| \times |Y|$ dans $|X|$ (resp. $|Y|$) qui envoie la paire (x, y) sur l'élément x (resp. y) est monotone.*

Propriété 72. *Soient $f : Z \to X$ et $g : Z \to Y$ deux fonctions monotones. La fonction $\langle f, g\rangle : Z \to X \times Y$ qui envoie $z \in Z$ dans $(f(z), g(z)) \in X \times Y$ est monotone.*

On appelle les fonctions π_i les *projections* du produit cartésien, et la fonction $\langle f, g\rangle$ le *pairage* de f et g.

Propriété 73. *Soient $f : Z \to X$, $g : Z \to Y$ et $h : Z \to X \times Y$ des fonctions monotones. On a les égalités suivantes.*

$$\pi_1 \circ \langle f, g\rangle = f \tag{11.8}$$
$$\pi_2 \circ \langle f, g\rangle = f \tag{11.9}$$
$$h = \langle \pi_1 \circ h, \pi_2 \circ h\rangle \tag{11.10}$$

Ces équations sont faciles à démontrer, et ne dépendent en réalité pas de la monotonie de f, g et h. On aurait pu énoncer les mêmes pour le modèle ensembliste de $\mathcal{T}$.

Préordres fonctionnels.

Définition 87. Le préordre des fonctions monotones de X et Y, noté $[X, Y]$, a pour ensemble d'éléments les fonctions monotones de X dans Y, et la relation d'ordre telle que $f \leq_{[X,Y]} g$ si et seulement si pour tout $x \leq_X x'$ on a $f(x) \leq_Y g(x')$.

Propriété 74. *Si X et Y sont des ensembles ordonnés alors $[X, Y]$ l'est aussi.*

Propriété 75. *La fonction dite d'*évaluation *de X dans Y, dénotée $ev_{X,Y} : [X,Y] \times X \to Y$, qui envoie la paire (f,x) sur $f(x)$ est monotone.*

Propriété 76. *Soit $f : Z \times X \to Y$ une fonction monotone. Pour tout élément z de Z, on dénote $f_z : X \to Y$ la fonction qui envoie z dans $f(z,x)$. On dénote $curry_{X,Y,Z}(f)$ la fonction qui envoie z dans f_z. Alors :*

1. *pour tout élément $z \in X$, $f_z : X \to Y$ est monotone,*
2. *$curry_{X,Y,Z}(f) : Z \to [X,Y]$ est monotone.*

Propriété 77. *Soient $f : Z \times X \to Y$, $g : Z \to X$ et $h : Z \to [X,Y]$ des fonctions monotones. On a les égalités suivantes.*

$$ev_{X,Y} \circ \langle curry_{X,Y,Z}(f), g \rangle = f \circ \langle id_Z, g \rangle \tag{11.11}$$

$$h = curry_{X,Y,Z}(ev_{X,Y} \circ \langle h \circ \pi_1, \pi_2 \rangle) \tag{11.12}$$

Suprema et infima

On va maintenant s'intéresser à l'existence de propriétés supplémentaires dans un ensemble ordonné ou préordonné. En particulier, l'existence de *plus petits majorants* et *plus grands minorants* fournit une variante de la notion usuelle de limite bien adaptée au cadre des ensembles ordonnés.

Définition 88 (Majorants, minorants)**.** Soit P une partie d'un préordre X. Un *majorant* de P dans X est un élément x de X tel que tout x' dans P est plus petit que x. Un *minorant* de P dans X est un majorant de P dans X^{op}.

On dit aussi que x *majore* P lorsque x est un majorant (resp. minorant) de P.

Définition 89 (Suprema, infima)**.** Soit X un préordre et P une partie de X. Un *supremum* (pluriel *suprema*) de P dans X est un élément x_0 de X qui est plus petit que tous les majorants de P dans X. Autrement dit, x_0 est tel que pour tout $x' \in X$, on a

$$(\forall x \in P, x \leq x') \quad \Leftrightarrow \quad x_0 \leq x'.$$

Un *infimum* (pluriel *infima*) de P est un supremum de P dans X^{op}.

Propriété 78. *Si $P \subseteq X$ a un supremum x_0, alors pour tout $x' \in P$, on a $x' \leq x_0$. De même, si $P \subseteq X$ a un infimum x_0, alors pout tout $x' \in P$, on a $x_0 \leq x'$.*

Propriété 79. *Si X est un ordre, alors une partie P de X a au plus un supremum (respectivement un infimum), que l'on note $\bigvee P$ (respectivement $\bigwedge P$) lorsqu'il existe.*

Soit $f : X \to Y$ une fonction monotone et P une partie de X. Si la partie $\{f(x) \mid x \in P\}$ de Y a un supremum, on désignera celui-ci par $\bigvee_{x \in P} f(x)$, et similairement pour les infimum.

Il est naturel de s'intéresser à des classes d'ensembles ordonnés disposant de tous les suprema et/ou infima pour les parties d'une certaine forme. Parmi les classes les plus populaires, on trouve des variantes de celle des treillis.

Définition 90. Soit X un ensemble ordonné. On dit que X est :

- un $\vee$-*demi-treillis* s'il dispose de tous les suprema finis non vides,
- un $\wedge$-*demi-treillis* s'il dispose de tous les infima finis non vides,
- un *treillis* s'il dispose de tous les suprema et infima finis non vides,
- un $\vee$-*demi-treillis borné* s'il dispose de tous les suprema finis,
- un $\wedge$-*demi-treillis borné* s'il dispose de tous les infima finis,
- un *treillis borné* s'il dispose de tous les suprema et infima finis,
- un *treillis complet* s'il dispose de tous les suprema.

Dans la définition ci-dessus, “suprema finis” ou “infima finis” signifie les suprema ou infima des *parties* finies. Plus généralement, le terme “suprema bleus” ou “infima bleus”, avec “bleu” une propriété quelconque, désigne des suprema des parties qui ont la propriété en question.

Propriété 80. *Un treillis complet dispose aussi de tous les infima.*

Les suprema et infima étant parfaitement duaux, on aurait également pu définir les treillis complets en utilisant les infima plutôt que les suprema.

11.3.3. Les ensembles ordonnés inductifs

Définitions et intuitions

Lorsque les ensembles ordonnés sont les interprétations des types d'un langage de programmation permettant la récursion générale, la relation d'ordre correspond au fait d'être mieux défini. Par exemple, dans le cas simple de l'ordre qui interprète le type des entiers de $\mathcal{PCF}$, le terme divergent Ω devrait être minimalement défini, tandis qu'un entier litéral devrait être interprété par un élément maximalement défini puisqu'il n'est pas possible de “converger plus”. Il est clair que la relation d'ordre aux types fonctionnels sera nettement plus complexe a décrire.

Nous allons exiger la présence de certains suprema pour être capable de rendre compte de certaines opérations du langage. Par exemple, puisque Ω est de type A, nous allons exiger la présence d'un minimum pour tout ordre qui va interpréter un type A. Or, le minimum d'un ordre est le supremum vide, qui doit donc exister. Mais quels autres suprema demander ?

Une solution serait de ne pas se poser la question et d'adopter les treillis complets comme modèles des langages de programmation. C'est une des solutions historiques. Toutefois, demander l'existence de tous les suprema nous éloigne a priori de la programmation. Reprenons l'exemple du type `Nat`, et supposons qu'il soit interprété dans un certain treillis complet $[\![\texttt{Nat}]\!]$. On peut supposer que les entiers littéraux `lit`(1) et `lit`(2) sont interprétés par les entiers 1 et 2. Le supremum $1 \vee 2$ existe puisque $[\![\texttt{Nat}]\!]$ est un

treillis complet et a fortiori un treillis. Mais il est loin d'être clair que ce supremum soit l'interprétation d'un terme clos de type `Nat`, et on souhaite autant que possible éviter d'avoir des éléments qui ne sont pas les interprétations de programmes.

On voudrait se donner, en plus du supremum vide, uniquement les supremum des parties qui tendent à l'infini vers un résultat. Mais comment préciser cette notion de convergence ? Une idée intuitive pour ce faire serait de considérer que de telles parties sont les *ω-chaînes* de X, c'est-à-dire les suites $(x_i)_{i\in\mathbb{N}}$ d'éléments de X avec $x_i \leq x_{i+1}$ pour tout i. C'est la bonne intuition, et on va légèrement généraliser cette définition au delà des ordres totaux grâce à la notion de partie *filtrante*.

Définition 91 (Ensemble ordonné filtrant, inductif)**.** Un ensemble ordonné est *filtrant* si chacune de ses parties finies a un majorant. On dit qu'une partie P de X est *filtrante dans* X lorsque P, munie de la relation d'ordre de X, est un ensemble ordonné filtrant. Un ensemble ordonné est dit *filtrant-complet* s'il dispose de tous les suprema filtrants. Il est dit *inductif* s'il est filtrant-complet et dispose de surcroît d'un plus petit élément.

Les propriétés suivantes ne sont pas difficiles mais peuvent permettre de mieux comprendre la notion de partie filtrante.

Propriété 81. *Une partie P de X est filtrante si et seulement si elle n'est pas vide et que pour tout x_1, x_2 dans P, il existe x dans P tel que $x_1 \leq x$ et $x_2 \leq x$.*

Propriété 82. *Une partie finie P de X est filtrante si et seulement si elle a un maximum.*

Propriété 83. *Toute ω-chaîne de X est une partie filtrante de X.*

On peut voir une partie filtrante P de X comme un ensemble de points qui converge progressivement vers un point idéal $x \in X$ situé à l'infini, sa limite au sens de la théorie des ordres, c'est-à-dire son supremum. Si P est fini alors la limite est atteinte au sens où x appartient à P, en vertu de la propriété 82. Si P est infini, ce n'est pas le cas, et le fait que X est inductif indique précisément que x existe dans X.

Remarque 26. En anglais, les ensemble *filtrants* sont dits *directed* (dirigés), et les ensembles ordonnés inductifs sont parfois appelés *directed-complete partial orders* (DCPO). On rencontre aussi le terme “ordre partiel complet”, remarquablement peu informatif. La terminologie “ensemble ordonné inductif” est due à Gordon Plotkin [6] et Paul Taylor.

Les programmes vont être interprétés comme des fonctions entre ensembles ordonnés inductifs qui commutent aux suprema filtrants. Soit P une partie de X et $f : X \to Y$ une fonction. On écrit $f(P)$ pour l'image directe de P par f, c'est-à-dire l'ensemble $\{f(x) \in \mathcal{Y} \mid x \in P\}$.

Propriété 84. *Si P est une partie filtrante de X et que $f : X \to Y$ est monotone, alors $f(P)$ est une partie filtrante de Y.*

Définition 92 (Fonction finitaire)**.** Soient X et Y des ensembles ordonnés filtrant-complets. Une fonction monotone $f : X \to Y$ est *finitaire* si elle commute aux suprema filtrants, autrement dit, si

$$f\left(\bigvee_{x\in P} x\right) = \bigvee_{x\in P} f(x) \tag{11.13}$$

pour toute partie filtrante P de X.

Notons que le supremum dans le membre droit de l'équation (11.13) existe en vertu de la propriété 84, sans quoi la définition n'aurait pas de sens.

Une fonction finitaire est parfois appelée *continue au sens de Scott* ou simplement *Scott-continue*. Il s'agit d'une propriété très importante qu'on peut voir comme une version abstraite de la notion de fonction calculable. Dans ce contexte, il faut penser au supremum x d'une partie filtrante P comme à un objet idéal infini et aux éléments de P comme à des objets finis. Si f est finitaire alors $f(x)$ est entièrement fixé par les résultats de f sur les éléments de P. Autrement dit, le comportement de f sur les objets infinitaires est entièrement dicté par son comportement sur les objets finitaires. L'infini est donc présent par commodité mais ne joue pas de vrai rôle dans les calculs.

Remarque 27. Il est possible de rendre cette intuition plus exacte en définissant une notion précise d'élément finitaire de X, et en exigeant que tout élément x de X soit le supremum des éléments finitaires qui lui sont inférieurs. On aboutit ainsi aux notions de *treillis algébrique* ou de *domaine de Scott*, qui dépassent le cadre de ces notes.

Propriété 85. *Soit X un ensemble ordonné filtrant-complet. La fonction identité $id : X \to X$ est finitaire. La composition $gf : X \to Z$ de $f : X \to Y$ et $g : Y \to Z$ finitaires est finitaire.*

Enfin, les fonctions finitaires ont une caractéristique essentielle lorsqu'il s'agit de modéliser un langage de programmation avec la récursion générale. Supposons que X soit un ensemble ordonné inductif dont nous notons $\bot$ le plus petit élément. Si f est une fonction monotone, alors la suite $\bot \leq f(\bot) \leq f(f(\bot)) \leq \dots$ est une ω-chaîne. Le supremum de cette suite est toujours le plus petit point fixe de f.

Théorème 13 (Kleene)**.** *Soit X un ensemble ordonné inductif. Si $f : X \to X$ est finitaire, alors f a un plus petit point fixe $\mathsf{fix}(f)$ défini par*

$$\mathsf{fix}(f) = \bigvee_{n \geq 0} f^n(\bot).$$

Démonstration. On commence par montrer que $\mathsf{fix}(f)$ est bien un point fixe de f. On a

$$f(\mathsf{fix}(f)) = f(\bigvee_{n \geq 0} f^n(\bot)) = \bigvee_{n \geq 0} f^{n+1}(\bot) = \bigvee_{n \geq 1} f^n(\bot) = \bigvee_{n \geq 0} f^n(\bot) = \mathsf{fix}(f).$$

Supposons maintenant que x soit un point fixe quelconque de f. Puisque $\bot$ est minimal dans X, on a $\bot \leq x$ et donc $f^n(\bot) \leq f^n(x) = x$ pour tout $n \geq 0$. Donc $\mathsf{fix}(f) = \bigvee_{n \geq 0} f^n(\bot) \leq x$ par la propriété universelle du supremum. □

Constructions

Dans ce qui suit X, Y, Z désignent des ensembles ordonnés inductifs.

Ordre discret, soulèvement, produits cartésiens. La plupart des constructions de la section 11.3.2 préservent l'inductivité. De plus, les fonctions monotones liées à ces constructions sont finitaires. Ainsi :

- l'ensemble ordonné discret $\mathsf{Disc}\, S$ sur un ensemble S est filtrant-complet, et la fonction monotone $f^{\#} : \mathsf{Disc}\, S \to X$ associée à $f : S \to X$ est finitaire,
- le soulèvement $\uparrow X$ d'un ensemble ordonné X filtrant-complet est toujours inductif, la fonction $\langle - \rangle : X \to \uparrow X$ est finitaire, la fonction $\langle f \rangle : \uparrow X \to \uparrow Y$ associée à une fonction $f : X \to Y$ est finitaire,
- le produit cartésien $X \times Y$ de deux ensembles filtrant-complets (resp. inductifs) X et Y est filtrant-complet (resp. inductif), les projections π_i sont finitaires et le pairage de deux fonctions finitaires est finitaire.
- l'ordre $\mathbf{1}$ a un seul élément () est inductif, et la fonction $!_X : X \to \mathbf{1}$ qui envoie tout $x \in X$ sur () est finitaire.

Le lemme suivant concernant les produits cartésiens est essentiel.

Lemme 27. *Soient X_1, X_2, Y des ensembles ordonnés filtrant-complets. Une fonction monotone $f : X_1 \times X_2 \to Y$ est finitaire si et seulement si elle est séparément finitaire, au sens où pour tout x_1 fixé la fonction $f(x_1, -) : X_2 \to Y$ est finitaire, et pour tout x_2 fixé la fonction $f(-, x_2) : X_1 \to Y$ est finitaire.*

Démonstration. La direction seulement si est évidente. Pour la direction si, supposons f séparément finitaire. Soit P une partie filtrante de $X_1 \times X_2$. On pose $P_i \equiv \pi_i(P)$ pour $i \in \{1, 2\}$. Ces parties sont filtrantes et on a $\bigvee P = (\bigvee P_1, \bigvee P_2)$.

On doit montrer $f(\bigvee P) = \bigvee_{x \in P} f(x)$. Puisque f est monotone, on a $\bigvee_{x \in P} f(x) \leq f(\bigvee P)$, et il suffit donc de montrer $f(\bigvee P) \leq \bigvee_{x \in P} f(x)$ pour établir l'égalité. Par propriété universelle du supremum, il suffit de montrer qu'on a $f(\bigvee P) \leq x'$ pour tout x' tel que $\bigvee_{x \in P} f(x) \leq x'$.

Soit x' tel que $f(x) \leq x'$ pour tout $x \in P$. Soient $x_1 \in P_1$ et $x_2 \in P_2$. Par définition, il existe $x_2' \in P_2$ et $x_1' \in P_1$ tels que $(x_1, x_2') \in P$ et $(x_1', x_2) \in P$. Comme P est filtrant, il doit exister $(x_1'', x_2'') \in P$ tel que $(x_1, x_2'), (x_1', x_2) \leq (x_1'', x_2'')$, et donc $f(x_1, x_2) \leq f(x_1'', x_2'') \leq x'$. Puisque f est finitaire en son premier argument, on a $f(\bigvee X_1, x_2) \leq x'$. Puisque f est finitaire en son deuxième argument, on a $f(\bigvee X_1, \bigvee X_2) \leq x'$. On conclut $f(\bigvee X) = f(\bigvee X_1, \bigvee X_2) \leq x'$. □

Ordre des fonctions finitaires. Un cas plus délicat est celui de l'ensemble ordonné $[X, Y]$ des fonctions monotones de X dans Y, avec X et Y inductifs. Pour que l'évaluation soit finitaire, on est amené à se restreindre au sous-ensemble ordonné des fonctions finitaires de X dans Y, qu'on note $[X, Y]_{\mathsf{fin}}$.

Propriété 86. $[X, Y]_{\mathsf{fin}}$ *est un ensemble ordonné inductif.*

Démonstration. Soit F une partie filtrante de $[X,Y]_{\mathsf{fin}}$. Soit x un élément de X. Puisque l'ordre sur les fonctions est point à point, la partie $F_x \equiv \{f(x) \mid f \in F\}$ de Y est filtrante, et son unique supremum $\bigvee_{f\in F} f(x)$ existe donc dans Y. On a ainsi défini une fonction g de X dans Y par $g(x) = \bigvee_{f\in F} f(x)$. Cette fonction est monotone puisque si $x \leq x'$ alors tout élément de F_x est inférieur à un élément de $F_{x'}$, et donc $g(x) = \bigvee F_x \leq \bigvee F_{x'} = g(x')$. Cette fonction est finitaire puisque si P est une partie filtrante de X, on a

$$g\left(\bigvee_{x\in P} x\right) = \bigvee_{f\in F} f\left(\bigvee_{x\in P} x\right) = \bigvee_{f\in F}\bigvee_{x\in P} f(x) = \bigvee_{x\in P}\bigvee_{f\in F} f(x) = \bigvee_{x\in P} g(x).$$

Supposons h tel que h majore F. Alors pour tout $x \in X$ et $f \in F$, on a $f(x) \leq h(x)$. Donc $g(x) = \bigvee_{f\in F} f(x) \leq h(x)$ et g est bien le supremum de F. □

Le morphisme d'évaluation et la curryfication d'un morphisme finitaire sont définis comme pour les ensembles ordonnés. Il faut simplement vérifier qu'ils sont finitaires.

Propriété 87. *Le morphisme d'évaluation $ev_{X,Y} : [X,Y]_{\mathsf{fin}} \times X \to Y$ est finitaire.*

Démonstration. Soit F une partie de $[X,Y]_{\mathsf{fin}}$ et P une partie de X. On a

$$\begin{aligned} ev_{X,Y}\left(\bigvee F, \bigvee X\right) &= \bigvee_{f\in F} f(\bigvee_{x\in X} x) \\ &= \bigvee_{f\in F}\bigvee_{x\in X} f(x) && (f \text{ est finitaire}) \\ &= \bigvee_{f\in F}\bigvee_{x\in X} ev_{X,Y}(f,x). \end{aligned}$$

□

Propriété 88. *Si $f : Z \times X \to Y$ est une fonction finitaire, alors :*

- *pour tout $z \in Z$ la fonction monotone $f_z : X \to Y$ est finitaire,*
- *$curry_{X,Y,Z}(f) : Z \to [X,Y]_{\mathsf{fin}}$ est finitaire.*

Démonstration. Conséquence immédiate du lemme 27. □

Opérateur de point fixe

Le théorème 13 exprime l'existence d'un plus petit point fixe pour toute fonction f de X dans X dès lors que X est inductif et f finitaire. Pour interpréter la récursion générale, il faut de plus que le calcul de ce plus petit point fixe soit lui-même finitaire.

Propriété 89. *La fonction $\mathsf{fix}_X : [X,X]_{\mathsf{fin}} \to X$ qui envoie f dans $\mathsf{fix}(f)$ est finitaire.*

Les entiers naturels. Soient X et Y deux ensembles ordonnés et y un élément de Y On écrit $const(y) : X \to Y$ pour la fonction qui envoie tout x sur y. Cette fonction est finitaire.

Propriété 90. *Soit X filtrant-complet. Il existe une fonction finitaire*

$$match_X : \uparrow \mathsf{Disc}\,\mathbb{N} \times (X \times [\mathsf{Disc}\,\mathbb{N}, X]_{\mathsf{fin}}) \to X$$

qui satisfait les équations suivantes.

$$match_X(\langle\,\rangle, f_0, f_s) = \bot_X \tag{11.14}$$
$$match_X(\langle 0\rangle, x_0, f_s) = x_0 \tag{11.15}$$
$$match(\langle n+1\rangle, x_0, f_s) = f_s(n) \tag{11.16}$$

11.3.4. Le modèle inductif

On peut maintenant utiliser toute la structure étudiée en section 11.3.3 pour interpréter $\mathcal{PCF}$ dans les ensembles ordonnés inductifs. Cette interprétation va beaucoup ressembler à celle de $\mathcal{T}$ dans les ensembles définie en section 10.2.3.

Chaque type A de $\mathcal{PCF}$ est interprété par un ensemble ordonné inductif $[\![A]\!]_{\mathsf{Ind}}$ défini par récurrence sur la structure de A.

Définition 93 (Modèle inductif, interprétation des types et contextes)**.** On interprète chaque type A (resp. contexte Γ) par un ensemble $[\![A]\!]_{\mathsf{Ind}}$ (resp. $[\![\Gamma]\!]_{\mathsf{Ind}}$) défini par récurrence sur sa structure comme suit.

$$[\![-]\!]_{\mathsf{Ind}} : \mathsf{Type}_{\mathcal{T}} \to \mathbf{Ind} \qquad [\![-]\!]_{\mathsf{Ind}} : \mathsf{Con}_{\mathcal{T}} \to \mathbf{Ind}$$
$$[\![\mathtt{Nat}]\!]_{\mathsf{Ind}} = \uparrow \mathsf{Disc}\,\mathbb{N} \qquad [\![\cdot]\!]_{\mathsf{Ind}} = \mathbf{1}$$
$$[\![A \to B]\!]_{\mathsf{Ind}} = [[\![A]\!]_{\mathsf{Ind}}, [\![B]\!]_{\mathsf{Ind}}]_{\mathsf{fin}} \qquad [\![\Gamma, x : A]\!]_{\mathsf{Ind}} = [\![\Gamma]\!]_{\mathsf{Ind}} \times [\![A]\!]_{\mathsf{Ind}}$$

Comme pour le modèle ensembliste de $\mathcal{T}$, les substitutions sont interprétées par des fonctions à valeur dans le produit cartésien, et le jugement d'affaiblissement par une projection. On ne répète pas ces définitions, identiques à ceci près qu'elles définissent des fonctions finitaires.

Définition 94 (Modèle inductif, interprétation des termes)**.** On interprète une dérivation de typage $\Gamma \vdash M : A$ par une fonction finitaire de $[\![\Gamma]\!]_{\mathsf{Ind}}$ dans $[\![A]\!]_{\mathsf{Ind}}$. La figure 11.5 présente les clauses de l'interprétation.

Lemme 28 (Fonctorialité)**.** $[\![\Delta \vdash M[\sigma] : A]\!]_{\mathsf{Ind}} = [\![\Gamma \vdash M : A]\!]_{\mathsf{Ind}} \circ [\![\Delta \vdash \sigma : \Gamma]\!]_{\mathsf{Ind}}$.

Théorème 14 (Invariance)**.** *Si $\Gamma \vdash M : A$ et $M \to_\beta M'$ alors*

$$[\![\Gamma \vdash M : A]\!]_{\mathsf{Ind}} = [\![\Gamma \vdash M' : A]\!]_{\mathsf{Ind}}.$$

Démonstration. C'est la conséquence directe du lemme 28 et de toutes les équations établies à la section 11.3.3. □

$$[\![-]\!]_{\mathsf{Ind}} : \mathcal{PCF}(\Gamma; A) \to [\![\Gamma]\!]_{\mathsf{Ind}} \to [\![A]\!]_{\mathsf{Ind}}$$

$$\left[\!\!\left[\frac{\Gamma \supseteq x : A}{\Gamma \vdash x : A} \right]\!\!\right]_{\mathsf{Ind}} = \pi_2 \circ [\![\Gamma \supseteq x : A]\!]_{\mathsf{Ind}}$$

$$\left[\!\!\left[\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash \mathtt{app}(M, N) : B} \right]\!\!\right]_{\mathsf{Ind}} = ev_{[\![A]\!]_{\mathsf{Ind}}, [\![B]\!]_{\mathsf{Ind}}} \circ \langle f, g \rangle \quad \text{où } f = [\![\Gamma \vdash M : A \to B]\!]_{\mathsf{Ind}},\ g = [\![\Gamma \vdash N : A]\!]_{\mathsf{Ind}}$$

$$\left[\!\!\left[\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \mathtt{fun}(x.M) : A \to B} \right]\!\!\right]_{\mathsf{Ind}} = \mathit{curry}([\![\Gamma, x : A \vdash M : B]\!]_{\mathsf{Ind}})$$

$$\left[\!\!\left[\frac{}{\Gamma \vdash \mathtt{zero} : \mathtt{Nat}} \right]\!\!\right]_{\mathsf{Ind}} = \mathit{const}(\langle 0 \rangle)$$

$$\left[\!\!\left[\frac{\Gamma \vdash M : \mathtt{Nat}}{\Gamma \vdash \mathtt{suc}(M) : \mathtt{Nat}} \right]\!\!\right]_{\mathsf{Ind}} = \langle \mathit{suc} \rangle \circ [\![\Gamma \vdash M : \mathtt{Nat}]\!]_{\mathsf{Ind}}$$

$$\left[\!\!\left[\frac{\Gamma \vdash M : \mathtt{Nat} \qquad \Gamma \vdash N : A \qquad \Gamma, x : \mathtt{Nat} \vdash P : A}{\Gamma \vdash \mathtt{match}_{\mathtt{Nat}}(M, N, x.P) : A} \right]\!\!\right]_{\mathsf{Ind}} = \mathit{match}_{[\![A]\!]_{\mathsf{Ind}}}(f, g, h) \quad \text{où } f = [\![\Gamma \vdash M : \mathtt{Nat}]\!]_{\mathsf{Ind}},\ g = [\![\Gamma \vdash N : A]\!]_{\mathsf{Ind}},\ h = [\![\Gamma, x : \mathtt{Nat} \vdash P : A]\!]_{\mathsf{Ind}}$$

$$\left[\!\!\left[\frac{\Gamma \vdash M : A \to A}{\Gamma \vdash \mathtt{fix}(M) : A} \right]\!\!\right]_{\mathsf{Ind}} = \mathsf{fix}_{[\![A]\!]_{\mathsf{Ind}}} \circ [\![M]\!]_{\mathsf{Ind}}$$

FIG. 11.5. : Modèle inductif de $\mathcal{PCF}$: termes

Pour établir le lemme qui suit, il faut de nouveau définir une relation logique entre les termes de $\mathcal{T}$ et les éléments du modèle.

Lemme 29 (Adéquation du modèle inductif)**.** *Soit* $\cdot \vdash M : \mathtt{Nat}$.

- *Si* $[\![\cdot \vdash M : \mathtt{Nat}]\!]_{\mathsf{Ind}}() = \langle\,\rangle$ *alors* $M \uparrow$.
- *Si* $[\![\cdot \vdash M : \mathtt{Nat}]\!]_{\mathsf{Ind}}() = \langle n \rangle$ *alors* $M \to_{wh}^{*} \mathtt{lit}(n)$.

Corollaire 14. *Soient* $\Gamma \vdash M : A$ *et* $\Gamma \vdash N : A$. *Si* $[\![\Gamma \vdash M : A]\!]_{\mathsf{Ind}} \leq [\![\Gamma \vdash N : A]\!]_{\mathsf{Ind}}$ *alors* $\Gamma \vdash M \lesssim N : A$.

Exercices

* **Ex. 20 ---** Expliciter la définition de $K[\Box\backslash M]$.

* **Ex. 21 ---** Démontrer le lemme 23. En déduire une définition inductive du jugement $\Box : (\Delta \vdash B); \Gamma \vdash K : A$.

* **Ex. 22 ---** Démontrer la propriété 61.

* **Ex. 23 ---** Démontrer le lemme 26.

* **Ex. 24 ---** Démontrer que la relation $\Gamma \vdash - \lesssim = : A$ équipe l'ensemble $\mathcal{PCF}(\Gamma; A)$ d'une structure de préordre.

* **Ex. 25 ---** Démontrer la propriété 66.

** **Ex. 26 ---** Démontrer la propriété 80.

* **Ex. 27 ---** Définir une construction qui ajoute un maximum à un (pré)ordre X uniquement en utilisant les constructions de la section 11.3.2.

* **Ex. 28 ---** Démontrer la propriété 82.

* **Ex. 29 ---** Soient X_1 et X_2 des ensembles ordonnés inductifs. Soit P une partie filtrante de $X_1 \times X_2$. Soit P_i la partie de X_i définie comme $\pi_i(P)$ pour $i\{1, 2\}$. Démontrer que P_1 et P_2 sont filtrantes, puis que $\bigvee P = (\bigvee P_1, \bigvee P_2)$.

* Exercice 30 Propriétés de la clôture contextuelle

Démontrer les propriétés 52 à 56.

*** Exercice 31 Évaluation à grands pas

On définit le jugement inductif $M \Downarrow V$ par les règles suivantes.

$$\boxed{M \Downarrow V} \qquad \frac{}{\texttt{fun}(x.M) \Downarrow \texttt{fun}(x.M)} \qquad \frac{}{\texttt{lit}(n) \Downarrow \texttt{lit}(n)}$$

$$\frac{M \Downarrow \texttt{fun}(x.P) \qquad P[x\backslash N] \Downarrow V}{\texttt{app}(M, N) \Downarrow V} \qquad \frac{M \Downarrow \texttt{lit}(n)}{\texttt{suc}(M) \Downarrow \texttt{lit}(n+1)} \qquad \frac{M \Downarrow \texttt{lit}(0) \qquad N \Downarrow V}{\texttt{match}_{\texttt{Nat}}(M, N, x.P) \Downarrow V}$$

$$\frac{M \Downarrow \texttt{lit}(n+1) \qquad P[x\backslash\texttt{lit}(n)] \Downarrow V}{\texttt{match}_{\texttt{Nat}}(M, N, x.P) \Downarrow V} \qquad \frac{\texttt{app}(M, \texttt{fix}(M)) \Downarrow V}{\texttt{fix}(M) \Downarrow V}$$

Les premières questions ci-dessous utilisent les termes définis à la section 11.1.5.

1. Construire une dérivation de $add\ \texttt{lit}(1)\ \texttt{lit}(2) \Downarrow \texttt{lit}(3)$.
2. Pourquoi ne peut-il pas y avoir de valeur V telle que $\Omega \Downarrow V$, informellement ? Même question pour le terme $\texttt{app}(\texttt{lit}(0), \texttt{lit}(0))$.

3. Démontrer que la relation $\Downarrow : \mathsf{Term}_{\mathcal{PCF}} \nrightarrow \mathsf{Val}_{\mathcal{PCF}}$ est déterministe.
4. Démontrer que si $M \Downarrow V$ alors $M \to_{wh}^* V$.
5. Démontrer que si $M \to_{wh}^* V$ alors $M \Downarrow V$.

* Exercice 32 Équivalences et inéquivalences de $\mathcal{PCF}$

Considérons les termes M_i : $\mathtt{Nat} \to \mathtt{Nat}$ ci-dessous. On rappelle que Ω désigne le terme $\mathtt{fix}(\mathtt{fun}(x.x))$.

$$
\begin{aligned}
M_1 &\equiv \mathtt{fun}(x.\mathtt{match}_{\mathtt{Nat}}(x, \Omega, y.1)) \\
M_2 &\equiv \mathtt{fun}(x.\mathtt{match}_{\mathtt{Nat}}(x, 42, y.\Omega)) \\
M_3 &\equiv \mathtt{fun}(x.\mathtt{match}_{\mathtt{Nat}}(x, 42, y.1)) \\
M_4 &\equiv \mathtt{fun}(x.\mathtt{match}_{\mathtt{Nat}}(x, 42, y.2)) \\
M_5 &\equiv \mathtt{fun}(x.\mathtt{match}_{\mathtt{Nat}}(x, 42, y.\mathtt{match}_{\mathtt{Nat}}(y, 12, z.\Omega))) \\
M_6 &\equiv \mathtt{fun}(x.\mathtt{match}_{\mathtt{Nat}}(x, 42, y.\mathtt{match}_{\mathtt{Nat}}(y, \Omega, z.13))) \\
M_7 &\equiv \mathtt{fun}(x.\mathtt{match}_{\mathtt{Nat}}(x, 42, y.\mathtt{match}_{\mathtt{Nat}}(y, 12, z.13))) \\
M_8 &\equiv \mathtt{fun}(x.\mathtt{match}_{\mathtt{Nat}}(x, 42, y.\mathtt{match}_{\mathtt{Nat}}(x, 42, z.\Omega))) \\
M_9 &\equiv \mathtt{fun}(x.\mathtt{match}_{\mathtt{Nat}}(x, 1, y.1)) \\
M_{10} &\equiv \mathtt{fun}(x.1)
\end{aligned}
$$

Donner les couples (M_i, M_j) tels que $M_i \lesssim M_j : \mathtt{Nat} \to \mathtt{Nat}$. Si $M_i \not\lesssim M_j : \mathtt{Nat} \to \mathtt{Nat}$, donner un contexte aussi simple que possible qui sépare les deux termes.

* Exercice 33 Relations d'ordre

Dans ce qui suit, S désigne un ensemble quelconque. On désigne par $\mathsf{POrd}(S)$ et $\mathsf{Ord}(S)$ par

$$
\begin{aligned}
\mathsf{POrd}(S) &\equiv \{R : S \nrightarrow S \mid R \text{ est une relation de préordre}\} \\
\mathsf{Ord}(S) &\equiv \{R : S \nrightarrow S \mid R \text{ est une relation d'ordre}\}.
\end{aligned}
$$

Pour chaque énoncé ci-dessous, déterminer s'il est vrai ou faux, et le cas échéant le prouver ou donner un contre-exemple.

1. $\leq_1 \cup \leq_2$ est une relation de préordre sur S.
2. $\leq_1 \cap \leq_2$ est une relation de préordre sur S.
3. Pour tout ensemble S, il existe une relation de préordre maximale sur S.
4. Pour tout ensemble S, il existe une relation d'ordre maximale sur S.

** Exercice 34 Composantes fortement connexes d'un préordre

1. Démontrer que la construction décrite par la définition 82 construit bien un ensemble ordonné.
2. Démontrer que l'application ι est monotone.

3. Démontrer la propriété universelle de $\mathsf{CC}\,X$, c'est-à-dire la propriété 64.
4. En déduire que $\mathsf{CC}\,X$ est l'unique ensemble ordonné possédant cette propriété universelle, à isomorphisme d'ordre près. (Un isomorphisme d'ordre est une bijection (f, f^{-1}) telle que f et f^{-1} sont des fonctions monotones.)

* Exercice 35 Treillis

1. Donner l'exemple d'un ensemble ordonné qui n'est pas un treillis.
2. Lister tous les treillis à un, deux, trois, et quatre éléments.
3. Démontrer qu'un treillis fini est nécessairement borné.
4. Donner l'exemple d'un treillis qui n'est pas un treillis borné.
5. Donner l'exemple d'un treillis borné qui n'est pas un treillis complet.

* Exercice 36 L'ordre partiel complet des suites

Liste des exercices

Table des figures

Index

Bibliographie

[1] Henk Barendregt. *The Lambda Calculus : Its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics. College Publications, 2012. isbn : 9781848900660. url : https://books.google.fr/books?id=b8jsMQEACAAJ (cf. p. 47).

[2] Brian A. Davey et Hilary A. Priestley. *Introduction to lattices and order*. Cambridge : Cambridge University Press, 1990. isbn : 0521365848. url : http://www.worldcat.org/search?qt=worldcat_org_all&q=0521367662 (cf. p. 95).

[3] Kurt Gödel. ``Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes''. In : *Dialectica* 12.3/4 (1958), p. 280-287. issn : 00122017, 17468361. url : http://www.jstor.org/stable/42964248 (cf. p. 37).

[4] Jean-Louis Krivine. *Lambda-calcul, types et modèles*. Masson, 1990. url : https://www.irif.fr/~krivine/articles/Lambda.pdf (cf. p. 39, 69).

[5] Andrew M. Pitts. *Nominal Sets*. Cambridge University Press, mai 2013. isbn : 9781139084673. doi : 10.1017/cbo9781139084673. url : http://dx.doi.org/10.1017/cbo9781139084673 (cf. p. 47).

[6] Gordon Plotkin. ``A Powerdomain Construction''. In : *SIAM Journal on Computing* 5.3 (sept. 1976), p. 452-487. issn : 1095-7111. doi : 10.1137/0205035. url : https://homepages.inf.ed.ac.uk/gdp/publications/Powerdomain_Construction.pdf (cf. p. 100).

[7] W. W. Tait. ``Intensional Interpretations of Functionals of Finite Type I''. In : *The Journal of Symbolic Logic* 32.2 (1967), p. 198-212. issn : 00224812. url : http://www.jstor.org/stable/2271658 (visité le 16/01/2023) (cf. p. 37).

[8] Masako Takahashi. ``Parallel reductions in -calculus''. In : *Journal of Symbolic Computation* 7.2 (fév. 1989), p. 113-123. issn : 0747-7171. doi : 10.1016/s0747-7171(89)80045-8. url : http://dx.doi.org/10.1016/s0747-7171(89)80045-8 (cf. p. 68).

Table des matières