Notions préliminaires

Ce chapitre est dédié à l'introduction de notions et résultats issus de la littérature sur lesquels s'appuie notre contribution. Outre des notations mathématiques de base, nous rappelons les notions de logique formelle et de théorie des modèles ; puis nous présentons le lambda-calcul, et en particulier la notion de typage simple et ses propriétés ; enfin, nous décrirons plusieurs systèmes issus de la théorie des langages qui s'avèrent pertinents pour répondre à des problèmes de formalisation linguistique. Nous préciserons pour chacune de ces notions l'importance qu'elle revêt dans le cadre de cette thèse.

2.1 Définitions

2.1.1 Notations

Le sous-ensemble $\{1, \ldots, n\}$ des entiers naturels $\mathbb{N}$ est abrégé en $[n]$. Le cardinal d'un ensemble E est noté $\#(E)$, et l'ensemble des parties de cet ensemble $\mathscr{P}(E)$. Si $T = (e_1 \ldots e_n)$ est un tuple, la i-ème projection de T est notée $\pi_i(T)$, pour $i \in [n]$. Si E est un ensemble, $\vec{\mathbb{N}}^E$ est l'ensemble des vecteurs de dimension E, où pour tout vecteur v, la notation $v.a$ désigne la composante associée à l'élément a de E dans v ; $\vec{0}$ désigne le vecteur nul, et $\vec{1}_a$ le vecteur dont la composante $v.a$ vaut 1, et nul partout ailleurs. Enfin, la notation $\|v\|$ désigne la norme supremum de v définie par $\|v\| = \max(v.a)$ pour $a \in E$.

La substitution d'une variable x par un élément a dans une expression E est notée $E[x \leftarrow a]$; par contraste, la substitution sans capture introduite dans la section 2.2.2 est notée $E[a/x]$.

Dans le cadre des systèmes de réécriture, si $\rightarrow$ est une relation, $a \xrightarrow{k} b$ dénote qu'il existe k éléments $a_1 \ldots a_k$ tels que $a \rightarrow a_1$ et $a_i \rightarrow a_{i+1}$ pour chaque $i \in [k-1]$, et $a_k = b$; implicitement, $k = 0$ dénote que $a = b$. Par extension, la relation $a \xrightarrow{*} b$ dénote la clôture réflexive et transitive de $\rightarrow$.

2.1.2 Alphabets, mots, langages

Soit Σ un ensemble de symboles. Nous considérons le monoïde libre (noté Σ^*) formé à partir de la base Σ par l'opération « . », qui est donc associative et dotée d'un élément neutre noté ε.

L'ensemble Σ est appelé *alphabet*, et ses éléments *lettres* ; les éléments de Σ^* sont appelés *mots*, l'opération « . » *concaténation* et son élément neutre le *mot vide*. Un ensemble (fini ou non) de mots sur Σ^* est appelé *langage*. L'ensemble Σ^* est appelé *langage universel*. Enfin, un ensemble de langages est appelé une *classe* de langages.

Pour tout mot w de Σ^* et tout entier naturel n, w^n dénote le mot formé en concaténant n fois le mot w ; par convention, $w^0 = \varepsilon$. Pour toute lettre l de Σ et tout mot w, $|w|_l$ dénote le nombre d'occurrences de l dans w ; et la longueur d'un mot est notée $|w| = \sum_{l \in \Sigma} |w|_l$.

Enfin, l'*image de Parikh* est une fonction (notée ψ) qui associe à tout mot w construit sur un alphabet Σ un vecteur v de $\vec{\mathbb{N}}^{\Sigma}$, tel que $v.l = |w|_l$ pour tout $l \in \Sigma$. Par extension, l'image de Parikh d'un langage L est définie comme l'ensemble des images de Parikh de ses mots : $\psi(L) = \{\psi(w) \mid w \in L\}$.

2.1.3 Alphabets gradués, termes, contextes

Un *alphabet gradué* est un ensemble de symboles, accompagné d'une fonction nommée *arité* (notée ρ) qui associe à tout élément de l'ensemble un entier naturel. Dans ce contexte, nous écrirons parfois s_k pour introduire un nouveau symbole s d'arité $\rho(s) = k$.

Si $\mathcal{S}$ est un alphabet gradué, l'ensemble des *termes* sur $\mathcal{S}$ est défini inductivement de la manière suivante : étant donné n termes $t_1 \dots t_n$ sur $\mathcal{S}$ et un symbole s de $\mathcal{S}$ tel que $\rho(s) = n$, $s(t_1, \dots, t_n)$ est un terme sur $\mathcal{S}$. Si s est d'arité zéro, nous écrirons parfois s au lieu de $s()$.

Une *position* dans un terme est un mot de $(\mathbb{N})^*$. Étant donné un terme t, une position p est dite *valide* dans t si et seulement si $p = \varepsilon$ ou si $p = ip'$, $t = s(t_1 \dots t_n)$ et $i \in [n]$ et que p' est une position valide dans t_i. L'ensemble des positions valides d'un terme t est appelé le *domaine* de t (noté $\mathrm{Dom}(t)$). Le *plus petit ancêtre commun* de deux positions p et p' dans un terme désigne la position construite comme le plus long préfixe commun à p et p'.

Étant donné une position p valide dans un terme t, le *sous-terme* de t à la position p (noté $t|_p$) est défini inductivement comme : $t|_p = t$ si $p = \varepsilon$; et $t|_p = t_i|_{p'}$ si $t = s(t_1 \dots t_n)$ et $p = ip'$; la définition des termes et de la validité d'une position garantit l'existence d'un tel sous-terme. Par analogie avec les arbres, une position valide désignant un sous-terme d'arité nulle est appelée une *feuille*, et la position ε la *racine*.

Considérons maintenant un ensemble dénombrable de variables $\mathcal{X}$ (notées $x, y, z, \dots$) : un *contexte* sur $\mathcal{S}$ est un terme construit sur l'alphabet gradué

$\mathcal{S} \cup \mathcal{X}$ où tous les éléments de $\mathcal{X}$ sont d'arité nulle. Tout sous-terme $x()$ dans un contexte C est appelé une *occurrence* de x dans C. Un contexte est dit *linéaire* s'il contient au plus une occurrence de x pour tout $x \in \mathcal{X}$.

Étant donné un contexte C, son *arité* $\rho(C)$ dénote le nombre de variables distinctes ayant des occurrences dans C ; un contexte d'arité k est également dit *k-aire*. Par suite, si $X = \{x_1 \dots x_n\}$ est l'ensemble ordonné des variables d'un contexte n-aire C, $C[t_1, \dots, t_n] = C[x_i \leftarrow t_i]$ dénote le résultat de la substitution simultanée de chacune des variables x_i par t_i dans C, pour $i \in [n]$. Celle-ci est distincte de la séquence de substitutions $C[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]$, en particulier si l'un des termes t_i contient une variable x_j avec $j > i$: aucune substitution n'est alors effectuée dans le terme substitué t_i. Enfin, si C est un contexte unaire et $n \in \mathbb{N}$, l'exponentiation $C^n = C[x \leftarrow C] \dots [x \leftarrow C]$ dénote le contexte obtenu en substituant $n - 1$ fois C à x (séquentiellement) ; ainsi, $C^1 = C$ et, par abus de notation, $C^0 = x$ est le contexte trivial.

2.1.4 Ensembles semilinéaires

Étant donné un ensemble E et un tuple de vecteurs $T = (v_0, \dots, v_n)$ appartenant chacun à $\vec{\mathbb{N}}^E$, l'*ensemble linéaire* défini par T est l'ensemble de vecteurs :

$$\left\{ v_0 + \sum_{i \in [n]} k_i v_i \mid \forall i \in [n], k_i \in \mathbb{N} \right\}$$

En outre, un ensemble de vecteurs est dit *semilinéaire* s'il peut être décrit comme une union finie d'ensembles linéaires de vecteurs. L'appartenance d'un vecteur v à un ensemble semilinéaire est décidable avec une complexité en espace logarithmique par rapport à la norme supremum $\|v\|$ de v (en choisissant le bon ensemble linéaire et en soustrayant itérativement un v_i judicieusement choisi jusqu'à aboutir à v_0).

2.1.5 Logique formelle

La logique formelle permet de formuler et d'évaluer la vérité de propositions portant sur une classe particulière d'objets. Un langage logique permet d'écrire des formules, lesquelles s'appuient sur un ensemble de prédicats et de relations en lien avec la classe d'objets étudiée. Différents types de logiques offrent divers degrés d'expressivité, ce qui revêt une importance particulière lorsqu'on considère, en théorie des modèles, les ensembles d'objets satisfaisant une formule en particulier. Nous définissons ici deux types de logiques, la logique du premier ordre et la logique du second ordre monadique, sur une signature quelconque, en rappelant la syntaxe et la sémantique usuelle des formules qu'elles construisent.

Signatures, formules Soit $\mathcal{X}_1$ et $\mathcal{X}_2$ deux ensembles dénombrables de variables, respectivement dites du *premier ordre* et du *second ordre*, et notées $x, y, z, \ldots$ et $X, Y, Z, \ldots$. Une *signature* est un ensemble de symboles nommés *prédicats* ou *relations* (notés $P, Q, R, \ldots$), munis de deux arités ρ_1 et ρ_2. Ces arités correspondent respectivement aux nombres de variables du premier et du second ordre dont dépend un prédicat. Nous emploierons souvent dans la suite le terme de *prédicat* par défaut, en réservant le terme de *relation* pour des prédicats dont l'arité est supérieure ou égale à deux.

Dans le cas général, une signature peut également contenir des *termes*, interprétés comme des constantes, ainsi que des *fonctions* s'appliquant à des séries de termes ou de variables, et munies d'une interprétation fonctionnelle de façon similaire à la sémantique donnée plus loin pour les prédicats. N'en ayant pas l'usage dans la suite, nous excluons ces derniers éléments de cette présentation. En outre, toute fonction dans une signature peut être remplacée de façon équivalente par un prédicat d'arité supérieure dénotant une relation fonctionnelle.

Si une signature ne contient que des symboles du premier ordre, c'est-à-dire que $\rho_2(P) = 0$ pour tout P, la *logique du premier ordre* sur cette signature décrit l'ensemble de *formules* suivant :

— Si P est un prédicat, alors les expressions $P(x_1, \ldots, x_{\rho_1(P)})$ et $x = y$ (égalité) sont des formules dites *atomiques*.

— Si ϕ, ψ sont des formules, alors les expressions $\neg\phi$ (négation), $\phi \wedge \psi$ (conjonction), $\phi \vee \psi$ (disjonction), $\phi \Rightarrow \psi$ (implication), $\phi \Leftrightarrow \psi$ (équivalence), $\exists x.\phi$ (quantification existentielle) et $\forall x.\phi$ (quantification universelle) sont des formules dites *composites*.

L'ensemble des formules de la *logique monadique du second ordre* est composé de la même façon, en y ajoutant d'éventuels prédicats portant sur des variables du second ordre, ainsi que les constructions suivantes pour les formules :

— Les expressions $x \in X$ (appartenance) et $X \subseteq Y$ (inclusion) sont des formules atomiques.

— Si ϕ est une formule, les expressions $\exists^2 X.\phi$ et $\forall^2 X.\phi$ sont des formules composites.

Le qualificatif de « monadique » dénote le fait que les variables du second ordre peuvent être vues comme des prédicats unaires sur les variables du premier ordre. Ainsi, une formule atomique d'appartenance $x \in X$ dénote simplement l'application $X(x)$ du prédicat dénoté par X à la variable x. Plusieurs autres formules atomiques données ici peuvent être définies à partir d'un ensemble plus réduit de primitives, comme l'illustre la sémantique donnée plus bas ; cependant, nous les définissons comme des primitives par simplicité.

Comme pour les termes, une *sous-formule* est formule apparaissant dans une formule composite. Si une formule ϕ est une quantification, la variable qui lui est rattachée est dite *liée* dans toute sous-formule de ϕ. Toute variable

apparaissant dans une formule sans être liée par un quantificateur est dite *libre* dans cette formule ; et une formule *close* est une formule ne contenant pas de variables libres.

Sémantique, modèles Toute formule logique construite sur une signature dénote une affirmation portant sur une classe d'objets, dont le sens dépend de l'interprétation associée aux prédicats de la signature et aux variables libres de la formule. L'approche de la théorie des modèles confronte les formules logiques aux objets dont elles traitent (les modèles), en associant à toute formule une valeur de vérité dépendant d'un modèle et de la valuation des éventuelles variables libres de la formule dans un domaine fixé.

Plus précisément, pour une signature donnée, un *modèle* $\mathcal{M} = (\mathcal{D}, [\![])$ est formé par un *domaine* $\mathcal{D}$ et une *interprétation* $[\![]$ qui associe à tout prédicat P de la signature une fonction $[\![P]\!]$ de $\mathcal{D}^{\rho_1(P)} \times \mathscr{P}(\mathcal{D})^{\rho_2(P)}$ dans $\mathbb{B} = \{\mathit{Vrai}, \mathit{Faux}\}$. Enfin, une *valuation* est une fonction partielle de $\mathcal{X}_1$ dans $\mathcal{D}$ et de $\mathcal{X}_2$ dans $\mathscr{P}(\mathcal{D})$.

Étant donné un modèle $\mathcal{M}$, une valuation ν et une formule ϕ sur la signature du modèle, la *satisfaction* de ϕ par $\mathcal{M}$ et ν (notée $\mathcal{M}, \nu \models \phi$) est définie ainsi :

- $\mathcal{M}, \nu \models P\big(x_1 \ldots x_{\rho_1}, X_1 \ldots X_{\rho_2}\big)$ ssi $[\![P]\!]\big(\nu(x_1) \ldots \nu(x_{\rho_1}), \nu(X_1) \ldots \nu(X_{\rho_2})\big)$
- $\mathcal{M}, \nu \models x = y$ ssi $\nu(x) = \nu(y)$
- $\mathcal{M}, \nu \models x \in X$ ssi $\nu(x) \in \nu(X)$
- $\mathcal{M}, \nu \models \neg\phi$ ssi $\mathcal{M}, \nu \not\models \phi$ ($\mathcal{M}$ et ν ne satisfont pas ϕ)
- $\mathcal{M}, \nu \models \phi \wedge \psi$ ssi $\mathcal{M}, \nu \models \phi$ et $\mathcal{M}, \nu \models \psi$
- $\mathcal{M}, \nu \models \exists x.\phi$ ssi il existe $v \in \mathcal{D}$ tel que $\mathcal{M}, \nu' \models \phi$ si ν' est la fonction définie par $\nu'(y) = \nu(y)$ pour tout $y \in \mathrm{Dom}(\nu) \setminus \{x\}$ et $\nu'(x) = v$.
- $\mathcal{M}, \nu \models \exists^2 X.\phi$ ssi il existe $V \subseteq \mathcal{D}$ tel que $\mathcal{M}, \nu' \models \phi$ si ν' est la fonction définie par $\nu'(Y) = \nu(Y)$ pour tout $Y \in \mathrm{Dom}(\nu) \setminus \{X\}$ et $\nu'(X) = V$.

La notion de satisfaction pour les formules construites autrement peut être décomposée pour ne reposer que sur les primitives listées ci-dessus, de la manière suivante :

- $\mathcal{M}, \nu \models X \subseteq Y$ ssi $\mathcal{M}, \nu \models \forall x.x \in X \Rightarrow x \in Y$
- $\mathcal{M}, \nu \models \phi \Leftrightarrow \psi$ ssi $\mathcal{M}, \nu \models (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$
- $\mathcal{M}, \nu \models \phi \Rightarrow \psi$ ssi $\mathcal{M}, \nu \models \neg\phi \vee \psi$
- $\mathcal{M}, \nu \models \phi \vee \psi$ ssi $\mathcal{M}, \nu \models \neg(\neg\phi \wedge \neg\psi)$
- $\mathcal{M}, \nu \models \forall x.\phi$ ssi $\mathcal{M}, \nu \models \neg\exists x.\neg\phi$
- $\mathcal{M}, \nu \models \forall^2 X.\phi$ ssi $\mathcal{M}, \nu \models \neg\exists^2 X.\neg\phi$

La notion de satisfaction d'une formule en théorie des modèles donne lieu à la notion de langage d'une formule. Intuitivement, étant donné une signature, le langage $\mathcal{L}(\phi)$ d'une formule ϕ construite sur cette signature est l'ensemble des modèles pour lesquels il existe une valuation satisfaisant la formule ϕ. En particulier, le langage d'une formule close ϕ est $\mathcal{L}(\phi) = \{\mathcal{M} \mid \mathcal{M}, \emptyset \models \phi\}$, où $\emptyset$ désigne la valuation dont le domaine est l'ensemble vide. Par suite, un langage

L est dit *définissable* lorsqu'il existe une formule ϕ telle que $L = \mathcal{L}(\phi)$; cette notion trouvera son utilité dans le cadre des langages réguliers de termes, dans la section 2.3.4.

2.2 Lambda-calcul

Le lambda-calcul est un système formel, proposé par Church [1936b], qui permet de modéliser des expressions fonctionnelles et leur évaluation. Il permet en général de modéliser n'importe quelle fonction calculable, mais son expressivité peut être restreinte par l'ajout de contraintes supplémentaires. En particulier, l'ajout d'un système de typage dit simple permet de garantir la terminaison du calcul, tout en préservant une certaine capacité à manipuler des expressions.

2.2.1 Lambda-termes

Nous définissons tout d'abord les lambda-termes, qui représentent des expressions fonctionnelles.

Définition 2.1. Un *lambda-terme* est un terme construit inductivement à partir d'un ensemble de variables $\mathcal{X}$ (notées $x, y, z, \dots$) selon les règles suivantes :
- Si $x \in \mathcal{X}$, alors x est un lambda-terme (appelé *variable*).
- Si M est un lambda-terme et $x \in \mathcal{X}$, alors $(\lambda x.M)$ est un lambda-terme (appelé *abstraction* de x).
- Si M_1 et M_2 sont des lambda-termes, alors $(M_1 M_2)$ est un lambda-terme (appelé *application* de M_1 à M_2).

Par convention de notation, les parenthèses inutiles sont souvent omises selon les règles suivantes. L'application est prioritaire sur l'abstraction – c'est-à-dire que le terme $\lambda x.xy$ s'interprète implicitement comme $\lambda x.(xy)$, et non comme $(\lambda x.x)y$. De plus, l'application est associative à gauche – c'est-à-dire que xyz signifie $(xy)z$ et non $x(yz)$.

Les notions de variables libres et liées sont définies de manière similaire à celles de la logique formelle, les abstractions tenant dans cette définition le même rôle que les quantificateurs en logique.

Définition 2.2. Étant donné un lambda-terme M, l'ensemble des variables *libres* (resp. *liées*) de M est noté $\mathrm{FV}(M)$ (resp. $\mathrm{BV}(M)$) et défini comme suit :
- Si $M = x$, alors $\mathrm{FV}(M) = \{x\}$ et $\mathrm{BV}(M) = \emptyset$.
- Si $M = \lambda x.N$, alors $\mathrm{FV}(M) = \mathrm{FV}(N) \setminus \{x\}$ et $\mathrm{BV}(M) = \mathrm{BV}(N) \cup \{x\}$.
- Si $M = M_1 M_2$, alors $\mathrm{FV}(M) = \mathrm{FV}(M_1) \cup \mathrm{FV}(M_2)$ et $\mathrm{BV}(M) = \mathrm{BV}(M_1) \cup \mathrm{BV}(M_2)$.

2.2.2 Sémantique opérationnelle

Les lambda-termes représentent des expressions fonctionnelles, où une abstraction dénote une définition de fonction, et une application représente l'application d'une fonction (à gauche) à son argument (à droite). Afin d'obtenir la sémantique recherchée pour l'évaluation d'une expression, le lambda-calcul définit plusieurs systèmes de réécriture sur les lambda-termes.

α-conversion

L'α-conversion est une relation qui dénote l'égalité entre deux expressions fonctionnelles modulo renommage de leurs paramètres – c'est à dire de leurs variables liées. Deux lambda-termes sont dits α-équivalents si l'un peut être obtenu à partir de l'autre par une série de renommages qui remplacent une variable dans la définition et dans le corps d'une fonction simultanément.

Définition 2.3. La relation de *α-renommage* entre deux lambda-termes M et M' (notée $M \to_\alpha M'$) est vérifiée dans les cas suivants :
- Si $M = \lambda x.N$ et que $M' = \lambda x.N'$ avec $N \to_\alpha N'$.
- Si $M = M_1 M_2$ et que $M' = M'_1 M_2$ avec $M_1 \to_\alpha M'_1$.
- Si $M = M_1 M_2$ et que $M' = M_1 M'_2$ avec $M_2 \to_\alpha M'_2$.
- Si $M = \lambda x.N$ et que $M' = \lambda y.N[x \leftarrow y]$ avec $y \notin \mathrm{FV}(N) \cup \mathrm{BV}(N)$.

Par extension, deux termes M et M' sont définis comme *α-équivalents* (noté $M =_\alpha M'$) si $M \overset{*}{\to}_\alpha M'$ ou si $M' \overset{*}{\to}_\alpha M$.

Les trois premiers cas dans la définition de $\to_\alpha$ stipulent que deux termes sont égaux par renommage s'ils peuvent être obtenus par renommage d'un sous-terme. Le dernier cas recouvre l'opération de renommage en elle-même, qui s'effectue directement par substitution ; afin de préserver le sens de l'expression fonctionnelle que dénote le terme renommé, cette substitution ne peut s'effectuer que si le nouveau nom de la variable (y) est absent du sous-terme où la substitution s'effectue.

Par la suite, nous adopterons la pratique usuelle consistant à considérer systématiquement des termes α-équivalents comme égaux, sans évoquer explicitement la notion d'α-conversion. Ce résultat peut être obtenu formellement en utilisant des indices de de Bruijn [1972] à la place des variables, qui dénotent la distance entre une variable et l'abstraction qui la lie : les termes ainsi obtenus sont indépendants de tout choix de dénomination de variables, et correspondent aux classes d'α-équivalence du lambda-calcul.

β-conversion

La β-contraction est une relation qui représente un « pas de calcul » dans l'évaluation d'une expression fonctionnelle. Le mécanisme de l'évaluation en lambda-calcul est la substitution d'une variable dans le corps d'une fonction

par l'argument auquel cette dernière est appliquée. Afin de préserver la sémantique des lambda-termes impliqués dans une évaluation, cette substitution doit également éviter la capture de variables. L'opération de β-contraction requiert donc une notion de substitution sans capture, s'appuyant au besoin sur un renommage des variables.

Définition 2.4. Une *substitution sans capture*, notée $M[N/x]$, est une substitution par N des occurrences libres de x dans M de sorte qu'aucune des variables libres de N n'est liée dans M. Cette opération peut impliquer une α-conversion implicite de M avant la substitution. Le terme $M[N/x]$ s'évalue donc inductivement comme suit :

— Si $M = x$, le résultat est N.
— Si $M = M_1 M_2$, le résultat est $M_1[N/x]\, M_2[N/x]$.
— Si $M = \lambda x.M'$, le résultat est M.
— Si $M = \lambda y.M'$ avec $y \neq x$ et $y \notin \mathrm{FV}(N)$, le résultat est $\lambda y.M'[N/x]$.
— Enfin, si $M = \lambda y.M'$ avec $y \neq x$ et que $y \in \mathrm{FV}(N)$, le résultat est $\lambda z.M'[y \leftarrow z][N/x]$; où z n'apparaît pas dans M ou dans N.

Le dernier cas de figure correspond au cas dans lequel le lambda-terme où a lieu la substitution requiert une α-conversion ; choisir une variable z « fraîche » résout dans ce cas les problèmes de capture éventuels.

Nous pouvons maintenant définir la β-contraction et ses relations dérivées.

Définition 2.5. Un *β-redex* est un lambda-terme formé d'une application dont la composante gauche est une abstraction.

La *β-contraction* relie un terme M contenant un β-redex $R = (\lambda x.P)N$ au terme M' obtenu par la substitution sans capture de x par N dans P. Formellement, la relation $M \to_\beta M'$ dénote que :

— Soit $M = (\lambda x.N)P$ et $M' = N[P/x]$.
— Soit $M = \lambda x.N$ et $M' = \lambda x.N'$ de sorte que $N \to_\beta N'$.
— Soit $M = M_1 M_2$ et $M' = M_1' M_2'$ de sorte que $M_1 = M_1' \wedge M_2 \to_\beta M_2'$ ou $M_1 \to_\beta M_1' \wedge M_2 = M_2'$.

La clôture réflexive/transitive $\xrightarrow{*}_\beta$ de $\to_\beta$ est nommée *β-réduction*, et la relation inverse *β-expansion*. La relation d'équivalence $=_\beta$ obtenue à partir de $\xrightarrow{*}_\beta$ est appelée *β-équivalence*.

Un lambda-terme ne contenant aucun β-redex est dit en *forme β-normale*.

Remarquons que les définitions de $\to_\alpha$ et $\to_\beta$ entraînent que le renommage des paramètres ne modifie pas le résultat de l'évaluation d'une fonction ; c'est à dire que si $M =_\alpha N$ et $M \to_\beta M'$, alors $N \to_\beta N'$ de sorte que $M' =_\alpha N'$.

De plus, la relation $\xrightarrow{*}_\beta$ présente une propriété importante, dite de Church-Rosser, qui garantit que tout choix entre deux redex dans l'opération de β-contraction n'est pas irréversible, en ce sens qu'il existe des réductions pour les termes résultants qui aboutissent au même terme.

Définition 2.6. Une relation $\rightarrow$ a la *propriété de Church-Rosser* si et seulement si $M \rightarrow M_1$ et $M \rightarrow M_2$ entraînent qu'il existe M' tel que $M_1 \rightarrow M'$ et $M_2 \rightarrow M'$.

Théorème 2.1. *La relation $\xrightarrow{*}_\beta$ a la propriété de Church-Rosser[cf. Barendregt, 1984, p. 59–62].*

Cette propriété entraîne immédiatement que si un lambda-terme M peut se réduire en N et que N est en forme β-normale, alors N est l'unique forme β-normale associée à M.

η-conversion

Enfin, la relation d'η-conversion dénote la vacuité d'une fonction f qui applique simplement une autre fonction f' à son argument, indépendamment de celui-ci. Elle permet d'éliminer dans ce cas l'abstraction et l'application inutiles, réduisant la fonction f à f'.

Définition 2.7. Un *η-redex* est un lambda-terme formé d'une abstraction de x sur l'application à x d'un terme où x n'est pas libre.

L'*η-contraction* relie un terme M contenant un η-redex $R = \lambda x.Nx$ au terme obtenu en substituant N à R dans M. Formellement, $M \rightarrow_\eta M'$ dénote :

- Soit $M = (\lambda x.M'x)$ et $x \notin \mathrm{FV}(M')$.
- Soit $M = \lambda x.N$ et $M' = \lambda x.N'$ de sorte que $N \rightarrow_\eta N'$.
- Soit $M = M_1 M_2$ et $M' = M'_1 M'_2$ de sorte que $M_1 = M'_1 \wedge M_2 \rightarrow_\eta M'_2$ ou $M_1 \rightarrow_\eta M'_1 \wedge M_2 = M'_2$.

Les notions d'*η-réduction* ($\xrightarrow{*}_\eta$), *η-équivalence* ($=_\eta$) et de *forme η-normale* sont définies de manière similaire à celles associées à la β-conversion.

En outre, les notions de *$\beta\eta$-contraction* ($\rightarrow_{\beta\eta}$), *$\beta\eta$-réduction* ($\xrightarrow{*}_{\beta\eta}$) et *$\beta\eta$-équivalence* ($=_{\beta\eta}$) sont définies pareillement pour le système produit à partir de la relation $M \rightarrow_{\beta\eta} M' \overset{def}{\Leftrightarrow} M \rightarrow_\beta M' \vee M \rightarrow_\eta M'$.

La relation $\xrightarrow{*}_\eta$ a trivialement la propriété de Church-Rosser, et cette propriété s'étend au système formé par $\xrightarrow{*}_{\beta\eta}$.

Théorème 2.2. *La relation $\xrightarrow{*}_{\beta\eta}$ a la propriété de Church-Rosser[cf. Barendregt, 1984, p. 65–67].*

2.2.3 Lambda-calcul simplement typé

Sans restriction, le lambda-calcul permet de modéliser le comportement de n'importe quelle fonction calculable. L'inconvénient majeur de cette expressivité est que l'existence d'un résultat final à l'évaluation d'un lambda-terme, c'est-à-dire d'une forme β-normale, n'est pas en général décidable. Ce fait est une conséquence immédiate de la principale propriété du problème de l'arrêt, qui stipule que le résultat de l'évaluation d'une fonction calculable est

indécidable en général[Turing, 1936][Church, 1936a]. Une restriction importante du lambda-calcul, qui permet de décider de l'équivalence entre deux termes, consiste à ne considérer que les termes dits simplement typés décrits par Church [1940], pour lesquels la terminaison du mécanisme d'évaluation est garantie. Nous décrivons d'abord deux méthodes classiques pour imposer cette restriction (des typages dits à la Curry et à la Church) et leurs conséquences, avant de préciser leur utilité dans la suite de la thèse.

Nous introduisons tout d'abord les notions de types simples et de signature typée.

Définition 2.8. Soit A un ensemble de symboles nommés *types atomiques*. L'ensemble $\mathcal{T}$ des *types simples* sur A (notés $\tau, \sigma, \dots$) est défini inductivement de la manière suivante : si $\tau \in A$, alors $\tau \in \mathcal{T}$; et si $\tau, \sigma \in \mathcal{T}$, alors $(\tau \to \sigma) \in \mathcal{T}$.

Une *signature* du lambda-calcul typé est formée par un ensemble $\mathcal{C}$ de *constantes* (notées par la suite $a, b, c, \dots$), et la définition d'un lambda-terme est étendue de sorte que toute constante est également un lambda-terme atomique. De plus, toute constante c est munie d'un type simple $\tau_c \in \mathcal{T}$.

Tout comme pour les lambda-termes, les parenthèses inutiles sont fréquemment omises dans les types simples, et l'opérateur $\to$ est associatif à droite : le type $\tau \to \tau \to \tau$ s'interprète comme $\tau \to (\tau \to \tau)$ et non comme $(\tau \to \tau) \to \tau$.

Le lambda-calcul simplement typé considère seulement des lambda-termes auxquels sont associés des types simples. Le constructeur $\to$ sur lequel sont basés ces derniers dénote l'abstraction fonctionnelle, en ce sens qu'une expression qui a pour type $\tau \to \sigma$ dénote une fonction dont l'argument est de type τ et dont le résultat de l'évaluation est de type σ. La notion de signature donnée ici répondra par la suite au même besoin que son pendant logique, où les constantes seront interprétées comme les éléments de base d'un modèle.

Typage à la Curry Nous introduisons maintenant un système de typage dit à la Curry, qui consiste à attribuer un type (par le biais d'un système d'inférence) à des lambda-termes. Celui-ci est basé sur les règles données par la figure 2.1.

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\, Var \qquad \frac{c \in \mathcal{C}}{\Gamma \vdash c : \tau_c}\, Cst$$

$$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \tau \to \sigma}\, Abs \qquad \frac{\Gamma \vdash M : \tau \to \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma}\, App$$

FIGURE 2.1 – Système de typage simple pour le lambda-calcul

Définition 2.9. Le *système de typage* décrit figure 2.1 produit et manipule des *jugements* de typage de la forme $\Gamma \vdash M : \tau$, où τ est un type simple et M un

lambda-terme. Le membre gauche Γ, appelé un *environnement* de typage, est une fonction partielle, de domaine fini, de $\mathcal{X}$ dans $\mathcal{T}$. De façon transparente, la notation $\Gamma, x : \tau$ représente la fonction qui associe le type τ à x, et $\Gamma(y)$ à toute variable $y \neq x$; cette notation requiert implicitement que $x \notin \mathrm{Dom}(\Gamma)$.

Une inférence construite à partir de ces règles est appelée une *dérivation* de typage, et le jugement apparaissant en conclusion associe un type simple à un lambda-terme. S'il existe une dérivation dont la conclusion est de la forme $\Gamma \vdash M : \tau$, le lambda-terme M est dit (simplement) *typable*, et de type τ.

L'ensemble des lambda-termes typables est un sous-ensemble strict de celui des lambda-termes. Ces termes possèdent certaines propriétés immédiates en raison de leur définition : en particulier, tout sous-terme d'un terme typable est également typable. Une autre conséquence relativement immédiate est que les notions de α, β et η-conversion ainsi que leurs propriétés sont préservées sur les lambda-termes simplement typés. En outre, ceux-ci possèdent deux propriétés supplémentaires présentant un fort intérêt pour la suite.

Théorème 2.3 (Réduction du sujet)**.** *Si M est un lambda-terme typable de type τ et que $M \to_{\beta\eta} N$, alors N est également typable et de type τ [cf. Barendregt, 1993, p. 57].*

Théorème 2.4 (Normalisation forte)**.** *Le système de réécriture engendré par $\overset{*}{\to}_{\beta\eta}$ sur les lambda-termes simplement typés est fortement normalisant, c'est-à-dire qu'il ne contient pas de réduction infinie. Autrement dit, pour tout terme typable M, il existe $K \in \mathbb{N}$ tel que si $M \overset{k}{\to}_{\beta\eta} N$, alors $k \leq K$ [cf. Barendregt, 1993, p. 61].*

Notons que la conjonction de la propriété de normalisation forte et de la propriété de Church-Rosser entraînent que tout terme typable M possède une unique forme $\beta\eta$-normale, notée $|M|_{\beta\eta}$.

Typage à la Church Alternativement, le lambda-calcul simplement typé peut être obtenu par une méthode dite à la Church, en définissant directement l'ensemble des lambda-termes munis de leurs types. Les lambda-termes typés à la Church peuvent également être vus comme une notation pour une dérivation de type à la Curry portant sur un lambda-terme.

Définition 2.10. L'ensemble $\mathcal{X}$ des variables utilisées est complété par des annotations de type, construisant l'ensemble des variables typées $\mathcal{X} \times \mathcal{T}$. Ces variables sont notées avec leurs types en exposant : (x, τ) est ainsi noté x^τ. Une signature $\mathcal{C}$ est définie comme précédemment.

L'ensemble des lambda-termes typés est alors défini comme suit :
— Si $x^\tau \in \mathcal{X} \times \mathcal{T}$, alors x^τ est un lambda-terme de type τ (variable).
— Si $c \in \mathcal{C}$, alors c est un lambda-terme de type τ_c (constante).

— Si M est un lambda-terme de type σ et $x^\tau \in \mathcal{X} \times \mathcal{T}$, alors $\lambda x^\tau.M$ est un lambda-terme de type $\tau \to \sigma$ (abstraction).
— Si M_1 et M_2 sont des lambda-termes de type $\tau \to \sigma$ et τ respectivement, alors $M_1 M_2$ est un lambda-terme de type σ (application).

Par abus de notation, les annotations de type sur les variables sont parfois omises lorsqu'il n'y a pas d'ambigüité ; ainsi, le terme $\lambda x^\tau.x^\tau$ peut être noté $\lambda x^\tau.x$, les variables typées construites à partir d'une même variable étant alors considérées comme ayant le même type. Dans certains cas, le type d'une variable peut être laissé totalement implicite, par exemple lorsqu'il peut être déduit à partir du contexte fourni par une application (ainsi, y a implicitement le type τ dans l'expression $(\lambda x^\tau.x)\ y$).

Le lambda-calcul simplement typé ainsi obtenu possède – dans l'ensemble – la même sémantique que celle produite par un typage à la Curry, et il jouit des mêmes propriétés précédemment mentionnées (normalisation forte, réduction du sujet). En outre, comme il ne contient que des termes typables, il possède également la propriété d'*expansion du sujet* (la β-expansion préserve le type d'un terme). Le choix d'un système à la Church simplifie la question du typage en garantissant que tous les termes possèdent, par construction, un type immédiatement apparent ; mais n'offre pas, contrairement aux dérivations à la Curry, la possibilité de parler explicitement du processus permettant d'attribuer un type à un terme.

2.2.4 Représentations par le lambda-calcul typé

Choisir une signature appropriée et fixer une interprétation d'une manière similaire à la théorie des modèles permet d'utiliser des lambda-termes pour représenter différents objets d'intérêt. En particulier, il est possible de représenter tous les types d'objets définis dans la section 2.1 (mots, termes, contextes et formules logiques), ou des tuples de tels objets, par des lambda-termes sur une signature spécifique.

Ainsi, le monoïde libre sur un alphabet Σ peut être représenté en considérant l'unique type atomique $*$, et en construisant une signature composée de l'ensemble des constantes $\{a^{*\to *} \mid a \in \Sigma\}$ dénotant les lettres de Σ. Afin de représenter les termes de Σ^*, la concaténation est dénotée par le lambda-terme $. = \lambda s_1^{*\to *}.\lambda s_2^{*\to *}.\lambda x^*.s_1\ (s_2\ x)$. L'interprétation du lambda-calcul garantit alors l'associativité de la concaténation : $.\ f\ (.\ g\ h) =_\beta .\ (.\ f\ g)\ h =_\beta \lambda x^*.f\ (g\ (h\ x))$; de même que l'existence d'un élément neutre qui est l'identité $\varepsilon = \lambda x^*.x$. Tous les lambda-termes dénotant des mots ont ainsi le type $* \to *$.

Au delà de cette correspondance immédiate, la sémantique opérationnelle du lambda-calcul typé permet également de représenter des opérations sur ces mots, et de composer et d'évaluer le résultat ces dernières tout en garantissant leur terminaison. Il est ainsi possible de représenter des projections $\pi_i = \lambda s_1^{*\to *} \dots s_n^{*\to *}.s_i$, ou des tuples $\lambda\pi.\pi\ s_1 \dots s_n$ de taille n, ce qui permet

par exemple de dénoter des chaînes « à trous » (ou contextes), de manière transparente.

De manière similaire, il est possible de décrire des alphabets gradués, termes et contextes. Par convention de notation, τ^n est défini inductivement comme $\tau^{n-1} \to \tau$, avec $\tau^1 = \tau$ (par abus de notation, l'expression $\tau^0 \to \sigma$ désigne usuellement le type σ). Le type $*$ désigne alors un terme clos et $*^n \to *$ est le type d'un contexte d'arité n ; tout symbole s d'un alphabet gradué, d'arité $\rho(s) = n$, est alors représenté par une constante $s^{*^n \to *}$. La construction de termes est dénotée immédiatement par l'application des lambda-termes qui les représentent $(s\ t_1 \dots t_n)$, et les contextes s'obtiennent de manière similaire, avec une correspondance immédiate entre leurs variables et celles de leurs lambda-termes.

Cette construction s'étend aux formules logiques, en choisissant deux types e et t dénotant respectivement les termes ou variables du premier ordre, et les valeurs de vérité. Les variables du second ordre (et prédicats unaires) ont alors le type $e \to t$ et, plus généralement, un prédicat P d'arités $\rho_1(P) = n$, $\rho_2(P) = m$ est représenté par une constante P du type $e^n \to (e \to t)^m \to t$. Les connecteurs usuels sont également dénotés par des constantes, telles que : $\neg^{t \to t}$, $\wedge^{t \to t \to t}$, $\vee^{t \to t \to t}$, $\exists^{(e \to t) \to t}$ ou $\exists^{2((e \to t) \to t) \to t}$. La quantification d'une variable dans une formule est représentée en liant par une abstraction la variable en question dans le lambda-terme, puis en appliquant le terme résultant à la constante représentant le quantificateur, de la manière suivante : la formule $\exists x.P(x)$ est dénotée par le terme $\exists \lambda x^e.P\ x$. Intuitivement, un quantificateur peut être envisagé comme une opération qui, en fonction d'une formule dont la valeur de vérité dépend d'une variable donnée, « trouve » une valeur à substituer à cette variable afin de satisfaire la formule. Une conséquence pratique est que les notions de variables libres et liées coïncident entre une formule logique et le lambda-terme qui la représente.

2.2.5 Applications

Les grammaires catégorielles abstraites, décrites dans la section 2.4.2, utilisent des représentations par le lambda-calcul typé telles que celles évoquées ci-dessus pour décrire des langages d'objets sous la forme de langages de lambda-termes dénotant ces objets. Par suite, les linéarisations que nous décrivons dans le chapitre 3 s'appuient sur un lambda-calcul simplement typé à la Church. La notation des grammaires commutatives du chapitre 5 se base également sur ce principe ; et le système de réécriture qui permet de minimiser les termes et contextes impliqués dans nos dérivations repose sur un système de typage sémantique à la Curry 5.6 disposant prouvablement de certaines des propriétés décrites plus haut.

2.3 Les langages réguliers de termes

Nous étudions maintenant la notion de langages de termes, et en particulier la classe des langages réguliers de termes. Celle-ci possède plusieurs caractérisations simples et de bonnes propriétés de clôture ; en outre, nous verrons que l'image de ses langages par certaines classes de transductions produit des classes d'objets reconnaissables et pertinentes pour la formalisation linguistique.

Nous présentons ci-dessous plusieurs caractérisations des langages réguliers de termes, en précisant la façon dont elles sont reliées. Nous listons également plusieurs des propriétés de clôture de ces langages, et détaillons en particulier la connexion entre logique et automates qui est centrale tout au long du chapitre 3. Enfin, nous préciserons l'intérêt particulier que représentent les langages de termes dans le reste de la thèse.

2.3.1 Automates de termes

La première caractérisation que nous considérons est celle donnée par les automates à états finis. Ceux-ci permettent de décider aisément de l'appartenance d'un terme à un langage régulier, et d'établir simplement les différentes propriétés de clôture des langages réguliers : complémentation, union et intersection, et image par morphisme linéaire et morphisme inverse.

Définition 2.11. Un *automate de termes* est un 4-uplet $\mathcal{A} = (\mathcal{S}, Q, Q_f, \Delta)$, où :

— $\mathcal{S}$ est un alphabet gradué.
— Q est un ensemble fini d'*états*.
— $Q_f \subseteq Q$ est un ensemble d'*états finaux*.
— Δ est une *relation de transition* définie comme suit.

Une relation de transition est décrite par un ensemble (fini) de *transitions*, qui sont des entrées de la forme $s\,(q_1 \dots q_n) \to q_0$, où $s \in \mathcal{S}$, $\rho(s) = n$, et $q_i \in Q$ pour tout i. L'automate $\mathcal{A}$ est dit *déterministe* si Δ est une fonction, et *complet* si cette fonction est totale.

Un automate $\mathcal{A}$ reconnaît un terme sur un alphabet $\mathcal{S}$ en assignant des états à ses feuilles puis à ses nœuds internes de bas en haut jusqu'à la racine, d'après sa fonction de transition Δ. Le langage régulier associé à $\mathcal{A}$ est formé de l'ensemble des termes tels que $\mathcal{A}$ assigne à la racine du terme un état qui est final.

Définition 2.12. Un *parcours* d'un automate $\mathcal{A} = (\mathcal{S}, Q, Q_f, \Delta)$ sur un terme t est une fonction $\rho : \mathrm{Dom}(t) \mapsto Q$ qui respecte la fonction de transition Δ, en ce sens que si $t|_p = s(t_1 \dots t_n)$ et que $\rho(pi) = q_i$ pour tout $i \in [n]$, alors $s\,(q_1 \dots q_n) \to \rho(p)$ appartient à Δ.

Un parcours est dit *acceptant* si et seulement si $\rho(\varepsilon) \in Q_f$. Un automate $\mathcal{A}$ *reconnaît* un terme t si et seulement si il existe un parcours acceptant de $\mathcal{A}$ sur t. Enfin, le langage $\mathcal{L}(\mathcal{A})$ d'un automate $\mathcal{A}$ est l'ensemble des termes reconnus par $\mathcal{A}$.

Nous supposons dans la suite que tous les automates que nous manipulons sont déterministes et complets ; il est en effet possible pour tout automate $\mathcal{A}$ de produire un automate $\mathcal{A}'$ déterministe et complet tel que $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$. Il est à noter que cette opération de déterminisation peut, dans le pire cas, augmenter exponentiellement le nombre d'états d'un automate.

La classe des langages dits réguliers, reconnus par les automates à états finis, possède plusieurs propriétés de clôture démontrables à partir de la définition des automates. Nous listons ici ces propriétés et une intuition de la construction de l'automate correspondant au langage résultant ; le détail et la preuve de ces constructions peuvent se trouver par exemple dans Comon *et al.* [2007]. Ces propriétés sont :

— Complémentation, en choisissant $Q'_f = Q \setminus Q_f$ comme nouvel ensemble d'états finaux sur un automate complet.
— Union, en construisant l'union disjointe des deux automates.
— Intersection, par conséquence directe des deux précédentes.
— Image par un morphisme linéaire, en appliquant le morphisme aux symboles dans les transitions de l'automate et en ajoutant là où c'est nécessaire des transitions pour reconnaître localement les contextes obtenus.

2.3.2 Grammaires de termes

Nous introduisons maintenant les grammaires régulières de termes, qui peuvent être vues comme des systèmes formels de réécriture générant des termes. Elles définissent des langages de termes constructibles (dérivables) selon les règles dont elles sont munies. Nous verrons que la classe de langages ainsi obtenue coïncide avec celle décrite par les automates à états finis.

Définition 2.13. Une *grammaire régulière de termes* G est un tuple $G = (\mathcal{S}, N, S, P)$, où :

— $\mathcal{S}$ est un alphabet gradué.
— N est un ensemble de *symboles non-terminaux*.
— $S \in N$ est un symbole non-terminal de départ, nommé *axiome*.
— P est un ensemble fini de *productions*.

Une production $p \in P$ est une paire $A \to t$ dont le membre gauche $A \in N$ est un symbole non-terminal, et le membre droit t est un terme construit sur l'alphabet $\mathcal{S} \cup N$, où les éléments de N sont d'arité nulle. Une production dont le membre droit ne contient pas de symboles non-terminaux est dite *terminale*.

Toute grammaire peut générer des termes par l'application répétée de ses productions : dans un terme construit sur $\mathcal{S} \cup N$ tout non-terminal A appa-

raissant à gauche d'une production de G peut être réécrit en le sous-terme apparaissant à droite de cette même production. En considérant pour terme initial l'axiome S de G, et par l'application de productions terminales (qui remplacent un symbole non-terminal par un terme sur $\mathcal{S}$), une grammaire permet de générer un ensemble de termes construits uniquement sur $\mathcal{S}$: cet ensemble constitue le langage de la grammaire.

Définition 2.14. Toute grammaire $G = (\mathcal{S}, N, S, P)$ induit une *relation de dérivation* $\rightarrow_G$ définie comme suit. Deux termes termes t et t' construits sur $\mathcal{S} \cup N$ sont reliés par $t \rightarrow_G t'$ si et seulement si il existe un contexte linéaire C tel que $t = C[A]$ et $t' = C[u]$, et que P contient la production $A \rightarrow u$.

Le langage $\mathcal{L}(A)$ d'un non-terminal A est constitué de l'ensemble des termes t tels que $A \overset{*}{\rightarrow}_G t$ et qu'aucun symbole non-terminal de N n'apparaît dans t. Le langage $\mathcal{L}(G)$ *généré* par la grammaire G est défini comme étant celui de son axiome, $\mathcal{L}(S)$.

Additionnellement, une grammaire telle que chacune de ses productions contient exactement un symbole de $\mathcal{S}$ est dite en *forme normale*. L'emploi de grammaires en forme normale simplifie certaines constructions et preuves, et n'entraîne pas de perte de généralité : il est toujours possible de décomposer une production dont le membre droit contient un terme complexe et d'éliminer les productions dont le membre droit se réduit à un symbole non-terminal, sans modifier le langage généré par la grammaire.

Correspondance grammaires/automates Nous établissons maintenant la correspondance entre grammaires et automates, qui est directe. Nous nous intéressons seulement à la traduction des grammaires vers les automates, afin de montrer que les langages générés par ces grammaires sont réguliers, la réciproque est toutefois vraie [*cf.* Comon *et al.*, 2007, p. 54]. Pour toute grammaire régulière de termes $G = (\mathcal{S}, N, S, P)$ en forme normale, il est possible de construire un automate de termes $\mathcal{A} = (\mathcal{S}, Q, Q_f, \Delta)$ tel que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(G)$ de la manière suivante :

- $Q = N$
- $Q_f = \{S\}$
- Si $A \rightarrow s(B_1 \dots B_n)$ est une production de P (où n peut être zéro), alors $s\,(B_1 \dots B_n) \rightarrow A$ est une transition de Δ, et Δ ne contient pas d'autre transition.

2.3.3 Définissabilité en logique

Comme mentionné dans la section 2.1.5, il est possible d'utiliser des formules logiques pour définir des langages par la théorie des modèles. Le langage d'une formule (généralement close) est alors l'ensemble des objets du domaine

qui satisfont cette formule, en suivant l'interprétation des prédicats appartenant à la signature sur laquelle la formule est construite. C'est précisément l'approche choisie par MTS, décrite dans la section 1.1.

Nous considérons maintenant des signatures dites à k successeurs, associées à un alphabet gradué $\mathcal{S}$ d'arité k, sur la logique monadique du second ordre : les langages des formules qui en résultent coïncident exactement avec la classe des langages réguliers de termes sur $\mathcal{S}$.

Définition 2.15. Étant donné un alphabet gradué $\mathcal{S}$ dont l'arité maximale est k, nous considérons la signature SkS formée des prédicats suivants, qui portent uniquement sur des variables du premier ordre :

— Si $s \in \mathcal{S}$, alors $s(x)$ est un prédicat d'arité 1 dans SkS.

— Pour tout $i \in [k]$, $S_i(x, y)$ est une relation d'arité 2 dans SkS.

La sémantique d'une logique construite sur SkS est obtenue en prenant pour classe de modèles l'ensemble des termes t construits sur $\mathcal{S}$, et pour domaine l'ensemble $\mathrm{Dom}(t)$ des positions valides dans t. Étant donné une valuation ν de $\mathcal{X}$ dans $\mathrm{Dom}(t)$, l'interprétation des prédicats de SkS est :

— $t, \nu \models s(x)$ ssi le sous-terme $t|_{\nu(x)}$ est construit par le symbole s.

— $t, \nu \models S_i(x, y)$ ssi $\nu(y) = \nu(x).i$.

La logique monadique du second ordre construite sur la signature SkS est appelée *logique monadique du second ordre à k successeurs* (abrégé en MSOkS).

La notion de langage d'une formule est définie exactement comme dans la section 2.1.5. Nous détaillons maintenant un résultat essentiel dû à Thatcher et Wright [1968], qui montre que les langages définissables par MSOkS coïncident exactement avec les langages réguliers de termes.

2.3.4 Correspondance logique/automates

Théorème 2.5. *Pour toute formule close ϕ de MSOkS, il existe un automate de termes $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\phi)$ [cf. Comon et al., 2007, p. 95].*

Une construction classiquement associée à cette preuve consiste à restreindre la syntaxe de MSOkS afin de transformer la formule ϕ en une formule équivalente ϕ' utilisant uniquement des variables du second ordre et un petit nombre de constructeurs, puis à construire l'automate $\mathcal{A}$ inductivement sur la structure syntaxique de ϕ'. Cette construction s'appuie essentiellement sur les propriétés de clôture des automates.

Syntaxe restreinte

Afin de construire la formule ϕ' à partir de ϕ, on s'appuie sur une syntaxe restreinte utilisant uniquement des variables du second ordre.

Afin d'obtenir des formules équivalentes à celles utilisant des variables du premier ordre, un nouveau prédicat $\mathrm{Sgl}(X)$ est introduit, qui dénote qu'une variable du second ordre est un singleton. Ce prédicat est défini comme suit :

$$\mathrm{Sgl}(X) \overset{def}{\Leftrightarrow} \neg(\forall^2 Z.X \subseteq Z) \wedge \forall^2 Y.Y \subseteq X \Rightarrow (X \subseteq Y \vee \forall^2 Z.Y \subseteq Z)$$

Cette définition garantit que l'ensemble dénoté par X est un singleton : X n'est pas vide, et tout sous-ensemble Y de X est soit X lui-même, soit l'ensemble vide.

Nous considérons maintenant la signature SkS, que nous altérons pour que ses prédicats portent sur des variables du second ordre. L'interprétation de la signature résultante est la suivante :

— $t, \nu \models s(X)$ ssi $t|_p = s(t_1 \dots t_n)$ pour tout p de $\nu(X)$.

— $t, \nu \models S_k(X, Y)$ ssi $p' = pk$ pour tout p de $\nu(X)$ et tout p' de $\nu(Y)$.

Observons que l'interprétation de la signature résultante coïncide avec celle de SkS sous l'hypothèse que les variables X et Y sont des singletons (contenant exactement les valeurs des variables du premier ordre x et y dans l'interprétation usuelle).

Nous pouvons maintenant, en nous appuyant sur la signature altérée et le prédicat $\mathrm{Sgl}(X)$, transformer inductivement toute formule close ϕ de MSOkS en une formule ϕ' équivalente en utilisant les règles suivantes :

— Si $\phi = s(x)$, alors $\phi' = s(X)$.

— Si $\phi = S_i(x, y)$, alors $\phi' = S_i(X, Y)$.

— Si $\phi = (x = y)$, alors $\phi' = (X = Y)$.

— Si $\phi = x \in Y$, alors $\phi' = X \subseteq Y$.

— Si $\phi = \neg\psi$, alors $\phi' = \neg\psi'$.

— Si $\phi = \psi_1 \wedge \psi_2$, alors $\phi' = \psi'_1 \wedge \psi'_2$.

— Si $\phi = \exists x.\psi$, alors $\phi' = \exists^2 X.\, \mathrm{Sgl}(X) \wedge \psi'$.

— Si $\phi = \exists^2 X.\psi$, alors $\phi' = \exists^2 X.\psi'$.

Intuitivement, cette transformation substitue à toute variable du premier ordre x une variable du second ordre X qui lui correspond, et garantit que cette variable dénote un singleton lorsqu'elle est quantifiée (par hypothèse, ϕ est une formule close). L'interprétation de ϕ est ainsi préservée, et les formules résultantes s'appuient uniquement sur des variables du second ordre, quantifications existentielles, conjonctions, négations et formules atomiques.

Construction inductive de $\mathcal{A}$

Nous construisons maintenant, pour toute formule ϕ utilisant la syntaxe restreinte, un automate $\mathcal{A}$ qui reconnaît un langage de termes associé à ϕ. Les termes sur lesquels travaille $\mathcal{A}$ représentent simultanément un modèle et une valuation, et ceux appartenant à $\mathcal{L}(\mathcal{A})$ sont exactement ceux correspondant à des paires modèle/valuation qui satisfont la formule ϕ.

Le résultat général est obtenu par induction sur une formule de départ close, en montrant initialement qu'à toute formule atomique correspond un automate. Il est ensuite possible d'associer inductivement à toute formule composite un automate construit en s'appuyant sur les propriétés de clôture données plus haut (complémentation, intersection, morphisme linéaire). Finalement, il suffit d'effacer des termes du langage résultant toute l'information portant sur la valuation qui satisfait la formule (sans conséquence puisque la formule d'origine est close).

Nous décrivons d'abord l'alphabet gradué $\mathcal{S}'$, sur lequel s'appuient les termes qui représentent une paire modèle/valuation. Soit $\mathcal{S}$ l'alphabet gradué sur lequel est basé SkS, et $\mathcal{X}_\phi \subset \mathcal{X}_2$ l'ensemble des variables susceptibles d'apparaître dans la formule ϕ. Remarquons que cet ensemble est fini (puisque la formule dont on souhaite construire le langage de termes est finie) et ne contient que des variables du second ordre ; on note son cardinal $\#(\mathcal{X}_\phi) = n$. L'alphabet $\mathcal{S}'$ est obtenu par le produit de $\mathcal{S}$ avec un tuple de booléens de taille n : $\mathcal{S}' = \mathbb{B}^n \times \mathcal{S}$; les arités de ses symboles sont données par leur composante droite : $\rho(b_1, \ldots, b_n, s) = \rho(s)$. Ainsi, la projection droite des symboles de $\mathcal{S}'$ dans un terme dénote un modèle $\mathcal{M}$ possible pour ϕ, et les autres composantes dénotent une valuation ν pour ses variables libres $X_1 \ldots X_n$ de la manière suivante : si le symbole d'un sous-terme à une position p a pour i^{e} composante *Vrai*, alors $p \in \nu(X_i)$. Cette interprétation est exemplifiée par la figure 2.2, où les valeurs booléennes *Faux* et *Vrai* sont représentées par 0 et 1.

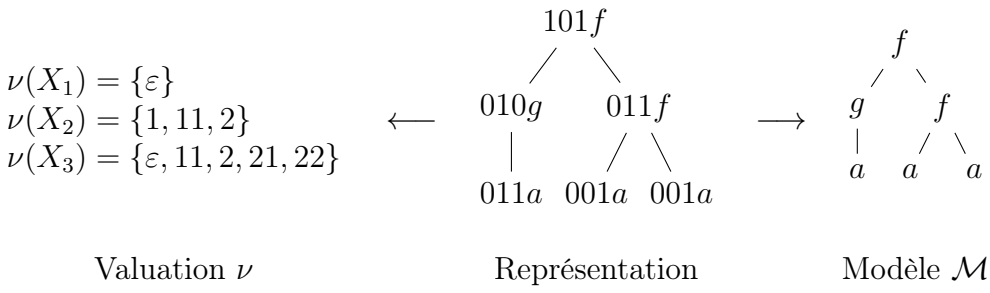

FIGURE 2.2 – Représentation d'un modèle et d'une valuation par un terme

Sans détailler leur construction explicite, nous donnons maintenant une indication des raisons pour lesquelles il existe un automate reconnaissant le langage L_ψ des termes qui dénotent un modèle et une valuation satisfaisant toute sous-formule atomique ψ de ϕ. Rappelons que la notation $\pi_i(s)$ dénote la i^{e} composante de s (c'est-à-dire dans ce cas l'appartenance de la position où est situé le symbole s à l'ensemble $\nu(X_i)$ dénoté par X_i) :

— Si $\psi = s(X_i)$, alors tout symbole s' présent dans un terme de L_ψ doit être tel que si $\pi_i(s') = \mathit{Vrai}$, alors leur dernière composante est s (toute position de $\nu(X_i)$ porte le symbole s). Cette propriété locale est aisément vérifiable par un automate simple.

— Si $\psi = S_l(X_i, X_j)$, considérons toute paire de positions (p, p') dans un terme de L_ψ dont les symboles sont (s_1, s_2) tels que $\pi_i(s_1)$ et $\pi_j(s_2)$ valent toutes les deux *Vrai* : alors cette paire de positions doit satisfaire $p' = p.l$ (p' est le l^{e} successeur de p). Cette propriété est également locale et peut être vérifiée par un automate de taille réduite, dont les états encodent la valeur de la j^{e} composante du dernier symbole parcouru.
— Si $\psi = (X_i = X_j)$, alors tout symbole s dans un terme de L_ψ doit vérifier $\pi_i(s) \Leftrightarrow \pi_j(s)$ (les valeurs des ensembles X_i et X_j coïncident).
— Si $\psi = X_i \subseteq X_j$, alors tout symbole s d'un terme de L_ψ doit vérifier $\pi_i(s) \Rightarrow \pi_j(s)$ (toute position de $\nu(X_i)$ appartient également à $\nu(X_j)$).

Nous traitons ensuite le cas des formules composites. Toute formule composite ψ de ϕ utilisant la syntaxe restreinte est soit une négation, soit une conjonction, soit une quantification existentielle du second ordre. Pour chacun de ces cas, le langage L_ψ des termes dénotant un modèle et une valuation satisfaisant ψ peut être décrit par un automate obtenu inductivement à partir des automates associées aux sous-formules immédiates de ψ pour les raisons suivantes :

— Si $\psi = \neg\psi'$, alors L_ψ est obtenu par complémentation de $L_{\psi'}$.
— Si $\psi = \psi_1 \wedge \psi_2$, alors L_ψ est l'intersection des langages L_{ψ_1} et L_{ψ_2}.
— Si $\psi = \exists^2 X_i.\psi'$, alors le langage L_ψ peut être obtenu en « oubliant » toute information sur l'ensemble X_i dans la valuation ν. Considérons le réétiquetage $h_i : (b_1 \dots b_{i-1}, b_i, b_{i+1} \dots b_n, s) \mapsto (b_1 \dots b_{i-1}, b_{i+1} \dots b_n, s)$: son application à un terme $t \in L_{\psi'}$ produit le terme $t' = h_i(t)$, dénotant la valuation attendue pour ψ (en supposant, sans perte de généralité, que la variable X_i n'apparaît dans aucune sous-formule de ϕ en dehors de ψ). Puisque les langages définis par les automates sont clos par morphisme linéaire (tel qu'un réétiquetage), le langage résultant est également régulier.

L'automate résultant reconnaît le langage L_ϕ des termes dénotant un modèle et une valuation satisfaisant ϕ ; puisque cette formule est close, la valuation ν dénotée par les n premières composantes est vide, ne laissant que l'information sur le modèle. L'automate $\mathcal{A}$ résultant reconnaît donc exactement l'ensemble des modèles qui, couplés à la valuation vide, satisfont la formule ϕ, c'est-à-dire que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\phi)$.

2.3.5 Application

Notre intérêt pour les langages de termes (et en particulier la classe des langages réguliers) est double. D'une part, les termes permettent d'encoder de manière transparente des arbres d'arité bornée, lesquels sont couramment utilisés pour représenter diverses structures du langage (structures syntaxiques, structures de traits, relations de dépendances, ...). D'autre part, des termes construits en tant qu'éléments d'une algèbre fixée peuvent être utilisés pour

représenter d'autres types d'objets (mots, graphes, ...), permettant de manipuler par des moyens algorithmiques simples toute structure représentable algébriquement.

La caractérisation donnée par les automates présentés initialement constitue pour nous une procédure de décision simple pour les langages réguliers, ainsi qu'un moyen d'établir simplement leurs propriétés de clôture. Ceci nous permet dans la suite de ramener des présentations alternatives, choisies pour leurs capacités descriptives, au cas des automates.

En revanche, le choix d'utiliser des grammaires de réécriture pour caractériser des langages réguliers de termes sera motivé dans le chapitre 3 par plusieurs avantages. Premièrement, les symboles non-terminaux et productions qu'elle contient donnent une structure visible au langage qu'elle génère : une production $p = A \to t(B_1 \dots B_n)$ s'interprète intuitivement comme la possibilité de construire un élément de type A par le biais d'un terme de la forme de t, où chaque B_i est remplacé par un sous-terme ayant le type B_i. Il est possible de donner une interprétation similaire aux transitions d'un automate, mais ceux-ci s'interprètent plus naturellement comme des outils de reconnaissance que comme des règles de génération.

Un autre intérêt, plus particulier à cette thèse, à l'emploi de grammaires régulières de termes, est leur similarité avec de nombreux mécanismes utilisés dans le cadre du traitement automatique des langues et de la formalisation linguistique, tels que les grammaires hors contexte de mots, les c-structures des grammaires lexicales-fonctionnelles ou les substitutions dans les grammaires d'arbres adjoints. Ces parallèles peuvent permettre à des usagers de ces formalismes n'ayant pas de notions approfondies en théorie des langages formels d'appréhender plus aisément les langages définis par des grammaires de réécritures.

Enfin, l'emploi de la logique formelle pour définir un langage de termes au moyen d'une formule sur MSOkS permet de décrire avec une grande concision, et indépendamment les unes des autres, les différentes propriétés des termes qui appartiennent à un langage – ou de ceux qui en sont spécifiquement exclus. Cette concision offerte par la logique, qui permet à des formules de taille réduite de décrire de très grands automates, peut être vue comme un obstacle algorithmique ; toutefois elle constitue également une source d'expressivité, qui nous permet de raccourcir et de simplifier grandement les descriptions linguistiques tout au long des chapitres 3 et 4.

De plus, la classe des langages réguliers de termes est incluse dans la classe des langages définissables par des grammaires de remplacement d'hyper-arêtes (HR), présentées dans Courcelle et Engelfriet [2011]. Cette dernière est close par une certaine classe de transductions dites MSO-définissables ; et les langages d'arbres qu'elle produit coïncident avec ceux définis par certaines classes de grammaires catégorielles abstraites (voir section 2.4.2). Cet état de fait nous permet d'établir quelques uns des résultats du chapitre 3, et constitue l'une

des motivations du travail de cette thèse.

Pour terminer, les grammaires commutatives présentées dans le chapitre 5 définissent des langages de termes dont l'interprétation produit indirectement les langages de mots étudiés. Les classes de langages de termes qui leurs sont associées incluent (mais ne se limitent pas à) la classe des langages réguliers.

2.4 Classes de langages et caractérisations

La linguistique informatique intègre dans ses hypothèses de travail le choix d'une classe de langages, conséquence des choix de conception du formalisme utilisé, qui constitue une approximation de la classe des langages naturels (ou du moins d'un sous-ensemble caractéristique de ceux-ci). Un exemple notable est la classe des langages légèrement sensibles au contexte, abordée dans la section 1.2.

À l'opposé des formalismes conçus selon des approches linguistiquement informées, d'autres caractérisations mathématiquement plus simples, issues de la théorie des langages, capturent des classes de langages similaires aux langages légèrement sensibles au contexte. Ces descriptions constituent à ce titre des candidats au titre d'outils « bas niveau » (à l'image des langages de programmation proches de la machine) pour la description de modèles linguistiques, et permettent d'étudier plus aisément les propriétés algorithmiques de ces classes de langages.

2.4.1 Grammaires hors-contexte multiples

Les grammaires hors-contexte multiples constituent une classe de grammaires décrivant des langages légèrement sensibles au contexte. Il existe plusieurs présentations équivalentes de ces grammaires[Seki *et al.*, 1991][Weir, 1988] ; nous adoptons ici celle donnée par Kanazawa [2010], où les productions peuvent s'interpréter directement d'une manière similaire aux clauses de Horn en programmation logique.

Définition 2.16. Une *grammaire hors-contexte multiple* (MCFG) G est définie comme un tuple $G = (\Sigma, N, S, P)$, dans lequel :

- Σ est un alphabet.
- N est un ensemble de symboles non-terminaux gradués.
- S est un élément de N, d'arité $\rho(S) = 1$ (l'axiome).
- P est un ensemble fini de productions de la forme :

$$A(u_1, \ldots, u_n) \leftarrow B_1(x_{1,1} \ldots x_{1,n_1}) \ldots B_p(x_{p,1} \ldots x_{p,n_p})$$

où, pour tout $k \in [n]$, u_k est un mot formé sur l'alphabet $\Sigma \cup \mathcal{X}$, l'ensemble $\mathcal{X} = \{x_{i,j} \mid i \in [p] \wedge j \in [n_i]\}$ dénotant l'ensemble des variables du membre droit.

Les productions de P doivent en outre respecter les conditions suivantes :
— Chaque variable de $\mathcal{X}$ apparaît au plus une fois dans le membre gauche.
— Pour tout $i \in [p]$, le symbole B_i est d'arité n_i, et A est d'arité n.

Une production dont le membre droit est vide est dite *terminale*. Une production dont le membre gauche contient toutes les variables de $\mathcal{X}$ est dite *non-effaçante*. Une MCFG est non-effaçante si toutes ses productions sont non-effaçantes.

Enfin, une m-MCFG est une MCFG dont tous les non-terminaux sont d'arité au plus m.

Une MCFG associe à chacun de ses non-terminaux un langage qui est un ensemble de tuples de mots. Ces tuples peuvent être dérivés successivement en interprétant intuitivement les productions comme des clauses de Horn, exprimant que le tuple $(u_1 \dots u_n)$ appartient au langage de A sous l'hypothèse que chaque $(x_{i,1} \dots x_{i,n_i})$ appartient au langage de B_i. Le langage de la grammaire s'obtient directement à partir de celui de son axiome.

Définition 2.17. Le langage $\mathcal{L}(A)$ associé par une grammaire G à un non-terminal A d'arité $\rho(A) = n$ est l'ensemble des tuples $(w_1 \dots w_n)$ respectant les conditions suivantes :
— $A(u_1, \dots, u_n) \leftarrow B_1(x_{1,1} \dots x_{1,n_1}) \dots B_p(x_{p,1} \dots x_{p,n_p})$ appartient à P.
— Pour tout $i \in [p]$, $(w_{i,1} \dots w_{i,n_i})$ appartient à $\mathcal{L}(B_i)$.
— Pour tout $k \in [n]$, $i \in [p]$ et $j \in [n_i]$, $w_k = u_k[x_{i,j} \leftarrow w_{i,j}]$.

Le langage $\mathcal{L}(G)$ d'une grammaire G est l'ensemble des mots contenus dans le langage de son axiome, $\mathcal{L}(S)$.

La caractérisation offerte par les MCFG permet de retrouver aisément d'autres classes de grammaires bien connues. Ainsi, les 1-MCFG équivalent aux grammaires hors-contexte (CFG) ; et restreindre les productions de P à la forme $A(w.x) \leftarrow B(x)$ (avec $w \in \Sigma^*$) produit la classe des grammaires régulières (RG). Une autre contrainte simple sur les productions des MCFG permet d'obtenir des grammaires dites sans entrelacement (*well-nested*, abrégées en MCFG_{WN}). Par suite, la classe de langages décrite par les 2-MCFG_{WN} coïncide avec les langages légèrement sensibles au contexte originellement envisagés par Joshi [1985], et les MCFG en général génèrent la même classe MCS (plus large) que les grammaires minimalistes et les les grammaires d'arbre adjoints multiples (*Multi-Component Tree Adjoining Grammars*).

Une dernière remarque sur le fonctionnement des MCFG est que, comme la plupart des grammaires génératives et des grammaires d'arbres, elles décrivent un ensemble d'arbres de dérivations formant un langage régulier (et même local) : ainsi, les choix de réécriture effectués pour un non-terminal donné n'affectent pas la réécriture des autres non-terminaux. Le tuple de mots généré par une dérivation peut par la suite être reconstruit en composant uniformément, de manière ascendante, les tuples associés aux sous-dérivations.

2.4.2 Grammaires catégorielles abstraites

Les grammaires catégorielles abstraites, proposées par de Groote [2001], offrent un cadre permettant de décrire de manière uniforme des langages d'objets représentables par le lambda-calcul simplement typé. Leur fonctionnement repose sur l'association entre un langage (dit abstrait) de structures grammaticales, et le langage (dit concret) des objets représentés par ces structures. Une grammaire catégorielle abstraite est décrite par son langage abstrait et par l'application (nommée lexique) associant les termes de ce langage à des termes de son langage concret.

Définition 2.18. Une grammaire catégorielle abstraite (ACG) G est définie comme un tuple $G = (\Sigma_a, \Sigma_c, \Lambda, s)$, où :

— Σ_a et Σ_c sont des signatures du lambda-calcul simplement typé (*cf.* définition 2.8), appelées respectivement le *vocabulaire abstrait* et le *vocabulaire concret* de G.

— Le *lexique* Λ de G est une fonction qui associe aux constantes de Σ_a des lambda-termes linéaires construits sur Σ_c. Cette fonction doit offrir un typage cohérent, en ce sens que tout type atomique de Σ_a doit correspondre à un unique type simple de Σ_c par Λ.

— Le *type distingué* s de G est un type simple de Σ_a.

L'ensemble des lambda-termes sur Σ_a ayant le type s est appelé le *langage abstrait* de G. L'image des termes du langage abstrait par le morphisme défini par Λ constitue le *langage concret* (ou *langage objet*).

Si les termes du langage concret d'une grammaire G peuvent être interprétés comme des objets mathématiques (mots, termes, graphes, ...), l'ensemble des objets ainsi dénotés peut être considéré de façon transparente comme le langage de G. À l'inverse, le langage abstrait peut être considéré, à l'image des arbres de dérivation d'une grammaire générative, comme l'ensemble des structures grammaticales « internes » de la grammaire G.

La classe des langages dénotés par l'ensemble des grammaires catégorielles abstraites est extrêmement riche. Pour cette raison, une restriction usuelle est de se limiter à un vocabulaire concret permettant de dénoter uniquement des mots sur un alphabet ou des termes d'une algèbre spécifique, en fixant une interprétation unique pour toutes les grammaires de ce type. Par suite, il est possible de décrire une hiérarchie sur les ACG résultantes, en restreignant la complexité du langage abstrait et du lexique qui le relie au langage concret.

Définition 2.19. L'*ordre* $\mathrm{ord}(\tau)$ d'un type simple τ est défini inductivement comme :

— $\mathrm{ord}(\tau) = 1$ si τ est un type atomique.

— $\mathrm{ord}(\sigma \to \tau) = \max(\mathrm{ord}(\sigma) + 1, \mathrm{ord}(\tau))$ autrement.

Par suite, l'*ordre* $\mathrm{ord}(\Sigma)$ d'un vocabulaire Σ est défini comme l'ordre maximal des types des constantes de Σ ; et la *complexité* $\mathrm{comp}(\mathscr{L})$ d'un lexique

$\mathscr{L}$ est définie comme l'ordre maximal des images des types atomiques de Σ_a (cette valeur étant fixée en raison de la cohérence du typage induit par $\mathscr{L}$).

La notation $\mathbf{ACG}(n, m)$ désigne l'ensemble des ACG dont le vocabulaire abstrait est d'ordre n et dont le lexique est de complexité m. Par extension, une grammaire $G \in \mathbf{ACG}(n, m)$ est dite d'ordre n et de complexité m.

Cette notation induit une double hiérarchie sur l'ensemble des grammaires catégorielles abstraites. Ainsi, la classe de langages induite par les ACG du premier ordre correspond aux langages finis. Les grammaires du second ordre induisent, en considérant une représentation canonique des mots par des lambda-termes telle que celle donnée dans la section 2.2.4, des classes de langages connues, selon la complexité de leur lexique :

Complexité	1	2	3	4+
Classe équivalente	RG	CFG	MCFG_{WN}	MCFG

Des résultats similaires existent pour les langages d'arbres représentables par des ACGKanazawa [2009a] (allant des langages réguliers pour $m = 1$, aux langages représentables par des grammaires de remplacement d'hyper-arêtes pour $m \geq 4$). Un autre résultat remarquable apparaît en considérant la classe des langages de mots induite à partir d'une classe de langages d'arbres par l'opération de concaténation de gauche à droite des feuilles des arbres (usuellement nommée *yield*) : l'image par cette opération de la classe des langages d'arbres induite par les $\mathbf{ACG}(2, m)$ est la classe des langages de mots induite par les $\mathbf{ACG}(2, m + 1)$.

2.4.3 Applications

La hiérarchie des MCFG est présentée ici non seulement pour des raisons illustratives, mais surtout parce qu'elle sert de support à la hiérarchie des grammaires commutatives construite et étudiée dans le chapitre 5, qui propose une manière d'étendre la notion de sensibilité légère au contexte aux langues non-configurationnelles. L'encodage permettant d'obtenir la correspondance entre les ACG du second ordre et les MCFG, dû à de Groote et Pogodalla [2004], est essentiellement identique à celui qui est donné dans la section 5.3.2.

Enfin, le formalisme de description linguistique décrit dans le prochain chapitre s'inspire directement des ACG dans sa structure générale ; construisant un langage abstrait de structures grammaticales, il permet ensuite de décrire des linéarisations de ces structures vers plusieurs langages concrets, dont le rôle est similaire à celui du lexique dans une ACG. Bien que ces linéarisations (et le langage abstrait qu'elles transforment) soient essentiellement décrits de manière implicite par la logique, nous mettrons en correspondance les langages de lambda-termes qui en résultent avec ceux générés par les ACG du second ordre.