Ordres libres

La méthodologie de formalisation linguistique adoptée dans le chapitre précédent permet de produire aisément des linéarisations vers la forme canonique des phrases dans les langues où l'ordre des mots est fixé. Toutefois, elle échoue à représenter de manière concise l'ensemble des réalisations de surface possibles dans des langues offrant un plus grand degré de liberté sur l'ordre des syntagmes. Il est donc nécessaire, pour linéariser nos structures abstraites vers leur forme phonologique, de proposer des réalisations multiples, couvrant l'ensemble des combinaisons possibles. Afin de condenser ces descriptions, nous proposons dans ce chapitre une représentation intermédiaire, qui décrit des phrases modulo permutation de certains de leurs composants. Par suite, nous proposons plusieurs classes de langages s'appuyant sur cette représentation, et étudions la complexité algorithmique de plusieurs problèmes liés à l'analyse d'un énoncé selon ces représentations.

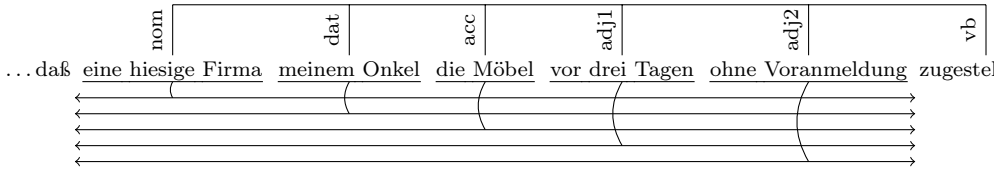
5.1 Motivation

La classe de phénomènes linguistiques que nous souhaitons modéliser, désignés par la suite sous le terme d'*ordres libres*, recouvre plusieurs notions. La plus évidente est la reconnaissance de phrases dans des langues à déclinaisons, où un marquage morphologique des cas (nominatif, accusatif, datif, *etc.*) est systématiquement présent et permet de retrouver la fonction syntaxique des arguments, indépendamment de leur position relative dans la phrase ; de telles langues, à l'image du latin, peuvent disposer d'un ordre canonique des arguments, mais autorisent fréquemment de larges déviations de cet ordre, sans affecter la grammaticalité ou le sens fondamental de la phrase [*cf.* Marouzeau, 1922].

Toutefois, le concept d'ordres libres peut également recouvrir des phénomènes plus restreints dans des langues imposant normalement des contraintes sur l'ordre de leurs arguments. Ainsi, les propositions subordonnées imbriquées en allemand sont usuellement construites de façon « parenthésée », en plaçant

les arguments nominaux du verbe au début, le verbe à la fin, et la proposition subordonnée entre les deux. Cependant, certains exemples attestent qu'il est possible, en préservant l'ordre des verbes à la fin, de mélanger librement leurs arguments, sans respecter le parenthésage induit par la structure abstraite. La reconstruction exacte de la structure syntaxique de ces énoncés (notamment les relations entre les verbes et leurs arguments) s'appuie alors usuellement sur des considérations sémantiques et pragmatiques.

Un premier exemple issu de Haider [1991] et repris dans Becker *et al.* [1992] est illustré par la figure 5.1. Dans cet exemple, les différents arguments de la proposition subordonnée forment des blocs, dont l'ordre relatif est quelconque : seule la position du verbe (rejeté à la fin) est constante.

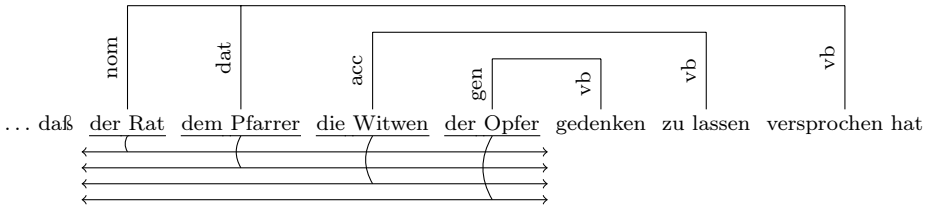


Exemple :

. . . daß vor drei Tagen meinem Onkel eine hiesige Firma ohne Voranmeldung die Möbel zugestellt hat.
. . . que il y a trois jours à mon oncle une entreprise locale sans préavis le mobilier a livré.
. . . qu'une entreprise locale a livré le mobilier à mon oncle sans préavis il y a trois jours.

FIGURE 5.1 – Liberté dans l'ordre des mots en allemand

Un second exemple, tiré de Becker *et al.* [1991] et donné dans la figure 5.2, illustre la possibilité de mélanger les arguments issus de plusieurs propositions subordonnées imbriquées. L'exemple comporte trois propositions subordonnées imbriquées, qui sont construites dans l'ordre canonique en haut de la figure : le placement de leurs arguments est toutefois libre, et indépendant de l'ordre (fixé) des verbes figurant à la fin, lequel permet de déduire la structure de la phrase. Observons que cette liberté permet de mélanger entre eux des composants appartenant à plusieurs propositions distinctes.



Exemple :

. . . daß die Witwen der Opfer dem Pfarrer der Rat gedengken zu lassen versprochen hat.
. . . que les veuves les victimes au prêtre le conseil commémorer de laisser a promis.
. . . que le conseil a promis au prêtre de laisser les veuves commémorer les victimes.

FIGURE 5.2 – Mélange des arguments en allemand (*scrambling*)

Notre objectif dans la suite de ce chapitre est de proposer une modélisation des phénomènes d'ordre libre, qui décrive de manière complètement

uniforme l'ensemble des ordres des mots autorisés par une langue ou par un phénomène donné. Cette modélisation exclut donc la possibilité de traiter des nuances de sens et d'emphase véhiculées par le choix d'un ordre en particulier de préférence à un autre. Tous les phénomènes offrant un degré modéré de liberté dans l'ordre des mots ne seront pas descriptibles par l'outil que nous proposons : ainsi, nous ne pourrons pas proposer de représentation unique pour toutes les réalisations d'une proposition indépendante en allemand, où le verbe doit obligatoirement occuper la seconde position, mais où le placement de ses arguments est libre. Choisi pour ses propriétés algorithmiques, cet outil permet seulement de rendre compte d'un placement séquentiel entre les syntagmes (comme dans une grammaire de réécriture classique) ou de leur permutation libre, c'est à dire sans contrainte aucune entre leurs positions respectives. En dépit de cette limitation, il semble néanmoins adéquat pour modéliser simplement la grammaticalité de langues comme le latin, ou pour factoriser partiellement des réalisations équivalentes dans des langues comme l'allemand.

5.2 L'algèbre $\mathrm{Comm}(\Sigma)$

Mots commutatifs Nous introduisons maintenant la notion de mots modulo permutations, ou *mots commutatifs* sur un alphabet, qui sera le support de notre représentation des ordres libres. Ces mots commutatifs sont formés soit comme des mots usuels, par la concaténation successive de leurs lettres ; soit comme des mots libres dont les composants peuvent être lus indifféremment dans n'importe quel ordre. L'ensemble des mots commutatifs sur un alphabet est défini à partir des lettres de cet alphabet, d'une opération binaire de concaténation, et d'une autre opération binaire de composition libre, qui permet d'assembler des mots commutatifs sans spécifier leur ordre de lecture.

Définition 5.1. L'ensemble $\mathrm{Comm}(\Sigma)$ des *mots commutatifs* construits sur un alphabet Σ est l'ensemble des termes de l'algèbre suivante :

- Le mot vide ε est un terme de $\mathrm{Comm}(\Sigma)$.
- Tout élément $l \in \Sigma$ est un terme de $\mathrm{Comm}(\Sigma)$.
- L'opération binaire $\odot$ (*concaténation*) est associative.
- L'opération binaire $\otimes$ (*composition libre*) est associative/commutative.

Les termes de $\mathrm{Comm}(\Sigma)$ peuvent représenter de façon abstraite des phrases, dans lesquelles l'ordre de certains composants est laissé libre. Remarquons que tout terme construit sans utiliser l'opération $\otimes$ s'interprète de façon transparente comme un mot de Σ^* : l'opération $\odot$ est identique à la concaténation usuelle.

Notion d'équivalence Les propriétés des opérations $\odot$ et $\otimes$ (associativité et commutativité) décrivent une notion d'équivalence entre les termes

de Comm(Σ), sur laquelle nous nous appuierons fortement par la suite. En particulier, deux termes sont équivalents si la seule chose qui les distingue est l'ordre des arguments d'une composition libre.

Définition 5.2. Deux termes $t, t' \in \mathrm{Comm}(\Sigma)$ sont dits immédiatement équivalents si et seulement si l'une de ces trois conditions est vérifiée :

— $t = C[\odot(\odot(t_1, t_2), t_3)]$ et $t' = C[\odot(t_1, \odot(t_2, t_3))]$ (associativité de $\odot$)
— $t = C[\otimes(\otimes(t_1, t_2), t_3)]$ et $t' = C[\otimes(t_1, \otimes(t_2, t_3))]$ (associativité de $\otimes$)
— $t = C[\otimes(t_1, t_2)]$ et $t' = C[\otimes(t_2, t_1)]$ (commutativité de $\odot$)

La figure 5.3 illustre cette définition.

Par extension, l'*équivalence* de t et t', notée $t \equiv_c t'$, est définie comme la clôture réflexive, symétrique et transitive de cette relation.

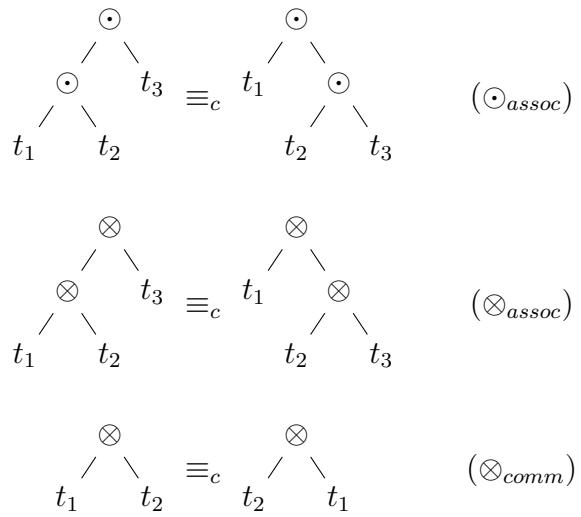


FIGURE 5.3 – Cas d'équivalence immédiate entre deux termes

Dans la suite, la notion d'équivalence issue de l'associativité de $\odot$ sera d'une importance réduite : l'équivalence entre $(a \odot b) \odot c$ et $a \odot (b \odot c)$, dénotant tous deux l'unique mot abc ne revêt pas d'importance particulière. En revanche, la combinaison de l'associativité et de la commutativité de $\otimes$ permettent de choisir librement l'ordre des composants d'un mot : ainsi, les termes $(a \otimes b) \otimes c$ et $a \otimes (c \otimes b)$ sont équivalents, matérialisant le fait que les mots abc et acb résultent tous deux de la composition libre des trois lettres a, b et c.

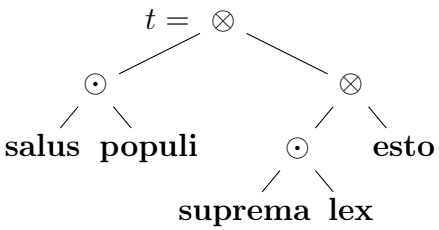
Langage d'un mot commutatif Nous définissons maintenant la notion de langage d'un terme de Comm(Σ) : il s'agit de l'ensemble des mots sur l'alphabet Σ pouvant être obtenus en choisissant un ordre linéaire quelconque pour les opérations de composition libre. Ces ordres peuvent être obtenus en faisant jouer les propriétés d'associativité et de commutativité de $\otimes$, puis en lisant simplement les feuilles des termes résultants de gauche à droite.

Définition 5.3. La *frontière* (ou *yield*) $\mathrm{frt}(t)$ d'un terme $t \in \mathrm{Comm}(\Sigma)$ est définie comme le mot de Σ^* obtenu en concaténant les feuilles de t de gauche à droite :

— $\mathrm{frt}(\varepsilon) = \varepsilon$
— $\mathrm{frt}(l) = l$ pour tout $l \in \Sigma$
— $\mathrm{frt}(\odot(t_1, t_2)) = \mathrm{frt}(t_1).\,\mathrm{frt}(t_2)$
— $\mathrm{frt}(\otimes(t_1, t_2)) = \mathrm{frt}(t_1).\,\mathrm{frt}(t_2)$

Le *langage* $\mathcal{L}(t)$ d'un terme t est défini comme l'ensemble des frontières des termes équivalents à t : $\mathcal{L}(t) = \{\mathrm{frt}(t') \mid t' \equiv_c t\}$.

Cette définition traduit la notion qu'un mot commutatif correspond à un ensemble de mots, formé de tous les choix possibles entre ses composants librement ordonnés. Une conséquence immédiate est que le langage d'un terme est fini, et que sa taille est bornée exponentiellement par la taille du terme dans le pire des cas. La figure 5.4 exemplifie la notion de langage d'un terme en listant les différentes permutations d'un terme représentant une phrase latine.



$$\mathcal{L}(t) = \left\{ \begin{array}{l} \text{Salus populi suprema lex esto.} \\ \text{Suprema lex salus populi esto.} \\ \text{Salus populi esto suprema lex.} \\ \text{Suprema lex esto salus populi.} \\ \text{Esto suprema lex salus populi.} \\ \text{Esto salus populi suprema lex.} \end{array} \right\}$$

FIGURE 5.4 – Terme représentant une phrase commutative et langage associé

Une autre propriété des termes de l'algèbre que nous avons définie pouvant en faciliter l'intuition est que le ré-ordonnancement des composants est local à un opérateur. En effet, les propriétés des opérateurs $\odot$ et $\otimes$ ne leur permettent pas d'interagir : les éléments d'un « groupe » combiné par $\otimes$ ne peuvent pas se mêler à ceux d'un autre groupe lorsque ces deux groupes sont séparés par l'opérateur $\odot$.

Une conséquence inattendue qui illustre ce principe est que le mot vide n'est *pas* un élément neutre pour $\mathrm{Comm}(\Sigma)$, comme illustré par les langages des termes de la figure 5.5. Sur la figure, les deux termes ont pour initialement pour frontière le mot *abc* : tous deux sont équivalents à un terme ayant

pour frontière *bac* par commutation de $\otimes(a, b)$, à un terme produisant *cab* par commutation de l'opérateur $\otimes$ situé à la racine, et incluent également dans leur langage le mot *cba* en commutant les deux occurrences de l'opérateur $\otimes$. Le premier terme t_1 est également équivalent à $\otimes(a, \otimes(b, c))$ par associativité de $\otimes$, permettant d'obtenir par commutation de $\otimes(b, c)$ la frontière *acb*, et le mot *bca* peut également être obtenu en commutant au préalable $\otimes(a, b)$. En revanche, le second terme t_2 n'inclut pas ces deux derniers mots dans son langage associé : la présence d'une opération de concaténation $\odot(x, \varepsilon)$ entre les deux occurrences de l'opérateur $\otimes$ interdit d'appliquer la règle d'associativité de ce dernier.

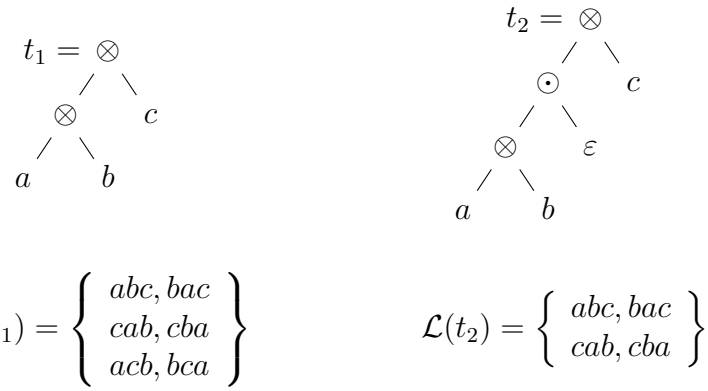


FIGURE 5.5 – Rupture de l'associativité de $\otimes$ par $\odot(\varepsilon, x)$

Alternativement, les mots commutatifs peuvent être envisagés comme des séquences (formées par $\odot$) ou des multi-ensembles [1] (formés par $\otimes$) de mots commutatifs plus petits. L'opérateur $\odot$ construit alors des séquences plus longues par concaténation de celles associées à ses arguments, et l'opérateur $\otimes$ construit des multi-ensembles comme l'union de ceux associés à ses arguments (un mot ou une lettre étant dans ce cas traité comme un singleton). Un multi-ensemble peut alors être réalisé comme un mot en concaténant tous ses éléments dans n'importe quel ordre.

5.3 Classes de grammaires pour Comm(Σ)

Les mots commutatifs décrits dans la section précédente constituent un outil de modélisation des énoncés contenant des composants librement ordonnés ; afin de modéliser des langues contenant de tels énoncés, nous avons maintenant besoin d'une notion de *grammaires commutatives*, produisant des langages de

1. Un multi-ensemble est un ensemble dont la fonction d'appartenance a pour domaine d'arrivée $\mathbb{N}$ au lieu de $\{0, 1\}$ [*cf.* Knuth, 1997, p. 694].

mots commutatifs. Étant donné que nos mots commutatifs sont définis comme des termes sur l'algèbre $\mathrm{Comm}(\Sigma)$, nous nous appuierons sur des grammaires de termes construisant des éléments de cette algèbre pour décrire des langages comportant des phénomènes d'ordres libres.

La classe de grammaires résultante manipule des tuples de contextes sur l'alphabet gradué $\{\odot_2, \otimes_2, \varepsilon_0\} \cup \Sigma_0$, à l'image des tuples de chaînes que manipulent les MCFG décrites section 2.4.1. Nous adoptons pour les productions de ces grammaires la notation décrite ci-dessous, similaire à celle de la définition 2.16.

Définition 5.4. Une *grammaire commutative* G est un tuple $(\Sigma, \mathcal{N}, S, \mathcal{P})$ où :
— Σ est l'*alphabet* de G.
— $\mathcal{N}$ est un ensemble de *symboles non-terminaux* typés.
— $S \in \mathcal{N}$ est un symbole non-terminal de type $[0]$ nommé *axiome*.
— $\mathcal{P}$ est un ensemble de *productions* construisant des tuples de contextes.

Les *types* des symboles non-terminaux (notés $\alpha, \beta, \dots$) sont des tuples (non-vides) de types simples construits à partir d'un unique type atomique 0 :
— 0 est un type simple (représentant un terme clos).
— Si τ et σ sont des types simples, $\sigma \to \tau$ est un type simple (contexte).
— Si $\tau_1 \dots \tau_n$ sont des types simples, $[\tau_1, \dots, \tau_n]$ est un type (tuple).

Les productions de $\mathcal{P}$ sont décrites au moyen de lambda-termes typés représentant des contextes, utilisant des tuples de variables issues du membre droit. Ainsi, toute production de $\mathcal{P}$ est de la forme :

$$A(M_1^{\alpha_1}, \dots M_n^{\alpha_n}) \leftarrow B_1(x_{1,1}, \dots, x_{1,n_1}) \dots B_p(x_{p,1}, \dots, x_{p,n_p})$$

où chaque M_i pour $i \in [n]$ est un lambda-terme linéaire simplement typé utilisant les constantes de $C = \{\odot^{0\to0\to0}, \otimes^{0\to0\to0}, \varepsilon^0\} \cup \{l^0 \mid l \in \Sigma\}$, et les variables libres $\mathrm{FV}(M_i) \subseteq \{x_{j,k} \mid j \in [p], k \in [n_j]\}$.

Toute production (ayant la forme décrite plus haut) doit respecter les types des non-terminaux de la façon suivante :
— Le non-terminal A a le type $[\alpha_1, \dots, \alpha_n]$.
— Chaque non-terminal B_j pour $j \in [p]$ est de type $[\alpha_{j,1}, \dots, \alpha_{j,n_j}]$.
— Toute variable $x_{j,k}$ est de type $\alpha_{j,k}$.

Une production dans laquelle l'une des variables $x_{i,j}$ est absente du membre gauche est dite *effaçante*. À l'inverse, une production dans laquelle une variable $x_{i,j}$ apparaît plusieurs fois dans le membre gauche est dite *parallèle*. Une grammaire commutative G contenant au moins une production effaçante (resp. parallèle) est également dite *effaçante* (resp. *parallèle*).

Dans la suite, nous désignerons fréquemment par la seule lettre $\mathcal{X}$ l'ensemble de variables $\{x_{j,k} \mid j \in [p], k \in [n_p]\}$ apparaissant dans une production de longueur p, dont les symboles non-terminaux sont d'arités respectives n_p. En outre, nos démonstrations portant sur les productions emploieront par convention la variable i pour désigner un indice allant de 1 à n (itérant sur le tuple

associé au membre gauche d'une production) et les variables j et k pour des indices appartenant respectivement à $[p]$ et $[n_j]$ (permettant de considérer l'ensemble des variables $x_{j,k}$ apparaissant dans une production). Enfin, sauf mention contraire, les productions et grammaires commutatives présentes dans ce chapitre ne seront ni effaçantes, ni parallèles.

Toute grammaire commutative définit un langage de termes de la même manière qu'une MCFG définit un langage de mots ; la contrainte sur le type de l'axiome (à savoir $[0]$) garantissant que les éléments de son langage représentent un unique terme clos. Par suite, le langage de mots d'une grammaire commutative est défini naturellement comme l'union des langages des termes qu'elle génère.

Définition 5.5. Étant donné une grammaire commutative $G = (\Sigma, \mathcal{N}, S, \mathcal{P})$, le langage de termes $\mathcal{T}_G(A)$ selon G d'un symbole non-terminal $A \in \mathcal{N}$ de type $[\alpha_1, \ldots, \alpha_n]$ est l'ensemble des tuples de lambda-termes $(P_1^{\alpha_1}, \ldots, P_n^{\alpha_n})$ respectant les conditions suivantes :

— $A(M_1, \ldots, M_n) \leftarrow B_1(x_{1,1}, \ldots, x_{1,n_1}) \ldots B_p(x_{p,1}, \ldots, x_{p,n_p})$ est dans $\mathcal{P}$.
— Pour tout $j \in [p]$, $\left(N_{j,1}, \ldots, N_{j,n_j}\right)$ appartient à $\mathcal{L}(B_j)$.
— Pour tout $i \in [n]$, $j \in [p]$ et $k \in [n_j]$, $P_i = M_i[x_{j,k} \leftarrow N_{j,k}]$.

Le *langage de termes* $\mathcal{T}(G)$ de G est alors défini comme celui de son axiome $\mathcal{T}_G(S)$. Par extension, le *langage* (de mots) $\mathcal{L}(G)$ de G est défini comme l'union des langages des termes appartenant à son langage de termes :

$$\mathcal{L}(G) = \bigcup_{t \in \mathcal{T}(G)} \mathcal{L}(t)$$

Nous considérons plusieurs classes de grammaires commutatives, comme candidats potentiels dans le but de capturer l'ensemble des langages naturels dotés de phénomènes d'ordres libres. Ces classes étendent aux ordres libres la hiérarchie des grammaires catégorielles abstraites du second ordre ($\mathbf{ACG}(2, m)$) sur les mots décrite dans la section 2.4.2, laquelle inclut les grammaires hors-contexte multiples (MCFG), hors-contexte (CFG) et régulières (RG). Du point de vue du traitement des langues, ces classes peuvent être considérées comme une tentative d'étendre aux ordres libres de la notion de grammaire légèrement sensible au contexte.

Comme pour les MCFG, nous obtenons nos différentes classes de grammaire en restreignant le type des symboles non-terminaux, et éventuellement la forme des productions.

Définition 5.6. Nous introduisons plusieurs classes de grammaires commutatives, définies par les contraintes suivantes :

— Une *grammaire hors contexte multiple commutative* (CMCFG) est une grammaire commutative non-restreinte. Ces grammaires associent des tuples de contextes à leurs symboles non-terminaux.

— Une *grammaire régulière multiple commutative* (CMREG) est une grammaire commutative dont tous les symboles non-terminaux sont de type $[\alpha_1, \ldots, \alpha_n]$, avec $\alpha_1 = \cdots = \alpha_n = 0$. Ces grammaires associent des tuples de termes clos à leurs symboles non-terminaux.
— Une *grammaire à macros commutative* (CMG) est une grammaire commutative dont tous les symboles non-terminaux sont de type $[\alpha]$. Ces grammaires associent un unique contexte à chacun de leurs symboles non-terminaux.
— Une *grammaire hors contexte commutative* (CCFG) est une grammaire commutative dont tous les symboles non-terminaux sont de type $[0]$. Ces grammaires associent un unique terme clos à chaque symbole non-terminal.
— Une *grammaire régulière commutative* (CRG) est une grammaire commutative dont les symboles non-terminaux sont de type $[0]$, et dont toutes les productions associent à leur membre gauche un contexte linéaire droit – c'est-à-dire un terme clos dont tous les fils gauches sont des feuilles constantes (lettres de Σ ou ε), et dont le dernier fils droit est l'unique variable, $x_{1,1}$.

À l'exception des CMREG, nos classes de grammaires commutatives sont nommées d'après les classes de grammaires de mots usuelles qu'elles généralisent (voir section 5.3.2). Le nom de la classe des CMREG provient pour sa part de la classe de grammaires de termes qui la sous-tend, les grammaires d'arbres régulières multiples [Mönnich, 2007].

La définition de nos différentes classes de grammaires commutatives induit immédiatement une hiérarchie, représentée par la figure 5.6. Les flèches représentent les relations d'inclusion entre les différentes classes.

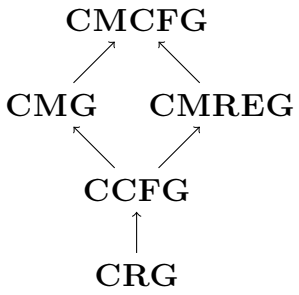


FIGURE 5.6 – Hiérarchie des grammaires commutatives

Illustration Nous illustrons maintenant le fonctionnement de nos grammaires, au travers de deux courts exemples.

Soit $G_1 = (\Sigma, \mathcal{N}_1, S, \mathcal{P}_1)$ la grammaire définie sur l'alphabet $\Sigma = \{a, b, c\}$ au moyen des non-terminaux suivants : $\mathcal{N}_1 = \left\{S^{[0]}, A^{[0\to0]}, B^{[0\to0]}, C^{[0\to0]}\right\}$. Les productions de $\mathcal{P}_1$ sont les suivantes :

1. $S(x\ \varepsilon) \leftarrow A(x)$
2. $A(\lambda t^0.\ \odot\ a\ (x\ t)) \leftarrow B(x)$
3. $B(\lambda t^0.x\ (\odot\ b\ t)) \leftarrow C(x)$
4. $C(\lambda t^0.\ \odot\ (x\ t)\ c) \leftarrow A(x)$
5. $A(\lambda t^0.t) \leftarrow$

Cette grammaire construit au moyen de ses non-terminaux A, B et C un contexte unaire, ayant la forme donnée par la figure 5.7. La production 5 (terminale) permet de partir d'un contexte trivial, et les productions 2 à 4 ajoutent respectivement un a concaténé à gauche au dessus du contexte x, un c concaténé à droite au dessus de x, et un b concaténé en dessous de x. Enfin, la première production complète un contexte produit par A au moyen du symbole ε (qui se substitue à t^0 dans la figure).

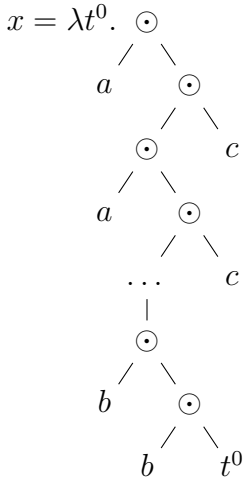


FIGURE 5.7 – Forme des contextes produits par le non-terminal A de G_1

Les termes construits par G_1 ne contenant que le symbole $\odot$, le langage de mots $\mathcal{L}(G_1)$ qui lui est associé s'obtient immédiatement en considérant l'ensemble des frontières des éléments de son langage de termes. Ceux-ci ayant tous la forme illustrée par la figure 5.7, leur frontière est un mot contenant n occurrences de la lettre a à gauche, n occurrences de c à droite, et n occurrences de b entre les deux ; par conséquent : $\mathcal{L}(G_1) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$.

Considérons maintenant une seconde grammaire $G_2 = (\Sigma, \mathcal{N}_2, S, \mathcal{P}_2)$ qui soit construite sur le même alphabet, avec $\mathcal{N}_2 = \left\{S^{[0]}, A^{[0,0,0]}\right\}$, et utilisant les productions suivantes :

1. $S(\otimes\ (\otimes\ x_1\ x_2)\ x_3) \leftarrow A(x_1, x_2, x_3)$
2. $A(\otimes\ a\ x_1, \otimes\ b\ x_2, \otimes\ c\ x_3) \leftarrow A(x_1, x_2, x_3)$
3. $A(\varepsilon, \varepsilon, \varepsilon) \leftarrow$

Le langage du non-terminal A est formé par des triplets de termes, combinant un nombre égal d'occurrences des symboles a, b et c (respectivement) au moyen de l'opérateur $\otimes$; l'axiome S combine à son tour les termes résultants au moyen du même opérateur. Ainsi, les termes de $\mathcal{T}(G_2)$ ont pour frontière un mot $a^n b^n c^n$, et tous leurs nœuds internes sont formés par l'opérateur $\otimes$. Les règles d'équivalence et la notion de langage d'un terme données par les définitions 5.2 et 5.3 permettent alors de réordonner librement les feuilles des termes générés par G_2 ; par conséquent, $\mathcal{L}(G_2) = \{w \mid |w|_a = |w|_b = |w|_c\}$.

Enfin, observons que les types des non-terminaux de G_1 sont des tuples de taille 1 (contenant soit le type atomique 0, soit le type d'un contexte unaire $0 \to 0$) ; par conséquent $G_1 \in \mathbf{CMG}$. En ce qui concerne G_2, les types de ses non-terminaux sont des tuples de taille variable, contenant uniquement le type atomique 0 ; par conséquent, $G_2 \in \mathbf{CMREG}$.

5.3.1 Semilinéarité

Une propriété essentielle des langages issus des grammaires commutatives que nous décrivons est leur semilinéarité. Cette propriété traduit le fait que le nombre de lettres dans les mots d'un tel langage respecte certaines proportions.

Définition 5.7. Un langage L est dit *semilinéaire* si son image de Parikh (formée par l'ensemble des images de Parikh de ses mots, voir section 2.1.2) constitue un ensemble semilinéaire (définis section 2.1.4).

La semilinéarité d'un langage est directement liée à la notion de *croissance constante* évoquée par Joshi [1985], qui est une propriété essentielle des langages légèrement sensibles au contexte[Joshi *et al.*, 1990]. Bien que ces deux propriétés ne soient pas exactement équivalentes (voir Kallmeyer [2010]), la première implique la seconde, et capture également l'intuition qui a présidé à sa formulation. Les CFG (et plus généralement les MCFG) produisent des langages semilinéaires, et la semilinéarité de nos langages de mots commutatifs peut être montrée par un argument de pompage exactement similaire à celui établissant la semilinéarité des langages hors-contexte, montrée originellement par Parikh [1966]. Une preuve claire et accessible suivant ce schéma pour les CFG peut se trouver dans Kozen [1997].

5.3.2 Généralisation des grammaires de mots

La hiérarchie que nous proposons pour les grammaires commutatives reflète la hiérarchie existante des MCFG sur les chaînes évoquée dans la section 2.4.1 ; et elle inclut successivement les classes de langages décrites par ces grammaires. Plus précisément, les langages décrits par nos grammaires coïncident avec les classes de langages de mots usuelles en interdisant l'emploi du symbole $\otimes$ dans les productions. Ce résultat est immédiatement apparent pour les classes de

grammaires commutatives manipulant des termes clos : un terme t combinant des lettres à l'aide du seul opérateur de concaténation $\odot$ équivaut immédiatement au mot obtenu en lisant sa frontière $\mathrm{frt}(t)$.

Plus généralement, cette correspondance peut être établie en exploitant des résultats existants dans la littérature des ACG : il est possible de construire une ACG d'ordre 2 sur les chaînes à partir d'une CMCFG sans $\otimes$, qui reconnaisse le même langage. Cette ACG peut successivement être simulée par un transducteur déterministe cheminant dans un arbre (DTWT) comme montré par Salvati [2007], puis par un système de réécriture linéaire hors-contexte (LCFRS), et donc par une MCFG, d'après Weir [1992].

Nous esquissons brièvement la technique permettant de construire une 2-ACG à partir d'une CMCFG. Cette construction exploite notre choix de représenter nos grammaires d'une manière similaire aux MCFG à l'aide du lambda-calcul, et reprend la méthode usuelle permettant de simuler un formalisme hors-contexte par une ACG, détaillée dans de Groote et Pogodalla [2004]. Pour rappel, la définition d'une ACG est donnée en 2.18. Soit $G = (\Sigma, \mathcal{N}, S, \mathcal{P})$ une CMCFG, nous construisons une ACG $G' = (\Sigma_a, \Sigma_c, \Lambda, S)$ dans laquelle :

- Le vocabulaire abstrait Σ_a représente les dérivations de G de la manière suivante : l'ensemble des types atomiques de Σ_a est formé par l'ensemble $\mathcal{N}$ des non-terminaux de G, et l'ensemble des constantes est formé par l'ensemble $\mathcal{P}$ de ses productions. Le type des constantes est donné par la structure des productions, c'est-à-dire que la constante p correspondant à une production de la forme $A(\dots) \leftarrow B_1(\dots)\dots B_n(\dots)$ a pour type $\tau_p = B_1 \to \cdots \to B_n \to A$. Les termes en forme normale construits sur Σ_a représentent ainsi des arbres de dérivation selon G, le typage des constantes garantissant leur bonne formation.
- Le vocabulaire concret Σ_c représente des mots sur Σ^* de la manière suivante : son unique type atomique est $*$, et ses constantes sont les lettres de Σ (ou ε), toutes munies du type $* \to *$. Cette représentation suit le schéma décrit dans la section 2.2.4.
- Le lexique Λ associe à chaque production un lambda-terme représentant son membre gauche, utilisant éventuellement la représentation des tuples et des projections évoquée par la section 2.2.4. L'image d'une feuille a^0 est la constante $a^{*\to *}$ associée à la lettre de Σ correspondante, et l'image de la constante $\odot^{0\to 0\to 0}$ est directement remplacée par le terme $\lambda f g x.f\ (g\ x)$ de type $(* \to *) \to (* \to *) \to * \to *$ dénotant la concaténation. Le terme du langage concret associé à une dérivation valide se normalise ainsi en une représentation de l'unique chaîne associée au terme de Comm(Σ) dérivé.
- Le type distingué S de G' est le type atomique associé à l'axiome de G.

Observons que dans le cas d'une CMG, il n'est pas nécessaire de représenter les tuples associés aux non-terminaux (puisque ceux-ci sont tous de taille 1). La construction précédente permet alors à une CMG d'être représentée par une

ACG munie d'un lexique de complexité 3, une CCFG employant un lexique de complexité 2 et une CRG correspondant à un lexique de complexité 1. Ce résultat reflète la hiérarchie des ACG mise en valeur dans la section 2.4.2.

Nous récapitulons dans la figure 5.8 la hiérarchie des classes de *langages* issues des différentes classes de grammaires commutatives : une flèche simple représente une relation d'inclusion stricte, et les flèches doubles décorées par « ? » dénotent une relation d'inclusion ou d'égalité ; l'abréviation « … L » désigne la classe de langages $\mathcal{L}(\ldots \mathbf{G})$. Les résultats d'inégalité entre les différentes classes découlent de résultats de difficulté algorithmique présentés dans la prochaine section. Pour référence, les classes usuelles de langages légèrement sensibles au contexte figurent également sur le schéma, reliées aux classes de langages de mots commutatifs qui les généralisent par un trait simple.

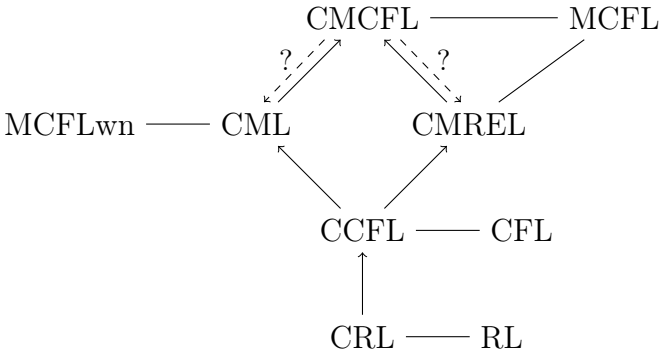


FIGURE 5.8 – Hiérarchie des langages de mots commutatifs

5.4 Problème de l'analyse

Une fois proposées les classes de grammaires permettant de décrire des langages de mots commutatifs, nous souhaitons étudier plus avant le problème de l'analyse. En raison de l'étape intermédiaire due à la notion de langage d'un terme commutatif, la relation entre un arbre de dérivation selon une grammaire commutative et le mot qu'il reconnaît n'est pas complètement triviale ; elle doit tenir compte de l'associativité et de la commutativité de l'opérateur $\otimes$. Toutefois, cette étape intermédiaire peut-être simplifiée : intuitivement, les mots commutatifs combinés par cet opérateur ont simplement besoin d'être comptés. Par exemple, décider si un mot appartient au langage de la grammaire G_2 décrite page 123 revient simplement à compter les occurrences de a, b et c, ce qui est algorithmiquement simple : de fait, il existe une grammaire commutative régulière décrivant le même langage, et pour laquelle, comme nous allons le voir, l'analyse d'un mot w ne requiert pas d'établir explicitement l'équivalence entre le terme dérivé par la grammaire et un terme t ayant pour frontière $\mathrm{frt}(t) = w$.

Analyse et analyse universelle Nous étudions les deux variantes suivantes du problème de décision lié à l'analyse :

— Étant donné un mot w et une grammaire G, est-ce que $w \in \mathcal{L}(G)$? (Analyse universelle.)

— Étant donné un mot w, est-ce que $w \in \mathcal{L}(G)$? (Analyse selon une grammaire G fixée à l'avance.)

La différence entre ces deux variantes tient dans la complexité algorithmique de la procédure : l'analyse universelle dépend de la taille de la grammaire G (et du mot w), alors que l'analyse selon une grammaire fixée ne dépend que de w, la taille de la grammaire étant considérée comme une constante de l'algorithme. Cette différence trouve son importance dans des applications liées au traitement automatique des langues. D'une part, beaucoup d'applications pratiques requièrent l'analyse de nombreux énoncés selon une grammaire fixe, offrant une certaine latitude pour pré-compiler ou optimiser l'analyseur correspondant. D'autre part, il est envisageable pour des raisons psycholinguistiques que nombre de langages naturels comprenant des phénomènes d'ordres libres ne requièrent pas en pratique toute la complexité atteignable par notre formalisme dans le pire cas : de tels langages pourraient éventuellement être décrits une grammaire commutative relativement peu coûteuse par rapport à d'autres grammaires de sa classe.

Résultats du chapitre La table 5.1 résume l'ensemble des résultats de complexité établis dans ce chapitre, incluant le cas des grammaires avec effacement. Pour chaque classe, le suffixe « -C » désigne un problème complet pour la classe en question et « -D » un problème difficile pour cette classe.

Classe	Analyse fixée	Analyse universelle	
		Non-effaçante	Effaçante
Terme	$\mathcal{O}(1)$	NP-C	
CRG	NL	NP-C	
CCFG	LogCFL-C	NP-C	
CMG	NP-C	PSPACE-D	EXPTIME
CMREG	NP-C	PSPACE-C	EXPTIME-C
CMCFG	NP-C	PSPACE-D	EXPTIME-C

Table 5.1 – Récapitulatif des résultats de complexité pour l'analyse

La plupart des algorithmes d'analyse établissant ces résultats s'appuient sur le système de réécriture $\rightarrow_\varepsilon$ introduit dans la section 5.6, qui permet de compresser les termes commutatifs ; ces algorithmes sont donc détaillés dans la dernière section du chapitre. L'analyse à grammaire fixée des CRG et CCFG, qui est polynomiale, fait l'objet de deux algorithmes dédiés, détaillés dans la section 5.5. Enfin, cette section ci établit les bornes inférieures sur la complexité de l'analyse. Certaines sont issues de la littérature des MCFG, que nos

grammaires peuvent émuler directement à l'aide de l'opérateur $\odot$; d'autres sont propres à l'emploi de l'opérateur commutatif $\otimes$. Nous établissons également la NP-complétude de l'analyse universelle par un terme commutatif, consistant à déterminer si un mot w appartient à $\mathcal{L}(t)$ étant donné w et t.

5.4.1 Résultats issus des MCFG

La classe de complexité LogCFL caractérise l'ensemble des problèmes réductibles en espace logarithmique à la reconnaissance d'un langage hors-contexte. Par conséquent, l'analyse d'un mot selon une CCFG fixée (capable de simuler directement une CFG, comme mentionné dans la section 5.3.2) constitue un problème LogCFL-difficile.

Suivant le même raisonnement, l'analyse universelle pour les CMG et CMREG (capables de simuler une MCFG_{WN}) est PSPACE-difficile dans le cas des grammaires sans effacement. Ces résultats de complexité sur les MCFG, établis par Kaji *et al.* [1992], ont également inspiré le fonctionnement de l'algorithme d'analyse général décrit à la fin de ce chapitre, qui établit la complexité maximale de l'analyse universelle pour les CMG, CMREG et CMCFG. Enfin, l'analyse universelle selon des CMREG (et donc CMCFG) incluant des productions effaçantes (et donc capables de simuler une MCFG avec effacement) est, pour les mêmes raisons, EXPTIME-difficile ; ce résultat ne s'étend toutefois pas nécessairement aux CMG, dont la complexité est liée aux grammaires sans entrelacement (MCFG_{WN}).

La table 5.2 reprend le récapitulatif des résultats de complexité du chapitre, en mettant en valeur ceux qui sont issus des MCFG.

Classe	Analyse fixée	Analyse universelle	
		Non-effaçante	Effaçante
Terme	$\mathcal{O}(1)$	NP-C	
CRG	NL	NP-C	
CCFG	LogCFL-D	NP-C	
CMG	NP-C	PSPACE-D	EXPTIME
CMREG	NP-C	PSPACE-D	EXPTIME-D
CMCFG	NP-C	PSPACE-D	EXPTIME-D

Table 5.2 – Résultats de complexité connus, issus des MCFG

5.4.2 Appartenance au langage d'un terme

Nous considérons maintenant le problème de l'appartenance d'un mot w au langage d'un terme commutatif t : ce problème est NP-complet. Nous décrivons dans cette sous-section un algorithme non-déterministe, s'exécutant en

temps polynomial, capable de décider si $w \in \mathcal{L}(t)$. La NP-difficulté de l'appartenance d'un mot au langage d'un terme commutatif sera, pour sa part, établie par réduction du problème (NP-complet) de 3-partition. Une conséquence immédiate de cette NP-difficulté est que l'analyse universelle selon une CCFG est également NP-difficile : pour tout terme t, il est possible de construire directement une CCFG dont l'axiome se réécrit en t, et dont le langage est donc exactement $\mathcal{L}(t)$.

Classe	Analyse fixée	Analyse universelle	
		Non-effaçante	Effaçante
Terme	$\mathcal{O}(1)$	NP-D	
CRG	NL	NP-C	
CCFG	LogCFL-C	NP-D	
CMG	NP-C	PSPACE-D	EXPTIME
CMREG	NP-C	PSPACE-C	EXPTIME-C
CMCFG	NP-C	PSPACE-D	EXPTIME-C

Table 5.3 – Résultats de complexité de la sous-section 5.4.2

Reconnaissance d'un mot selon un terme commutatif Soit w un mot de Σ^* et t un terme commutatif appartenant à $\mathrm{Comm}(\Sigma)$; par définition, s'il est possible de deviner un terme t' tel que $t \equiv_c t'$ et $\mathrm{frt}(t') = w$, alors $w \in \mathcal{L}(t)$. Il en résulte un algorithme non-déterministe, qui s'exécute en temps polynomial, puisque $|t| = |t'|$; la frontière w de t' peut être vérifiée immédiatement et l'équivalence entre t et t' peut être établie en fusionnant les occurrences voisines de chaque opérateur (aplatissant le terme), et en comparant les arbres résultants de manière descendante, modulo permutation des branches dominées par $\otimes$. La figure 5.9 illustre ce procédé, la permutation $\pi(1,2,3) = (2,3,1)$ permettant de choisir quels sont les sous-arbres à comparer deux à deux.

Problème de 3-partition (3-PART) Nous donnons d'abord une définition de 3-PART, qui est un problème NP-complet :

Instance Un ensemble $S = \{n_1 \dots n_{3m}\}$ d'entiers, et une constante k telle que $\frac{k}{4} < n < \frac{k}{2}$ pour tout $n \in S$, et que :

$$\sum_{n \in S} n = k.m$$

Solution *Vrai* si et seulement si il existe une partition $S_1 \dots S_m$ de S telle que chaque ensemble S_i respecte :

$$\sum_{n \in S_i} n = k$$

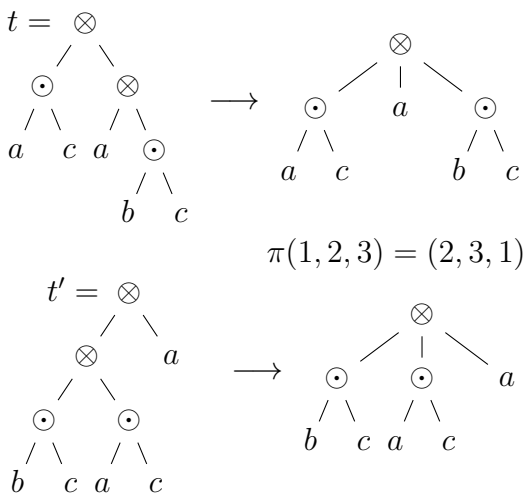


FIGURE 5.9 – Vérification de l'équivalence entre deux termes

Les contraintes initiales sur les n_i entraînent que tous les ensembles S_i d'une éventuelle solution contiennent exactement 3 éléments, notés $n_{i,1}$, $n_{i,2}$ et $n_{i,3}$. La figure 5.10 propose une représentation visuelle d'une instance de 3-PART et de sa solution, consistant en un ré-arrangement des éléments de S par groupes de 3, chaque ligne de la figure du bas correspondant à un des S_i.

Théorème 5.1 (Garey et Johnson [1990])**.** *Le problème* 3-PART *est* NP*-complet.*

Théorème 5.2. *L'appartenance d'un mot w au langage d'un terme commutatif t est un problème* NP*-complet.*

Démonstration. L'algorithme non-déterministe polynomial proposé plus haut permet de décider si $w \in \mathcal{L}(t)$, entraînant l'appartenance du problème à NP.

La preuve de NP-difficulté procède par réduction de 3-PART. Étant donné une instance $I = (S, k)$ de 3-PART nous construisons un terme t et un mot w tels que $w \in \mathcal{L}(t)$ si et seulement si I a une solution ; cette construction est illustrée par la figure 5.11. Soit $\Sigma = \{a, \#\}$ l'alphabet sur lequel sont construits t et w. Nous posons $w = (a^k\#)^m$, formant le mot à reconnaître en séparant m séquences de a, chacune de longueur k, à l'aide du symbole $\#$. Nous construisons pour tout $n \in S$ les termes $t_n = \odot(a, x)^n[\varepsilon]$ concaténant n fois la lettre a ; ainsi que le terme $t_\# = \otimes(\#, x)^m[\varepsilon]$ combinant librement m fois le symbole $\#$. Nous considérons alors le terme $t = \otimes(t_{n_1}, \cdots \otimes (t_{n_{3m}}, t_\#)\dots)$ combinant librement chacun des termes t_n et le terme $t_\#$, et montrons que $w \in \mathcal{L}(t) \Leftrightarrow$ **3-PART**$(I) = \mathit{Vrai}$.

Nous montrons d'abord la réciproque de l'implication. Si l'instance I a une solution $S_1 \dots S_m$, alors il existe m triplets d'entiers dans S dont la somme est égale à k. À chacun de ces triplets correspondent trois sous-termes de

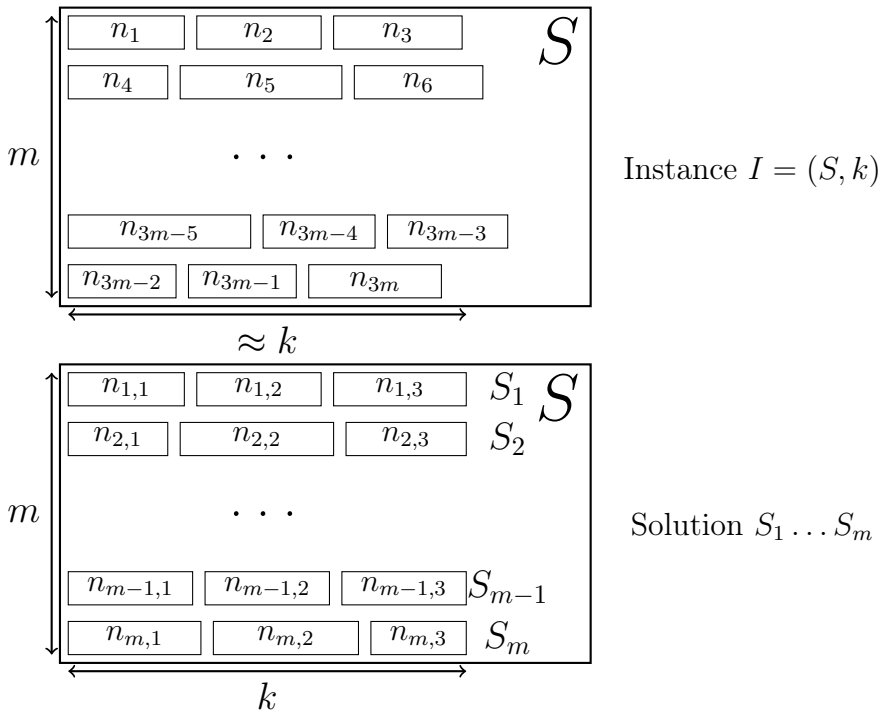


FIGURE 5.10 – Instance et solution associée de 3-PART

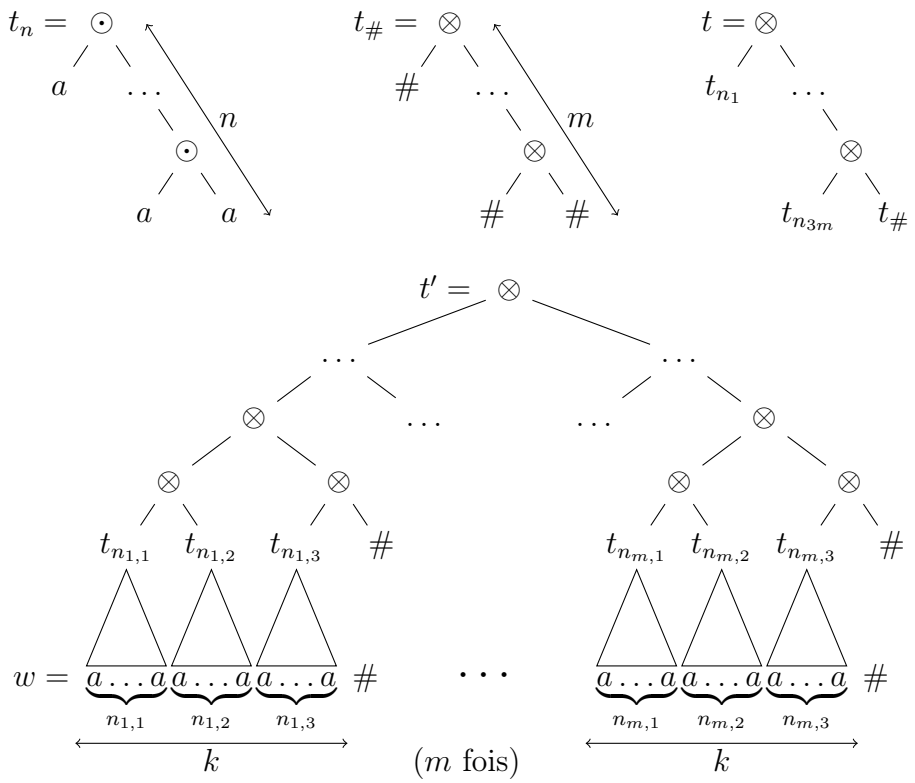


FIGURE 5.11 – Réduction d'une instance de 3-PART à $w \in \mathcal{L}(t)$

t, combinés au moyen de l'opérateur $\otimes$. Nous pouvons donc faire jouer les propriétés d'associativité et de commutativité de ce dernier pour regrouper ces triplets et les faire suivre d'un des m symboles # ajoutés par le sous-terme $t_{\#}$, formant ainsi un terme t' équivalent à t tel qu'illustré par la figure 5.11. La frontière de t' est alors exactement w, montrant que $w \in \mathcal{L}(t)$.

Nous montrons maintenant l'implication directe. Si $w \in \mathcal{L}(t)$, alors il existe un terme $t' \equiv_c t$ tel que $\mathrm{frt}(t') = w$. L'application des règles d'équivalence portant sur l'opérateur $\odot$ ne permet pas de de mêler les nœuds appartenant aux sous-termes t_n avec les autres, ni de modifier leur frontière. Le mot w étant la frontière de t', il contient donc nécessairement un facteur distinct a^n pour chaque $n \in S$. Or, w se compose par construction de m facteurs identiques de la forme $a^k\#$; par conséquent, nous pouvons déduire de la structure de t' une partition des sous-termes t_n dans laquelle chaque sous-ensemble contient k occurrences du symbole a. Cette partition se traduit directement en une partition de l'ensemble S dans laquelle la somme des éléments de chaque sous-ensemble est k, ce qui constitue une solution pour l'instance I. □

5.4.3 Résultats de difficulté supplémentaires

Nous établissons maintenant trois résultats de NP-difficulté, portant respectivement sur l'analyse universelle selon une CRG, et sur l'analyse à grammaire fixée selon une CMG et une CMREG. Ces bornes inférieures de complexité sont en fait optimales, mais l'existence d'algorithmes dans NP pouvant résoudre ces problèmes sera montrée à l'issue de ce chapitre, dans la section 5.6.3, leur fonctionnement requérant une notion de « compression » des termes qui en élimine les composantes inutiles à la formation du mot w.

Classe	Analyse fixée	Analyse universelle	
		Non-effaçante	Effaçante
Terme	$\mathcal{O}(1)$	NP-C	
CRG	NL	NP-D	
CCFG	LogCFL-C	NP-D	
CMG	NP-D	PSPACE-D	EXPTIME
CMREG	NP-D	PSPACE-C	EXPTIME-C
CMCFG	NP-D	PSPACE-D	EXPTIME-C

Table 5.4 – Résultats de complexité de la sous-section 5.4.3

La NP-difficulté de l'analyse selon une CMG et une CMREG est établie, dans les deux cas, par une réduction à 3-PART similaire à celle décrite plus haut. Ces résultats entraînent immédiatement que l'analyse selon une CMCFG fixée est NP-difficile. Le cas de l'analyse universelle selon une CRG est, quant à lui, établi par réduction au problème de 3-couverture exacte, également NP-complet. Observons toutefois que la preuve de ce ce dernier résultat repose

crucialement sur la taille de l'alphabet Σ sur lequel repose la grammaire, et n'est plus valide si le mot w à analyser est construit sur un alphabet de taille constante. De fait, la complexité de l'analyse universelle à alphabet fixé est NL, identique à celle de l'analyse à grammaire fixée établie dans la prochaine section.

La construction de ces réductions, ainsi que leurs preuves détaillées, sont données en annexe dans la section B.1.

5.5 Algorithmes d'analyse pour les cas restreints

Nous présentons maintenant deux algorithmes d'analyse à grammaire fixée, destinés respectivement aux CRG et aux CCFG. Nous montrerons la correction et la complétude, ainsi que la complexité de ces algorithmes, qui demeure polynomiale dans le pire cas. En particulier, la complexité de l'analyse pour une CCFG fixée est la même que pour l'analyse d'une CFG.

Classe	Analyse fixée	Analyse universelle	
		Non-effaçante	Effaçante
Terme	$\mathcal{O}(1)$	NP-C	
CRG	NL	NP-C	
CCFG	LogCFL	NP-C	
CMG	NP-C	PSPACE-D	EXPTIME
CMREG	NP-C	PSPACE-C	EXPTIME-C
CMCFG	NP-C	PSPACE-D	EXPTIME-C

Table 5.5 – Résultats de complexité de la section 5.5

Ces algorithmes alternent entre deux stratégies pour analyser des fragments d'un mot commutatif. Intuitivement, une série de fragments consécutifs d'un mot décrit par une grammaire commutative peut avoir été introduite de deux façons. Le premier cas est celui où le terme dérivé par la grammaire combine une série de symboles à l'aide de l'opérateur de concaténation $\odot$: dans ce cas, l'analyse procède de manière similaire à celle des grammaires hors contexte. Le second cas est celui où les composants ont pu être réordonnés, et descendent donc d'occurrences de l'opérateur $\otimes$ voisines dans le terme dérivé : dans ce cas, il s'agit d'un problème de comptage – le nombre de composants doit alors être cohérent avec ce que permettent les productions commutatives de la grammaire (à savoir celles utilisant uniquement l'opérateur $\otimes$) ; ce problème revient à décider de l'appartenance d'un vecteur à un ensemble semilinéaire.

5.5.1 Notions supplémentaires

Afin de simplifier la description des algorithmes d'analyse, nous introduisons deux notions intermédiaires : la première est celle de forme normale pour les CCFG, similaire à la notion de forme normale usuelle des grammaires de termes ou à la celle de forme normale de Chomsky pour les grammaires hors-contexte ; la seconde est celle de langage commutatif d'un symbole non-terminal, représentant l'ensemble des termes dérivables à partir d'un symbole non-terminal utilisant exclusivement l'opérateur $\otimes$.

En outre, par souci de simplification, nous exploitons librement dans cette section le fait que, dans une CCFG, tout symbole non-terminal dans une dérivation est associé à un unique terme clos. Nous parlerons ainsi du *langage étendu* (ou *langage non-terminal*) de termes d'un symbole non-terminal A selon une grammaire $G = (\mathcal{N}, \Sigma, S, \mathcal{P})$, formé de l'ensemble des termes commutatifs sur $\mathrm{Comm}(\Sigma \cup \mathcal{N})$ dérivables à partir de A en utilisant les productions de $\mathcal{P}$. Ainsi, si $A(\otimes\ x\ y) \leftarrow B(x)\ C(y)$ est une production de $\mathcal{P}$, alors $\otimes\ B\ C$ appartient au langage de termes étendu de A ; par extension, les mots BC et CB formés sur $(\Sigma \cup \mathcal{N})^*$ appartiennent au langage (de mots) étendu de A.

Forme normale d'une CCFG Une grammaire hors contexte commutative en forme normale utilise exclusivement les formes suivantes dans ses productions :

$$\begin{array}{lll} (1) & A(op\ x\ y) \leftarrow B(x)\ C(y) & \\ (2) & A(op\ a\ x) \leftarrow B(x) & \text{avec } \begin{cases} op \in \{\odot, \otimes\} \\ a \in \Sigma \cup \{\varepsilon\} \end{cases} \\ (3) & A(a) \leftarrow & \end{array}$$

Une CCFG (resp. une CRG) est dite en *forme normale* si et seulement si toutes ses productions sont de la forme 1 ou 3 (resp. 2 ou 3). Pour toute CCFG ou CRG G, il existe une grammaire équivalente G' en forme normale, telle que $\mathcal{T}(G') = \mathcal{T}(G)$. Nous supposons dans le reste de cette section, sans perte de généralité, que toutes les grammaires que nous manipulons sont en forme normale.

Langage commutatif d'un symbole non-terminal Le langage commutatif d'un symbole non-terminal est l'ensemble des termes étendus qu'il permet de dériver contenant exclusivement l'opérateur $\otimes$. Ce langage a pour intérêt d'être décidable par simple comptage des lettres d'un mot, puisque celles-ci peuvent être permutées librement par associativité et commutativité de $\otimes$.

Étant donné une grammaire commutative $G = (\mathcal{N}, \Sigma, S, \mathcal{P})$, le *fragment commutatif* de G est la grammaire commutative $G' = (\mathcal{N}, \Sigma, S, \mathcal{P}')$, dont les productions excluent l'emploi de l'opérateur $\odot$. Ainsi, $\mathcal{P}'$ contient uniquement les productions suivantes :

— Si une production $A(\otimes\ x\ y) \leftarrow B(x)\ C(y)$ appartient à $\mathcal{P}$, alors elle appartient également à $\mathcal{P}'$.
— Si une production $A(a) \leftarrow$ appartient à $\mathcal{P}$, alors elle appartient à $\mathcal{P}'$.

Par suite, le *langage commutatif* d'un non-terminal A selon G est défini comme le langage étendu de A selon le fragment commutatif G' de G. Ce langage étant construit exclusivement à l'aide de l'opérateur $\otimes$, l'ordre des lettres composant ses mots est indifférent : l'image de Parikh d'un mot w de $(\Sigma \cup \mathcal{N})^*$ suffit à décider de son appartenance au langage commutatif de A. Aussi, nous nous intéresserons uniquement dans la suite à l'image de Parikh de ce dernier, notée $\psi(G, A)$; et, par abus de langage, l'expression *langage commutatif de A selon G* désignera directement $\psi(G, A)$.

5.5.2 Algorithme d'analyse pour les CRG

Nous présentons d'abord l'algorithme d'analyse selon une CRG, sous la forme d'un ensemble de règles d'inférence, et détaillons son fonctionnement avant de démontrer ses propriétés. L'algorithme est donné par la figure 5.12 ; il permet de construire des *faits de dérivation* formés comme des quintuplets ou comme le symbole spécial $\top$. Un mot $w = l_1 \dots l_{|w|}$ est reconnu par une CRG $G = (\mathcal{N}, \Sigma, S, \mathcal{P})$ si et seulement si le fait $\top$ est dérivable à partir du fait $(S, \vec{0}, S, 0, |w|)$.

$$\frac{(A, \vec{0}, A, i, j) \quad A(\odot\ a\ x) \leftarrow B(x) \in \mathcal{P} \quad a = l_{i+1} \vee a = \varepsilon}{(B, \vec{0}, B, i + |a|, j)}\ \textsc{Analyse}$$

$$\frac{(A, v, B, i, j) \quad B(\otimes\ a\ x) \leftarrow C(x) \in \mathcal{P} \quad \|v\| \leq |w|}{(A, v + \vec{1}_a, C, i, j)}\ \textsc{Hypothèse}$$

$$\frac{(A, v, B, i, j) \quad v + \vec{1}_B \in \psi(G, A) \quad v = \psi(w, i, i') + \psi(w, j', j)}{(B, \vec{0}, B, i', j')}\ \textsc{Confirmation}$$

$$\frac{(A, v, B, i, j) \quad B(a) \leftarrow \in \mathcal{P} \quad v + \vec{1}_a = \psi(w, i, j)}{\top}\ \textsc{Réussite}$$

FIGURE 5.12 – Algorithme d'analyse selon une CRG

L'algorithme procède en déduisant de manière descendante un arbre de dérivation associé au mot w, tout en conservant continuellement en mémoire un fait de dérivation traduisant sa progression dans la reconnaissance de w. Un fait de dérivation est soit le symbole $\top$, soit un quintuplet (A, v, B, i, j), formé d'un symbole non-terminal $A \in \mathcal{N}$, d'un vecteur $v \in \mathbb{N}^{\Sigma \cup \mathcal{N}}$, d'un autre symbole non-terminal $B \in \mathcal{N}$ (possiblement égal à A), et de deux positions $i, j \in [|w|]$ dans le mot w. Le fait $\top$ dénote le succès de l'analyse de w par G, tandis qu'un

quintuplet dénote que le non-terminal A correspond à la position i dans w, et qu'il se réécrit en un non-terminal B après avoir introduit entre i et j, au moyen de l'opérateur $\otimes$, l'ensemble de lettres compté par le vecteur v. Observons que si la dernière étape de la dérivation est formée par une production employant l'opérateur non-commutatif $\odot$, le fait correspondant est toujours de la forme $(A, \vec{0}, A, i, j)$, et que dans ce cas, chaque étape de la dérivation fait progresser i de 1 vers la fin du mot w. La figure 5.13 montre un terme linéaire droit t appartenant au langage $\mathcal{T}(G)$ d'une grammaire G, et son terme équivalent t' dont la frontière est $w = w_1 w'_2 w_3 w'_4 w''_2$. Nous nous appuierons sur cette figure pour détailler le fonctionnement de l'algorithme.

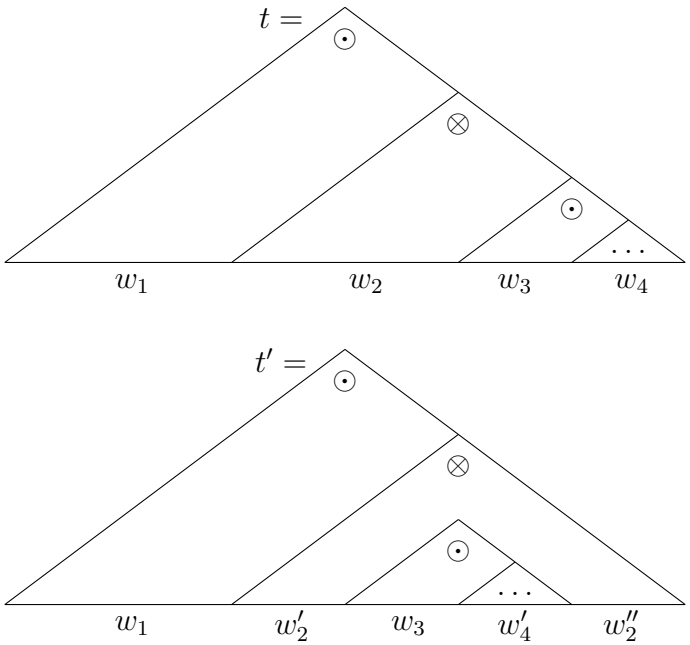


FIGURE 5.13 – Forme générale d'une dérivation CRG

Nous détaillons maintenant chacune des règles de l'algorithme et expliquons son fonctionnement dans la reconnaissance d'un terme.

— La règle ANALYSE correspond à l'emploi d'une production contenant l'opérateur $\odot$. Elle peut s'interpréter comme une transition dans un automate classique : le non-terminal en cours de reconnaissance passe de A à B, et la lettre a est lue dans w, avançant la position courante à $i + |a|$ (i restant constant dans le cas où $a = \varepsilon$). Le vecteur v et la position j, servant à la reconnaissance des productions utilisant $\otimes$, sont inchangés. Sur la figure 5.13, cette règle est employée dans les sous-termes marqués $\odot$ de t, par exemple pour reconnaître le premier facteur w_1 du mot w analysé.

— La règle HYPOTHÈSE devine l'emploi d'une production commutative réécrivant B en $\otimes\ a\ C$. Cette règle permet de descendre dans l'arbre de

dérivation de t, mais ne confirme pas immédiatement la présence de la lettre a dans w : celle-ci peut en effet se trouver à une position arbitraire entre i et j dans w, par ré-association et commutation de $\otimes$ dans t'. Le fait de dérivation résultant accumule cependant une occurrence de la lettre a dans le vecteur v, et met à jour le prochain non-terminal à réécrire de B en C. Observons enfin que l'usage de cette règle est restreint par la condition $\|v\| \leq |w|$, qui limite la taille de v au nombre de lettres dans w (par abus de notation, nous considérons $\vec{1}_\varepsilon = \vec{0}$). Appliquée à la figure 5.13, cette règle permet d'accumuler dans le vecteur v les lettres du facteur w_2.

— La règle Confirmation vérifie le bon emploi d'une série d'occurrences de la règle Hypothèse. Cette règle compte les lettres de w de i à i' et de j' à j, et vérifie que le résultat correspond au vecteur v accumulé par des productions successives. Sur la figure 5.13, l'emploi de cette règle correspond à la transition du terme marqué $\otimes$ à son sous-terme marqué $\odot$: le mot w'_2 est le fragment de w compris entre i et i', et le mot w''_2 est celui allant de j' à j. Ces mots doivent seulement respecter l'invariant $\psi(w'_2) + \psi(w''_2) = \psi(w_2)$; les occurrences de $\otimes$ qui dominent w_2 permettant d'en réorganiser les lettres dans n'importe quel ordre (sans toutefois modifier la structure du sous-terme $\odot$ qui domine w_3).

— La règle Réussite termine la lecture du mot w avec une production terminale, en vérifiant la validité d'un éventuel vecteur v accumulé. Cette règle effectue la même vérification que Confirmation lorsque le dernier opérateur visité est $\otimes$, ou vérifie simplement que la dernière lettre à analyser dans w est bien a dans le cas contraire. Dans tous les cas, elle vérifie l'ensemble des lettres restantes dans w (entre i et j), et termine ainsi l'analyse du mot.

Théorème 5.3. *L'algorithme décrit par la figure 5.12 est correct et complet, et s'exécute de manière non-déterministe en espace logarithmique (*NL*).*

Démonstration. La preuve détaillée de ce résultat s'appuie sur l'explication ci-dessus, en montrant que tout composant d'un fait de dérivation est représentable en espace logarithmique par rapport à $|w|$ (ou au cardinal de $\mathcal{N}$) ; elle figure en annexe dans la section B.2. □

5.5.3 Algorithme d'analyse pour les CCFG

De même que pour les CRG, nous présentons maintenant l'algorithme d'analyse pour les CCFG ; celui-ci est donné par la figure 5.14.

Les faits de dérivation de cet algorithme sont de la forme (v, i, j), se composant d'un vecteur de symboles non-terminaux $v \in \mathbb{N}^{\mathcal{N}}$ et de deux positions dans w : $i, j \in [|w|]$. Un fait (v, i, j) dénote que les symboles non-terminaux

$$\frac{A(l_i) \leftarrow \in \mathcal{P}}{(\vec{1}_A, i-1, i)}\ \textsc{Constante} \qquad \frac{\exists t \in \mathcal{T}_G(A).\, \mathrm{frt}(t) = \varepsilon}{(\vec{1}_A, i, i)}\ \textsc{Vide}$$

$$\frac{(\vec{1}_B, i, j) \quad (\vec{1}_C, j, k) \quad A(\odot\ x\ y) \leftarrow B(x)\ C(y) \in \mathcal{P}}{(\vec{1}_A, i, k)}\ \textsc{Analyse}$$

$$\frac{(u, i, j) \quad (v, j, k) \quad \|u+v\| \leq |w|+1}{(u+v, i, k)}\ \textsc{Combinaison}$$

$$\frac{(v, i, j) \quad v \in \psi(G, A)}{(\vec{1}_A, i, j)}\ \textsc{Réduction}$$

FIGURE 5.14 – Algorithme d'analyse selon une CCFG

contenus dans le vecteur v permettent, dans un certain ordre, de dériver le facteur $l_{i+1} \dots l_j$ de w ; l'algorithme procède de manière ascendante dans l'arbre de dérivation, en construisant des faits successifs décrivant des fragments plus grands de w. Un mot w est reconnu par une CCFG $G = (\mathcal{N}, \Sigma, S, \mathcal{P})$ si et seulement si l'algorithme permet de dériver le fait $(\vec{1}_S, 0, |w|)$. Observons que, suite à un emploi de l'opérateur $\odot$ qui ne permet pas de réordonner ses éléments, le vecteur $v = \vec{1}_A$ contient toujours un unique symbole non-terminal.

Nous détaillons maintenant les différentes règles de l'algorithme :

— La règle CONSTANTE interprète la i^{e} lettre de w selon une production terminale $A(l_i)$. Elle conclut que le fragment de w correspondant est dérivable à partir de A.

— La règle VIDE élimine un sous-terme trivial (ayant pour frontière le mot vide) dérivé à partir de A. Elle permet de raccourcir une dérivation en éliminant en une étape les sous-termes qui ne font pas progresser la reconnaissance de w.

— La règle ANALYSE regroupe deux non-terminaux B et C pouvant se réécrire en deux facteurs contigus de w (de i à j et de j à k respectivement) à l'aide d'une production de $\mathcal{P}$ permettant de réécrire un A comme la concaténation de B et C. Elle en déduit que le facteur de w allant de i à k peut s'interpréter comme un A.

— La règle COMBINAISON assemble deux vecteurs dont le contenu permet de dériver deux facteurs contigus de w, sans tenir compte de l'ordre de leurs éléments. Elle correspond à un emploi de l'opérateur $\otimes$ pour regrouper deux sous-termes également formés avec $\otimes$, et dont les éléments peuvent donc être librement permutés. La borne $\|u+v\| \leq |w|+1$ limite le nombre de symboles non-terminaux utilisés pour former w : chacun d'entre eux doit correspondre à au moins une lettre de w, à l'exception d'au plus un symbole produit par la règle VIDE. Observons que cette

règle ne vérifie pas que G permet de recombiner les éléments de u et de v à l'aide de l'opérateur $\otimes$.

— La règle RÉDUCTION vérifie le bon emploi d'une série d'occurrences de la règle COMBINAISON vis-à-vis de la grammaire G. Elle accepte le contenu du vecteur v comme les éléments (librement recombinables) dérivés à partir d'une occurrence de A, en s'appuyant sur le langage commutatif $\psi(G, A)$ de A selon G, qui est semilinéaire.

Théorème 5.4. *L'algorithme donné par la figure 5.14 est correct et complet, et s'exécute en* LOGCFL.

Démonstration. La preuve de la complexité de cet algorithme s'appuie sur un résultat de Ruzzo [1980] caractérisant la classe de complexité LOGCFL ; son détail est donné en annexe dans la section B.2. □

5.6 Algorithme général

Nous souhaitons maintenant proposer un algorithme de reconnaissance plus général que ceux de la section précédente. L'objectif est de pouvoir traiter les CMCFG et le problème de la reconnaissance universelle, tout en épousant les bornes de difficultés exhibées dans la section 5.4. Nous traiterons également le cas des grammaires dites effaçantes, qui permettent la non-utilisation de certains éléments associés au membre droit d'une production.

Le principal obstacle à la description d'un tel algorithme est la taille minimale d'une dérivation produisant un terme qui reconnaisse un mot w, comme l'illustre la figure 5.15. Une telle grammaire reconnaît simplement le langage $\mathcal{L}(G) = \{\varepsilon\}$, mais dérive des termes de taille exponentielle par rapport au nombre de symboles non-terminaux dans $\mathcal{N}$. Plus généralement, une CMCFG peut dériver des tuples de termes et/ou de contextes « triviaux », c'est à dire qui ne contribuent pas réellement à la formation du mot w à reconnaître. Ce problème ne peut pas être résolu en supprimant simplement les symboles ε de la grammaire G : comme illustré précédemment dans la figure 5.5, la présence de feuilles ε est susceptible d'altérer le langage d'un terme, lorsqu'elles apparaissent sous un opérateur $\odot$ situé entre deux opérateurs $\otimes$.

$$G = (\{A_0 \dots A_n\}, \Sigma, A_0, \mathcal{P})$$

$$\mathcal{P} = \left\{ A_i \left(\begin{array}{c} \otimes \\ / \quad \backslash \\ A_{i+1} \quad A_{i+1} \end{array} \right) \mid 0 \leq i < n \right\} \cup \{A_n(\varepsilon)\}$$

FIGURE 5.15 – Exemple de CCFG dérivant des termes de grande taille

Toutefois cette configuration, qui rompt l'associativité de l'opérateur commutatif $\otimes$, est la seule qui pose problème : l'ajout d'un contexte de la forme $\otimes(\varepsilon, x)$ n'a pas d'impact sur le langage d'un terme, pas plus que l'emploi de plusieurs concaténations successives du symbole ε. Nous proposons par conséquent dans ce chapitre un mécanisme de compression des termes et contextes sur un alphabet commutatif $\mathrm{Comm}(\Sigma)$, qui permet d'éliminer exactement les fragments d'un terme qui n'ont pas d'influence sur son langage associé. Ce mécanisme nous permettra d'analyser un mot selon une grammaire modulo compression du terme intermédiaire, nous permettant d'atteindre exactement les bornes de complexité données dans la section 5.4.

5.6.1 Typage sémantique des termes

Nous proposons un système de typage sémantique pour les lambda-termes représentant des termes et contextes commutatifs. Les types sémantiques atomiques que nous utilisons correspondent à des termes clos, de type syntaxique 0. Nous décrivons les types suivants :

- Le type *epsilon*, $\mathcal{E}$, est associé aux termes dont la frontière se réduit à ε. Ces termes pourront par la suite se voir substituer la constante ε.
- Le type *rigide*, $\mathcal{R}$, est associé aux termes se réduisant à une lettre, ou dont la racine est l'opérateur $\odot$. Intuitivement, ces termes forment un « bloc » solide dont les éléments ne peuvent pas se mélanger avec d'autres issus d'un contexte plus large.
- Le type *commutatif*, $\mathcal{C}$, est associé aux termes dont la racine est l'opérateur $\otimes$. Intuitivement, le langage d'un tel terme dépend de son contexte : des lettres supplémentaires peuvent par exemple s'intercaler entre plusieurs de ses sous-termes par associativité et commutativité de $\otimes$.
- Le type *vide*, $\bot$, est associé aux termes destinés à être ultérieurement supprimés lors de la dérivation. Ce type servira à raccourcir les dérivations dans le cas des grammaires effaçantes.

Familles de types Nous considérons ensuite quatre familles de types linéaires (V_α, E_α, T_α et U_α), permettant de typer des contextes d'arité quelconque. Ces familles sont définies de manière mutuellement récursive comme suit :

$$\begin{array}{lcllcl} V_0 & = & \{\bot\} & V_{\alpha\to\beta} & = & V_\alpha \to V_\beta \\ E_0 & = & \{\mathcal{E}\} & E_{\alpha\to\beta} & = & E_\alpha \to E_\beta \cup V_\alpha \to E_\beta \\ T_0 & = & \{\mathcal{R},\mathcal{C}\} & T_{\alpha\to\beta} & = & U_\alpha \to T_\beta \cup V_\alpha \to T_\beta \\ & & & U_\alpha & = & T_\alpha \cup E_\alpha \end{array}$$

Nous omettons dans la suite le type syntaxique en indice de V, E, T ou U lorsqu'il est quelconque. Les lambda-termes typés par V dénotent des termes ou contextes vides, qui seront effacés durant la dérivation. La famille E correspond à des contextes dont le type résultant est $E_0 = \mathcal{E}$, autrement dit des termes

dont la frontière est ultimement ε. La famille de types auxiliaire T recouvre des lambda-termes qui dénotent ultimement des termes commutatifs dont la frontière n'est pas vide. Enfin, la famille U recouvre l'ensemble des termes ou contextes utiles à une dérivation, c'est à dire les termes de T ou E.

Conventions de notation Par convention, nous utiliserons une police particulière pour les variables $\mathbf{a}, \mathbf{b}, \dots$ qui dénotent des types sémantiques, et emploierons librement dans les prochaines sous-sections le mot « terme » pour désigner des lambda-termes représentant des termes ou des contextes sur une algèbre commutative $\mathrm{Comm}(\Sigma)$. Nous appelons *environnement de typage* une fonction partielle (notée $\Gamma, \Delta, \dots$) associant à un ensemble de lambda-variables $\mathcal{X}$ des types sémantiques de U, et respectant leur type syntaxique en ce sens que $\Gamma(x^\alpha) \in U_\alpha$. Nous abrégeons par simplicité $x \in \mathrm{Dom}(\Gamma)$ en $x \in \Gamma$. Si deux environnements de typage Γ et Δ sont tels que $\mathrm{Dom}(\Gamma) \cap \mathrm{Dom}(\Delta) = \emptyset$, nous écrivons Γ, Δ pour dénoter leur union. Enfin, étant donné une constante c, nous écrirons $\tau(c)$ pour dénoter l'ensemble des types sémantiques de c.

Type des constantes Étant donné une algèbre commutative $\mathrm{Comm}(\Sigma)$ représentée par des lambda-termes, nous donnons aux constantes de l'ensemble $C = \{\odot, \otimes, \varepsilon\} \cup \Sigma$ les types sémantiques suivants :

- La constante ε est de type $\mathcal{E}$.
- Pour tout $a \in \Sigma$, a est de type $\mathcal{R}$.
- L'opérateur $\odot$ est dénoté par une constante pouvant avoir les types suivants : $\mathcal{E} \to \mathcal{E} \to \mathcal{E}$ ou $\mathbf{a} \to \mathbf{b} \to \mathcal{R}$ si $\mathbf{a}$ ou $\mathbf{b}$ est dans T_0.
- L'opérateur $\otimes$ est dénoté par une constante pouvant avoir les types suivants : $\mathbf{a} \to \mathcal{E} \to \mathbf{a}$ ou $\mathcal{E} \to \mathbf{a} \to \mathbf{a}$ si $\mathbf{a} \in U_0$, ou bien $\mathbf{a} \to \mathbf{b} \to \mathcal{C}$ si $\mathbf{a}, \mathbf{b} \in T_0$.
- Pour tout type syntaxique α, la constante supplémentaire Ω^α a le type sémantique V_α.

Nous complétons l'ensemble C des constantes issues de $\mathrm{Comm}(\Sigma)$ par un ensemble de constantes Ω^α représentant un lambda-terme arbitraire destiné à être effacé lors d'une dérivation dans le cadre d'une grammaire effaçante.

Jugements et dérivations de typage Un *jugement de typage* est un triplet $\Gamma \vdash M : \mathbf{a}$ où Γ est un environnement de typage, M est un lambda-terme *affine* (contentant au plus une occurrence de chaque variable) et $\mathbf{a}$ est un type sémantique appartenant à U. Le *sujet* d'un jugement de typage est le lambda-terme M auquel il attribue un type. Un jugement est dit *bien formé* lorsque :

1. Si $x \notin \mathrm{FV}(M)$ et $x \in \Gamma$, alors $\Gamma(x) \in E$.
2. Si $\mathbf{a} \in E$, alors pour tout $x \in \Gamma$, le type $\Gamma(x)$ appartient à E.

La figure 5.16 décrit l'ensemble des règles permettant de dériver des jugements de typage. Nous notons $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ l'existence d'une dérivation

$$\frac{}{\Gamma, x : \mathbf{a} \vdash x : \mathbf{a}}\, var \qquad \frac{c \in C \quad \mathbf{a} \in \tau(c)}{\Gamma \vdash M : \mathbf{a}}\, cst$$

$$\frac{\Gamma, x : \mathbf{a} \vdash M : \mathbf{b}}{\Gamma \vdash \lambda x.M : \mathbf{a} \to \mathbf{b}}\, abs \qquad \frac{\Gamma \vdash M : \mathbf{a} \to \mathbf{b} \quad \Delta \vdash N : \mathbf{a}}{\Gamma \vdash MN : \mathbf{b}}\, app$$

$$\frac{\Gamma \vdash M : \mathbf{b} \quad x \notin \Gamma}{\Gamma \vdash \lambda x.M : V_\alpha \to \mathbf{b}}\, abs, \bot \qquad \frac{\Gamma \vdash M : V_\alpha \to \mathbf{b}}{\Gamma \vdash MN^\alpha : \mathbf{b}}\, app, \bot$$

FIGURE 5.16 – Règles de dérivation pour les types sémantiques

complète $\mathtt{M}$ employant ce système de règles et dont la conclusion est $\Gamma \vdash M : \mathbf{a}$. Le *type* d'une dérivation est, par extension, celui du jugement qui la conclut ; et une dérivation est dite *close* lorsque l'environnement de typage Γ de sa conclusion est vide.

Propriétés Notre système de typage sémantique possède plusieurs propriétés essentielles : renforcement, affaiblissement, réduction et expansion (affine) du sujet. Les preuves détaillées de ces propriétés sont données en annexe (section B.3.1), incluant un lemme de substitution et un lemme d'extraction.

Propriété 5.5 (Renforcement)**.** *Si* $\Gamma, x : \mathbf{a} \vdash M : \mathbf{b}$ *est dérivable et que* x *n'est pas libre dans* M*, alors* $\Gamma \vdash M : \mathbf{b}$ *est dérivable (lemme B.7).*

Propriété 5.6 (Affaiblissement)**.** *Si* $\Gamma \vdash M : \mathbf{b}$ *est dérivable et que le jugement* $\Gamma, \Delta \vdash M : \mathbf{b}$ *est bien formé, alors* $\Gamma, \Delta \vdash M : \mathbf{b}$ *est dérivable (lemme B.8).*

Propriété 5.7 (Réduction du sujet)**.** *Étant donné un terme affine* M *tel que que* $\Gamma \vdash M : \mathbf{b}$ *est dérivable, si* $M \xrightarrow{*}_{\beta\eta} N$ *alors* $\Gamma \vdash N : \mathbf{b}$ *est dérivable (lemme B.10).*

Propriété 5.8 (Expansion affine du sujet)**.** *Étant donné deux termes affines* M *et* N *tels que* $M \xrightarrow{*}_{\beta\eta} N$ *et un environnement de typage* Γ *incluant les variables libres de* M*, s'il existe une dérivation* $\mathtt{N} :: \Gamma \vdash N : \mathbf{a}$*, alors il existe une dérivation* $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ *telle que* $\mathtt{M} \xrightarrow{*}_{\beta\eta} \mathtt{N}$ *(lemme B.12).*

Étant donné une dérivation $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ et un terme N tel que $M \xrightarrow{*}_{\beta\eta} N$, si $\mathtt{N} :: \Gamma \vdash N : \mathbf{a}$ est la dérivation obtenue par réduction du sujet M, nous écrirons directement dans la suite que $\mathtt{M} \xrightarrow{*}_{\beta\eta} \mathtt{N}$.

5.6.2 Le système de réécriture $\to_\varepsilon$

Nous introduisons maintenant une relation de réécriture $\to_\varepsilon$ sur les dérivations de types sémantiques, dont les règles sont données par la figure 5.17 ;

nous désignerons ces règles en les numérotant de 1 à 8 (de haut en bas) dans la suite du texte. Ce système de réécriture permet de compresser les termes et contextes apparaissant dans les dérivations d'une CMCFG sans altérer le langage du terme final : les opérations introduisant des occurrences inutiles de ε sont éliminées (par les paires de règles 3,4 et 6,7), et les termes et contextes dont le système de typage garantit la réduction à ε sont réduits à leur plus simple expression (par les règles 1,5 et 8) ; enfin, les termes et contextes typés par V, destinés à être effacés par β-réduction, sont remplacés par une constante Ω de taille fixe (règle 2). Observons que la relation $\to_\varepsilon$ ne modifie que le sujet d'un jugement de typage : son environnement de typage et le type sémantique qui lui sont attribués demeurent constants.

Propriétés de $\to_\varepsilon$ et $\to_{\beta\varepsilon}$ Le système de réécriture $\to_\varepsilon$ est confluent et fortement normalisant, induisant l'existence d'une forme normale unique. En outre, le sujet M de la conclusion d'une dérivation en forme ε-normale est affine, et toutes ses variables libres appartiennent au domaine de son environnement de typage Γ ; ces deux dernières propriétés servent uniquement à démontrer d'autres résultats intermédiaires, et sont établies en annexe par les lemmes B.15 et B.14.

Propriété 5.9 (Forme ε-normale)**.** *Le système de réécriture $\to_\varepsilon$ induit pour toute dérivation une unique forme normale (lemme B.13).*

Si $\mathtt{D} :: \Gamma \vdash M : \mathbf{a}$ est une dérivation sémantique de sujet M, sa forme ε-normale est notée $|\mathtt{D}|_\varepsilon$; par extension le sujet de cette dernière dérivation est dit en forme ε-normale et noté $|M|_\varepsilon$.

Nous nous intéressons ensuite à l'interaction entre ε et β-réduction. En utilisant l'extension de la notion de β-réduction aux dérivations de typage (conséquence de la propriété de réduction du sujet), nous pouvons montrer la normalisation forte et la confluence du système joint $\to_{\beta\varepsilon}$. Une autre propriété de ce système, cruciale dans le fonctionnement de notre algorithme d'analyse modulo compression, est la possibilité de réordonner deux réductions successives $\to_\beta$ et $\to_\varepsilon$: aucun ε-redex n'est créé par β-contraction, entraînant que la β-réduction préserve l'ε-normalité (en d'autres termes, un lambda-terme déjà compressé demeure compact lors de son évaluation par $\to_\beta$).

Propriété 5.10 (Réordonnement de $\to_\beta/\to_\varepsilon$)**.** *Considérons les deux termes affines M et N et leurs dérivations associées $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ et $\mathtt{N} :: \Gamma \vdash N : \mathbf{a}$, tels que $\mathtt{M} \to_\beta \mathtt{N}$ selon la procédure du lemme B.10. Si $\mathtt{N} \to_\varepsilon \mathtt{N}' :: \Gamma \vdash N' : \mathbf{a}$, alors il existe un terme M' muni d'une dérivation $\mathtt{M}' :: \Gamma \vdash M' : \mathbf{a}$ de sorte que $\mathtt{M} \to_\varepsilon \mathtt{M}'$ et $\mathtt{M}' \overset{*}{\to}_\beta \mathtt{N}'$ (lemme B.16). Cette relation est illustrée par le diagramme*

$$\frac{\mathtt{M}\,::}{\Gamma \vdash M : \mathcal{E}} \longrightarrow_{\varepsilon} \frac{}{\Gamma \vdash \varepsilon : \mathcal{E}}\, cst \qquad (M \neq \varepsilon) \tag{1}$$

$$\frac{\dfrac{\mathtt{M}\,::}{\Gamma \vdash M : V_\alpha \to \mathbf{b}}}{\Gamma \vdash MN^\alpha : \mathbf{b}}\, app, \bot \longrightarrow_{\varepsilon} \frac{\dfrac{\mathtt{M}\,::}{\Gamma \vdash M : V_\alpha \to \mathbf{b}}}{\Gamma \vdash M\Omega^\alpha : \mathbf{b}}\, app, \bot \qquad (N^\alpha \neq \Omega^\alpha) \tag{2}$$

$$\frac{}{\Gamma \vdash \odot : \mathcal{R} \to \mathcal{E} \to \mathcal{R}}\, cst \longrightarrow_{\varepsilon} \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{R}, y : \mathcal{E} \vdash x : \mathcal{R}}\, var}{\Gamma, x : \mathcal{R} \vdash \lambda y.x : \mathcal{E} \to \mathcal{R}}\, abs}{\Gamma \vdash \lambda xy.x : \mathcal{R} \to \mathcal{E} \to \mathcal{R}}\, abs \tag{3}$$

$$\frac{}{\Gamma \vdash \odot : \mathcal{E} \to \mathcal{R} \to \mathcal{R}}\, cst \longrightarrow_{\varepsilon} \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{E}, y : \mathcal{R} \vdash y : \mathcal{R}}\, var}{\Gamma, x : \mathcal{E} \vdash \lambda y.y : \mathcal{R} \to \mathcal{R}}\, abs}{\Gamma \vdash \lambda xy.y : \mathcal{E} \to \mathcal{R} \to \mathcal{R}}\, abs \tag{4}$$

$$\frac{}{\Gamma \vdash \odot : \mathcal{E} \to \mathcal{E} \to \mathcal{E}}\, cst \longrightarrow_{\varepsilon} \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{E}, y : \mathcal{E} \vdash \varepsilon : \mathcal{E}}\, cst}{\Gamma, x : \mathcal{E} \vdash \lambda y.\varepsilon : \mathcal{E} \to \mathcal{E}}\, abs}{\Gamma \vdash \lambda xy.\varepsilon : \mathcal{E} \to \mathcal{E} \to \mathcal{E}}\, abs \tag{5}$$

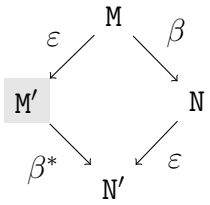
$$\frac{}{\Gamma \vdash \otimes : \mathbf{a} \to \mathcal{E} \to \mathbf{a}}\, cst \longrightarrow_{\varepsilon} \frac{\dfrac{\dfrac{}{\Gamma, x : \mathbf{a}, y : \mathcal{E} \vdash x : \mathbf{a}}\, var}{\Gamma, x : \mathbf{a} \vdash \lambda y.x : \mathcal{E} \to \mathbf{a}}\, abs}{\Gamma \vdash \lambda xy.x : \mathbf{a} \to \mathcal{E} \to \mathbf{a}}\, abs \qquad (\mathbf{a} \neq \mathcal{E}) \tag{6}$$

$$\frac{}{\Gamma \vdash \otimes : \mathcal{E} \to \mathbf{a} \to \mathbf{a}}\, cst \longrightarrow_{\varepsilon} \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{E}, y : \mathbf{a} \vdash y : \mathbf{a}}\, var}{\Gamma, x : \mathcal{E} \vdash \lambda y.y : \mathbf{a} \to \mathbf{a}}\, abs}{\Gamma \vdash \lambda xy.y : \mathcal{E} \to \mathbf{a} \to \mathbf{a}}\, abs \qquad (\mathbf{a} \neq \mathcal{E}) \tag{7}$$

$$\frac{}{\Gamma \vdash \otimes : \mathcal{E} \to \mathcal{E} \to \mathcal{E}}\, cst \longrightarrow_{\varepsilon} \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{E}, y : \mathcal{E} \vdash \varepsilon : \mathcal{E}}\, cst}{\Gamma, x : \mathcal{E} \vdash \lambda y.\varepsilon : \mathcal{E} \to \mathcal{E}}\, abs}{\Gamma \vdash \lambda xy.\varepsilon : \mathcal{E} \to \mathcal{E} \to \mathcal{E}}\, abs \tag{8}$$

FIGURE 5.17 – Le système de réécriture $\to_\varepsilon$

en diamant suivant :



Propriété 5.11 (Forme $\beta\varepsilon$-normale)**.** *Le système $\to_{\beta\varepsilon}$ est confluent et induit une forme normale unique (lemme B.19).*

Comme pour le système $\to_\varepsilon$, nous notons respectivement $|\mathtt{D}|_{\beta\varepsilon}$ et $|M|_{\beta\varepsilon}$ la forme $\beta\varepsilon$-normale d'une dérivation $\mathtt{D}$ et d'un terme M possédant un type sémantique.

Contraintes sur la taille des termes Nous établissons maintenant plusieurs contraintes sur la taille des termes en forme ε-normale, dans le but de montrer ultérieurement qu'un terme (ou contexte) commutatif peut, modulo ε-conversion, être représenté en espace polynomial relativement à la longueur des mots de son langage. Ces résultats nous serviront à établir la complexité de l'algorithme d'analyse général à l'issue de ce chapitre.

Nous définissons d'abord plusieurs mesures quantifiant la taille d'un terme et celle de son type syntaxique :

Définition 5.8. La *taille* $|M|$ d'un lambda-terme M est définie inductivement comme suit :
- $|x| = 1$
- $|c| = 1$ pour toute constante $c \in C$
- $|\lambda x.P| = |P|$
- $|MN| = |M| + |N|$

La *taille concrète* $\|M\|$ d'un lambda-terme M représentant un terme commutatif est définie inductivement comme suit :
- $\|x\| = 0$
- $\|c\| = \begin{cases} 3 \text{ si } c \in \{\odot, \otimes\} \\ 1 \text{ si } c \in \Sigma \\ 0 \text{ si } c \in \{\varepsilon, \Omega^\alpha\} \end{cases}$
- $\|\lambda x.P\| = \|P\|$
- $\|MN\| = \|M\| + \|N\|$

La *taille* $|\tau|$ d'un type syntaxique τ est définie inductivement comme suit :
- $|0| = 1$
- $|\alpha \to \beta| = |\alpha| + |\beta|$

Tout lambda-terme M peut aisément être représenté en espace polynomial par rapport à sa taille $|M|$, et sa taille concrète $\|M\|$, lorsqu'il est compressé, est proportionnelle à la longueur des mots qu'il reconnaît.

Nous établissons maintenant une propriété notable des termes en forme $\beta\varepsilon$-normale, qui est que leur taille est bornée linéairement par leur taille concrète et par la taille de leur type : ainsi, l'espace requis pour représenter un terme compressé est borné par longueur des mots de son langage. Une autre propriété utile à notre algorithme de reconnaissance général est que la substitution et la β-réduction de termes ε-normaux préserve leur taille concrète. Cette dernière propriété entrera en jeu lors de l'application d'une production : la somme des tailles des termes compressés associés aux non-terminaux résultants sera toujours inférieure ou égale à celle des termes associés au non-terminal de départ.

Propriété 5.12. *Si* $\mathtt{M}$ *est une dérivation en forme* $\beta\varepsilon$*-normale typant un terme clos* M *de type* α*, alors* $|M| \leq \|M\| + |\alpha|$ *(corollaire B.24).*

Propriété 5.13. *Soit* $M, N, N_1, \ldots, N_n$ *des termes respectant ces conditions :*
- $\mathrm{FV}(M) = \{x_1 \ldots x_n\}$
- $M[N_i/x_i] \xrightarrow{*}_\beta N$
- *Il existe des dérivation* ε*-normales bien formées des jugements suivants :*

$$\Gamma, x_1 : \mathbf{a_1} \ldots x_n : \mathbf{a_n} \vdash M : \mathbf{b} \qquad \vdash N : \mathbf{b} \qquad \vdash N_1 : \mathbf{a_1} \;\ldots\; \vdash N_n : \mathbf{a_n}$$

L'égalité suivante est respectée (lemme B.25) :

$$\sum_{i=1}^{n} \|N_i\| = \|N\| - \|M\|$$

5.6.3 Algorithmes d'analyse NP

Le système de réécriture $\to_\varepsilon$ et ses propriétés induisent une relation d'équivalence sur nos termes commutatifs : celle-ci garantit pour tout terme l'existence d'une forme normale de taille polynomiale par rapport à la longueur des mots de son langage, et supprime les sous-termes dont l'élimination par β-réduction est assurée. De plus, le langage d'un terme t est le même que celui de sa forme normale $|t|_\varepsilon$ (par abus de notation, nous étendons la relation $\to_\varepsilon$ sur les lambda-termes aux termes commutatifs qu'ils dénotent). Par suite, étant donné une grammaire commutative G et un mot w, des procédures construisant la forme ε-normale $|t|_\varepsilon$ d'un terme t tel que $t \in \mathcal{T}(G)$ et $w \in \mathcal{L}(t)$ permettent d'établir les résultats de complexité attendus.

Analyse universelle des CCFG L'analyse universelle d'un mot selon une CCFG (et, par conséquent, selon une CRG) peut s'effectuer en temps polynomial sur une machine non-déterministe. Soit $G = (\Sigma, \mathcal{N}, S, \mathcal{P})$ une CCFG, que

Classe	Analyse fixée	Analyse universelle	
		Non-effaçante	Effaçante
Terme	$\mathcal{O}(1)$	NP-C	
CRG	NL	NP	
CCFG	LogCFL-C	NP	
CMG	NP	PSPACE-D	EXPTIME
CMREG	NP	PSPACE-C	EXPTIME-C
CMCFG	NP	PSPACE-D	EXPTIME-C

Table 5.6 – Résultats de complexité de la sous-section 5.6.3

l'on suppose en forme normale (voir page 133), et w un mot de Σ^* à analyser selon G, nous considérons le raisonnement suivant :

1. Tout non-terminal de G dans une dérivation est associé à un unique terme, dont le type syntaxique est 0. Le système de typage sémantique décrit précédemment permet alors de raffiner les types rattachés aux non-terminaux, et de produire une grammaire G', produisant le même langage de termes $\mathcal{T}(G') = \mathcal{T}(G)$, mais dans laquelle le type sémantique des termes produits par chaque symbole non-terminal (à l'exception de l'axiome) est connu. Plus précisément, nous construisons $G' = (\Sigma, \mathcal{N}', \sigma, \mathcal{P}')$ ainsi :
   — L'axiome σ est un symbole non-terminal nouvellement introduit.
   — L'ensemble $\mathcal{N}'$ des non-terminaux de G' contient σ, et est formé pour le reste en associant aux symboles de $\mathcal{N}$ un type sémantique : $\mathcal{N}' = (\mathcal{N} \times \{\mathcal{R}, \mathcal{C}, \mathcal{E}\}) \cup \{\sigma\}$. Nous noterons le type sémantique de ces symboles en exposant (*e.g.* $A^{\mathcal{R}}$ lorsque $A \in \mathcal{N}$).
   — Les productions de $\mathcal{P}'$ étendent les productions de $\mathcal{P}$ en respectant le typage sémantique introduit par les non-terminaux de $\mathcal{N}'$. Ainsi, si $A(\odot\ x\ y) \leftarrow B(x)\ C(y)$ est une production de $\mathcal{P}$, alors $\mathcal{P}'$ contient toutes les productions de la forme $A^{\alpha}(\odot\ x\ y) \leftarrow B^{\beta}(x)\ C^{\gamma}(y)$ pour lesquelles $\beta \to \gamma \to \alpha$ est un type sémantique possible de la constante $\odot$ (voir page 140). De même, les productions terminales de $\mathcal{P}'$ sont de la forme $A^{\mathcal{E}}(\varepsilon) \leftarrow$ ou $A^{\mathcal{R}}(a) \leftarrow$. Finalement, $\mathcal{P}'$ contient trois productions de la forme $\sigma(x) \leftarrow S^{\alpha}(x)$, avec $\alpha \in \{\mathcal{R}, \mathcal{C}, \mathcal{E}\}$.
2. En appliquant les règles du système de réécriture $\to_{\varepsilon}$ aux productions de G', nous obtenons une grammaire G'' dont le langage de termes contient les formes ε-normales des termes de $\mathcal{T}(G)$. Les non-terminaux correspondant à des termes de type $\bot$ (ceux dont les variables sont absentes du membre gauche après ε-réduction) sont exclus du membre droit de la production. Aussi, la grammaire résultante n'est pas généralement en forme normale : ainsi, une production $A^{\mathcal{R}}(\odot\ x\ y) \leftarrow B^{\mathcal{R}}(x)\ C^{\mathcal{E}}(y)$ présente dans G' est remplacée par $A^{\mathcal{R}}(x) \leftarrow B^{\mathcal{R}}(x)$ dans G''. De plus,

puisque le système de réécriture $\to_\varepsilon$ préserve le langage des termes commutatifs qu'il réécrit, et que tout terme reconnu par G'' correspond, par ε-réduction, à un terme de G, les langages de mots de ces deux grammaires sont identiques.

3. Si $w \in \mathcal{L}(G)$, il existe donc un terme $t \in \mathcal{T}(G'')$ tel que $w \in \mathcal{L}(t)$. Construire et analyser un tel terme selon G'' permet donc de décider si $w \in \mathcal{L}(G)$.

Les étapes 1 et 2, consistant à construire successivement G' et G'', peuvent être effectuées en temps linéaire par rapport à la taille de G. La grammaire G'' peut, au besoin, être mise en forme normale (notamment en supprimant les règles unitaires) en temps polynomial. Le terme t de l'étape 3 étant ε-normal, sa taille est polynomiale en $|w|$ par la propriété 5.12, et il peut donc être construit en temps non-déterministe polynomial. Par suite, la reconnaissance de t par G'' revient à l'analyse d'un terme par une grammaire régulière de termes, laquelle s'effectue en temps polynomial. Par conséquent le problème de l'analyse universelle selon une CCFG (et, par extension, selon une CRG) est décidable et appartient à NP.

Analyse fixée des CMCFG Nous considérons maintenant la complexité de l'analyse pour une CMCFG fixée. Comme dans le cas précédent, celle-ci est obtenue en calculant une grammaire équivalente à G reconnaissant les formes ε-normales des termes de $\mathcal{T}(G)$, puis en devinant un terme $|t|_\varepsilon$ dont la taille est polynomiale par rapport à $|w|$. À la différence des CCFG, les types des non-terminaux d'une CMCFG ne permettent pas de garantir que la taille de la grammaire résultante est polynomiale par rapport à G : cette procédure n'est donc applicable que si la taille de G est fixée à l'avance. L'analyse universelle, abordée dans la prochaine sous-section, requiert de compresser dynamiquement le terme de $\mathcal{T}(G)$ qui reconnaît w.

Comme pour les CCFG, nous construisons une grammaire G' où les non-terminaux sont étiquetés par des types (ou des tuples de types) sémantiques ; observons que le nombre de types sémantiques possibles pour un non-terminal donné est exponentiel dans la taille de son type syntaxique. Nous normalisons ensuite cette grammaire de la même manière que les CCFG précédemment. La taille de la grammaire étant fixée, ces opérations s'effectuent en temps constant par rapport à $|w|$. Il suffit ensuite de deviner un terme ε-normal t (de taille polynomiale en $|w|$), de l'analyser selon la grammaire nouvellement calculée et de vérifier que $w \in \mathcal{L}(t)$ (décidable en NP) pour décider si $w \in \mathcal{L}(G)$ en NP.

5.6.4 Algorithme d'analyse général

En nous appuyant sur le système de réécriture $\to_\varepsilon$ permettant de compresser des termes commutatifs, ainsi que sur les propriétés établies dans cette

section, nous présentons un algorithme général permettant de déterminer si un tuple de lambda-termes (représentant des termes ou des contextes commutatifs) appartient, modulo ε-compression, au langage d'une grammaire commutative G. Décider si un mot w appartient au langage de G revient alors à deviner un terme $|t|_\varepsilon$ compressé (dont la taille est proportionnelle à $|w|$) tel que $w \in \mathcal{L}(t)$, et à déterminer si t appartient, modulo ε-compression, à $\mathcal{T}(G)$ (la construction de $|t|_\varepsilon$ pouvant se faire en NP).

L'algorithme que nous décrivons traite également le cas des grammaires commutatives effaçantes (mais pas celui des grammaires parallèles) ; son fonctionnement s'inspire de l'algorithme de reconnaissance proposé par Kaji *et al.* [1992] pour les MCFG avec ou sans effacement. Il s'appuie sur une notion de buts, formés par un symbole non-terminal associé à un tuple de termes en forme $\beta\varepsilon$-normale, qui doivent être établis comme solvables pour que l'analyse réussisse. L'algorithme procède en deux phases : premièrement, il calcule en EXPTIME (dans le pire cas) un ensemble de buts triviaux pouvant être éliminés par la grammaire considérée, indépendamment du mot à reconnaître ; deuxièmement, il tente de résoudre en PSPACE un ensemble de buts liés au mot à reconnaître, en les décomposant successivement en sous-buts de taille plus réduite. Durant la seconde phase, l'algorithme peut éliminer immédiatement tout but trivial calculé durant la première phase.

Classe	Analyse fixée	Analyse universelle	
		Non-effaçante	Effaçante
Terme	$\mathcal{O}(1)$	NP-C	
CRG	NL	NP-C	
CCFG	LogCFL-C	NP-C	
CMG	NP-C	PSPACE-D	EXPTIME
CMREG	NP-C	PSPACE	EXPTIME
CMCFG	NP-C	PSPACE-D	EXPTIME

Table 5.7 – Résultats de complexité de la sous-section 5.6.4

Nous introduisons les notations et définitions suivantes : étant donné une dérivation $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$, nous écrivons $\mathcal{L}(\mathtt{M})$ pour l'ensemble des λ-termes M' tels qu'il existe une dérivation $\mathtt{M}' :: \Gamma \vdash M' : \mathbf{a}$ et que $\mathtt{M}' \xrightarrow{*}_{\beta\varepsilon} \mathtt{M}$. En outre :

Définition 5.9. Un *élément* $\mathtt{E}$ d'un but est défini comme étant soit une dérivation $\beta\varepsilon$-normale d'un terme clos, soit un symbole spécial $\bot^\alpha$, où α est un type syntaxique. Dans ce dernier cas, l'élément $\mathtt{E}$ est dit *indéfini*.

Le *type syntaxique* d'un élément est, suivant le cas, soit le type syntaxique du sujet de sa dérivation, soit α ; et par abus de langage, lorsqu'un élément $\mathtt{E}$ est défini, son *type* (sémantique) sera le type $\mathbf{a}$ que cette dernière associe à son sujet, et nous noterons de façon transparente $\mathtt{E} :: \Gamma \vdash M : \mathbf{a}$.

Si A est un symbole non-terminal de type $[\alpha_1, \ldots, \alpha_n]$, alors $A(\mathtt{M_1}, \ldots, \mathtt{M_n})$ est un *but*, dans lequel chaque $\mathtt{M_i}$ est un élément de type syntaxique α_i.

Un but $A(\mathtt{M_1}, \ldots, \mathtt{M_n})$ est dit *solvable* lorsqu'il existe un tuple $(P_1, \ldots, P_n)$ appartenant à $\mathcal{L}(A)$ tel que chaque $\mathtt{M_i}$ est soit $\bot^{\alpha_i}$, soit une dérivation telle que $P_i \in \mathcal{L}(\mathtt{M_i})$. En d'autres termes, les buts solvables représentent, là où ils sont définis, des tuples de termes qui appartiennent au langage de leur symbole non-terminal modulo ε-compression.

Enfin, une dérivation dont le sujet est M est dite *triviale* lorsque $\|M\| = 0$ (M est sans constantes hormis ε), et un but $A(\mathtt{M_1}, \ldots, \mathtt{M_n})$ est dit *trivial* lorsque chaque $\mathtt{M_i}$ est soit $\bot^{\alpha_i}$, soit une dérivation triviale.

Première phase Puisque la taille d'une dérivation dépend linéairement de celle de son sujet, la propriété 5.12 implique que l'ensemble des buts triviaux solvables par une grammaire est de taille exponentielle par rapport à celle-ci (les buts triviaux étant formés par des dérivations dont le sujet a une taille concrète nulle, et dont la taille du type syntaxique est fixée par la grammaire). Nous présentons donc un algorithme de point fixe calculant en EXPTIME l'ensemble des buts triviaux solvables d'une grammaire. Une fois construit, l'appartenance d'un but à cet ensemble peut être testée en temps logarithmique par rapport à la taille de ce dernier, soit en PTIME dans notre cas.

Le pré-calcul des buts triviaux est nécessaire pour garantir que la seconde phase de l'algorithme s'effectue en espace polynomial. La complexité algorithmique de cette première phase peut en outre être considérée indépendamment de celle de la seconde, ce qui nous permettra notamment d'établir que l'analyse universelle d'une CMREG non-effaçante s'effectue en PSPACE (le nombre d'éléments triviaux possibles dont le type syntaxique est de la forme $[0, \ldots, 0]$ étant, pour sa part, polynomial).

Soit $G = (\Sigma, \mathcal{N}, S, \mathcal{P})$ une grammaire commutative ; toute règle r de $\mathcal{P}$ est alors de la forme :

$$r = A(M_1^{\alpha_1}, \ldots, M_n^{\alpha_n}) \leftarrow B_1(x_{1,1}, \ldots, x_{1,n_1}) \ldots B_p(x_{p,1}, \ldots, x_{p,n_p})$$

où le non-terminal A a le type $[\alpha_1, \ldots, \alpha_n]$, et chaque non-terminal B_j pour $j \in [p]$ a le type $[\beta_{j,1}, \ldots, \beta_{j,n_j}]$. Étant donné un ensemble T de buts triviaux (initialement vide), notre algorithme calcule l'ensemble $\mathscr{T}(T, G)$ des buts triviaux que G dérive à partir de T. Cet ensemble est formé par l'union des ensembles $\mathscr{T}(T, r)$, décrivant l'ensemble des buts triviaux dérivables à partir de T pour chaque règle $r \in \mathcal{P}$ de G. L'algorithme que nous décrivons calcule simplement un point fixe en faisant croître l'ensemble T des buts triviaux dérivables par la grammaire G : nous posons $\mathscr{T}_0 = \emptyset$ et $\mathscr{T}_{n+1} = \mathscr{T}_n \cup \mathscr{T}(\mathscr{T}_n, G)$; la séquence $(\mathscr{T}_k)_{k \in \mathbb{N}}$ est stationnaire à partir d'un certain k (puisque l'ensemble des buts triviaux solvables par G est fini), et nous notons $\mathscr{T}(G)$ le point fixe ainsi obtenu.

Considérons une règle r de $\mathcal{P}$ ayant la forme décrite plus haut ; suivant la convention décrite page 119 nous employons dans la suite les variables i, j et k pour itérer sur $[n]$, $[p]$ et $[n_j]$ respectivement. Étant donné un ensemble de buts triviaux T, l'ensemble $\mathscr{T}(T, r)$ des buts triviaux dérivables à partir de T selon r est défini comme suit. Un but trivial $A(\mathtt{D_1}, \ldots, \mathtt{D_n})$ appartient à l'ensemble $\mathscr{T}(T, r)$ si et seulement T contient p buts triviaux $B_j(\mathtt{N'_{j,1}}, \ldots, \mathtt{N'_{j,n_j}})$ et qu'il existe n environnements de typage Γ_i vérifiant les conditions suivantes pour tout j, k :

1. $\Gamma_i(x_{j,k}) = \boldsymbol{b_{j,k}}$ implique que l'élément $\mathtt{N'_{j,k}}$ est une dérivation, conclue par $\mathtt{N'_{j,k}} :: \vdash N'_{j,k} : \boldsymbol{b_{j,k}}$ (cohérence des types)
2. $\mathtt{N'_{j,k}} = \bot^{\beta_{j,k}}$ implique que $x_{j,k}$ n'appartient au domaine d'aucun des Γ_i (non-utilisation des éléments indéfinis)

et que, finalement, les n éléments $\mathtt{D_i}$ de ce but sont tels que :

3. $\mathtt{D_i} = \bot^{\alpha_i}$ implique que le domaine de Γ_i est vide (non-effacement des éléments définis)
4. $\mathtt{D_i} :: \vdash Q_i : \boldsymbol{a_i}$ implique qu'il existe une dérivation $\mathtt{M_i} :: \Gamma_i \vdash M_i : \boldsymbol{a_i}$ telle que, pour tout j, k tel que $\Gamma_i(x_{j,k})$ est défini, $\mathtt{M_i}\left[x_{j,k} \leftarrow \mathtt{N'_{j,k}}\right] \xrightarrow{*}_{\beta\varepsilon} \mathtt{D_i}$ (application de la règle r pour construire les éléments du prochain but)

Notons que la substitution d'une variable par une dérivation dans une dérivation, apparaissant dans la condition 4, correspond à un emploi du lemme de substitution (voir B.9) pour obtenir la dérivation résultante.

Le calcul de $\mathscr{T}(G)$ s'effectue en appliquant itérativement la définition précédente, suivant la procédure décrite plus haut, jusqu'à obtenir un ensemble stable de buts triviaux. Cet ensemble est exactement l'ensemble des buts triviaux solvables par la grammaire G, la preuve de cette affirmation figurant en annexe, dans le théorème B.26. En outre, le calcul de $\mathscr{T}(G)$ s'effectue en temps exponentiel dans le cas général, et polynomial dans le cas d'une CMREG non-effaçante, ce que montre le théorème B.28.

Seconde phase Nous présentons maintenant un algorithme descendant qui, étant donné un ensemble de buts, détermine en PSPACE si cet ensemble peut être effacé en appliquant itérativement certaines règles de réécritures sur son contenu : l'algorithme choisit un but à chaque étape, et peut soit le décomposer en un ensemble (potentiellement vide) de sous-buts en appliquant une règle de $\mathcal{P}$, soit l'éliminer immédiatement s'il s'agit d'un but trivial solvable de $\mathscr{T}(G)$, calculé durant la première phase.

Un ensemble de buts S est qualifié d'*effaçable* s'il peut être réécrit en l'ensemble vide par une série d'applications successives des deux règles suivantes :

— Réécrire S en $S \setminus \{A(\mathtt{D_1} \ldots \mathtt{D_n})\}$, si $\{A(\mathtt{D_1} \ldots \mathtt{D_n})\}$ est un but trivial solvable appartenant à $\mathscr{T}(G)$.

— Réécrire S en $S \setminus \{A(\mathtt{D_1} \dots \mathtt{D_n})\} \cup \Big\{ B_1(\mathtt{N'_{1,1}} \dots \mathtt{N'_{1,n_1}}) \dots B_p(\mathtt{N'_{p,1}} \dots \mathtt{N'_{p,n_p}}) \Big\}$, si il existe n environnements de typage $\Gamma_1 \dots \Gamma_n$ et une règle $r \in \mathcal{P}$ de la forme :

$$r = A(M_1^{\alpha_1}, \dots, M_n^{\alpha_n}) \leftarrow B_1(x_{1,1}, \dots, x_{1,n_1}) \dots B_p(x_{p,1}, \dots, x_{p,n_p})$$

tels que les quatre conditions suivantes (identiques à celles de la première phase) sont respectées :

1. si $\Gamma_i(x_{j,k}) = \boldsymbol{b_{j,k}}$, alors $\mathtt{N'_{j,k}} :: \vdash N'_{j,k} : \boldsymbol{b_{j,k}}$ (cohérence des types)
2. si $\mathtt{N'_{j,k}} = \bot^{\beta_{j,k}}$, alors $x_{j,k} \notin \mathrm{Dom}(\Gamma_i)$ (abandon des éléments $\bot$)
3. si $\mathtt{D_i} = \bot^{\alpha_i}$, alors $\mathrm{Dom}(\Gamma_i) = \emptyset$ (préservation des éléments non-$\bot$)
4. si $\mathtt{D_i} :: \vdash Q_i : \boldsymbol{a_i}$, alors il existe une dérivation $\mathtt{M_i} :: \Gamma_i \vdash M_i : \boldsymbol{a_i}$ et la substitution simultanée de tous les $x_{j,k}$ appartenant au domaine de Γ_i donne $\mathtt{S_i} = \mathtt{M_i}\left[x_{j,k} \leftarrow \mathtt{N'_{j,k}}\right] \overset{*}{\rightarrow}_{\beta\varepsilon} \mathtt{D_i}$ (application de r)

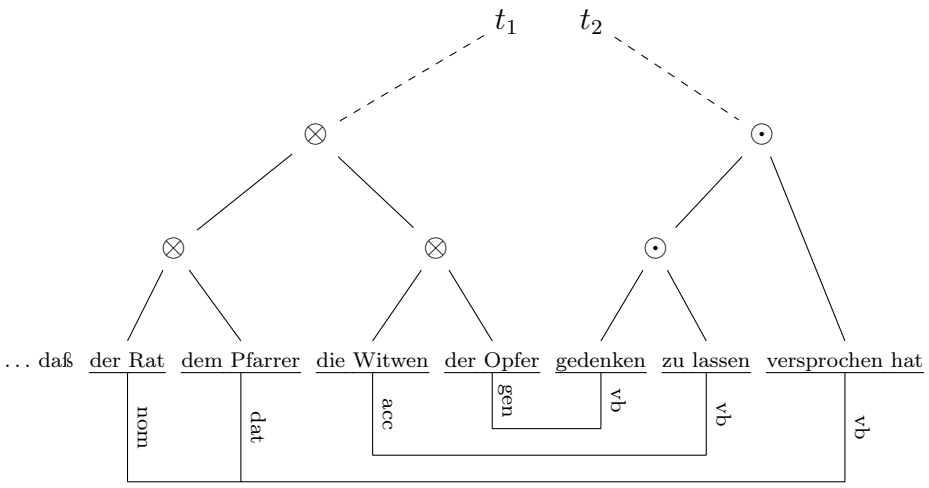
Un but individuel $A(\mathtt{D_1}, \dots, \mathtt{D_n})$ est dit effaçable lorsque le singleton qu'il forme est effaçable. Nous établissons maintenant qu'un but est solvable si et seulement si il est effaçable, et qu'il est possible de décider si un but est effaçable avec une complexité en espace polynomiale par rapport à sa taille : la preuve de ces affirmations est donnée par les théorèmes B.27 et B.29.

5.7 Conclusion

En résumé, nous avons proposé dans ce chapitre un mécanisme algébrique permettant de représenter de manière compacte des phrases où l'ordre des mots est partiellement ou totalement libre. Cet outil a donné lieu à une hiérarchie de grammaires dites commutatives, pour lesquelles nous avons étudié en profondeur la complexité de l'analyse, et dégagé des classes algorithmiquement abordables. Nous avons également proposé et prouvé des algorithmes pour l'analyse de ces différentes classes. Combinée au formalisme décrit dans le chapitre 3, cette représentation peut permettre à une linéarisation de décrire plus exactement, de façon compacte, l'ensemble des énoncés acceptables dans une langue incluant des phénomènes d'ordre libre. Ce travail reste cependant à l'heure actuelle une tâche à accomplir.

En dépit de cela, certains résultats de nature linguistique liés à cette représentation sont d'ores et déjà apparents. En particulier, la modélisation simple d'une langue permettant l'ordonnancement libre de ses syntagmes (telle que le latin) ne semble pas poser de problème majeur, même si la modélisation de phénomènes plus complexes telles que les disjonctions (l'inclusion d'un mot dans une proposition étrangère pour des raisons stylistiques, *cf.* Marouzeau [1922]) peut s'avérer plus délicate.

En revanche, le phénomène de *scrambling* en allemand, qui a originellement motivé le travail présenté dans ce chapitre, soulève un problème intéressant vis-à-vis de la complexité algorithmique de son analyse : le lien entre sa structure profonde et sa réalisation ne semble pas pouvoir être représenté par une CCFG, pour les mêmes raisons que les dépendances croisées du néerlandais échappent au pouvoir de génération forte d'une CFG ; il est en revanche parfaitement capturé par l'emploi d'une CMG employant des contextes unaires, ou d'une CMREG construisant des paires de termes. Rappelons que la complexité de l'analyse à grammaire fixée pour ces dernières est NP-difficile, alors celle des CCFG est identique à celle des grammaires hors-contexte.



(der Opfer, gedenken) → (⊗ die Witwen der Opfer, ⊙ gedenken zu lassen) →
(⊗ (⊗ der Rat dem Pfarrer) (⊗ die Witwen der Opfer), ⊙ (⊙ gedenken zu lassen) versprochen hat)

FIGURE 5.18 – Structure commutative et *scrambling*

Cet argument est illustré par la figure 5.18, reprenant l'un des exemples donné au début de ce chapitre : les termes dessinés au dessus de l'énoncé représentent sa structure phonologique, où la chaîne de verbes présente à la fin de la phrase forme un sous-terme séquentiel t_2, tandis que l'ensemble de leurs arguments est groupé dans un contexte purement commutatif pour former t_1 ; la structure abstraite de l'énoncé figure pour sa part en dessous de celui-ci, formant une série d'arbres enchâssés. Plus bas se trouve une série de réalisations phonologiques possibles pour une telle phrase dans le cadre d'une CMREG : chaque proposition successive est réalisée comme une paire de termes commutatifs, dont le premier contient les arguments verbaux, et le second les verbes eux-mêmes. Ce procédé de linéarisation est remarquablement similaire à celui utilisé dans le chapitre 4 pour rendre compte de l'ordre des mots en néerlandais.

Conclusion

La contribution de cette thèse a consisté en la conception (chapitre 3), la mise à l'épreuve (chapitre 4) et la préparation à l'extension aux ordres libres (chapitre 5) d'un formalisme de haut niveau, tourné vers l'ingénierie grammaticale, pour des tâches de description linguistique. Le formalisme et sa mise à l'épreuve ont fait l'objet d'une publication de journal [Clément *et al.*, 2015] ; et une version préliminaire des résultats du dernier chapitre a été présentée en conférence [Kirman et Salvati, 2013].

Résumé

Le formalisme en lui-même se compose de deux volets. D'une part, la description d'un langage de structures abstraites par l'intermédiaire d'une grammaire support (décrivant un langage d'arbres réguliers) agrémentée de contraintes de bonne formation (formulées dans un langage logique limité par la logique monadique du second ordre) permet de modéliser la structure syntaxique des énoncés, indépendamment de leur forme phonologique. D'autre part, un processus de linéarisation utilisant le lambda-calcul et guidé par la logique, utilisable en syntaxe comme en sémantique, permet de produire à partir de ces structures abstraites différents objets d'intérêt concret pour le TAL. Le lien entre ces objets et le langage abstrait est établi en s'appuyant sur des résultats existants dans la littérature des ACG et sur la connexion entre logique et automates, offrant une procédure de décision efficace à grammaire fixée – avec pour réserve que la taille des grammaires ainsi générées est en pratique excessive en l'absence de techniques d'optimisation spécifiques.

À l'usage, l'emploi de la logique pour définir des prédicats et relations additionnels permet de réifier des concepts linguistiques et de rendre explicites les hypothèses de modélisation de l'auteur d'une grammaire, facilitant sa maintenance ultérieure. Des réalisations phonologiques complexes, telles que l'ordre des mots en néerlandais, peuvent être décrites au moyen du lambda-calcul simplement typé, qui permet également des factorisations significatives en sémantique. Enfin, les phénomènes de mouvement et de dépendances à longue distance (illustrés par le mouvement *wh* et les contraintes d'îlot associées) peuvent être traités respectivement par l'emploi d'un mécanisme de requêtes

logiques et d'expression rationnelles intégrées au langage logique.

Enfin, l'outil algébrique de modélisation des ordres libres proposé dans le dernier chapitre induit une hiérarchie de grammaires commutatives, et nous a permis d'étudier leur complexité algorithmique en proposant des algorithmes d'analyse pour chacune des classes ainsi obtenues. Deux d'entre elles s'avèrent analysables efficacement à grammaire fixée, et permettent une modélisation simple des langues à ordre libre. Des phénomènes plus spécifiques, tels que le *scrambling*, peuvent également être modélisés de cette manière, bien que les langages résultants n'offrent pas de bonnes garanties de complexité algorithmique.

Questionnement

Au fil du document, nous avons soulevé plusieurs questions qui demeurent en suspens. Nous récapitulons ici quelques unes de ces interrogations, et tentons d'y apporter des éléments de réponse.

Compilation et optimisation La question la plus importante qui demeure non résolue à l'issue de ce travail est celle de la taille effective des grammaires générées par le formalisme. Nous nous sommes concentrés sur la complexité de l'analyse à grammaire fixée (qui est polynomiale) dans la majeure partie du document en supposant que, pour des applications de TAL, la grammaire est constante et peut donc être pré-calculée, optimisée et enregistrée à l'avance sur des supports de grande capacité. Cependant, il n'est pas exclu que la représentation compilée d'une grammaire crédible dépasse de loin les capacités de stockage des machines actuelles.

Le moyen le plus direct de répondre à cette question passe par l'implémentation du formalisme : celle-ci se heurtera tôt ou tard à ce problème, et requerra l'emploi de diverses techniques d'optimisation. Nous avons mentionné à l'issue du chapitre 3 l'existence d'outils logiciels dédiés à la vérification de modèles (*model-checking*), tels que MONA [Klarlund et Møller, 2001] ; ainsi que l'article de Morawietz et Cornell [1997], qui évoque plusieurs techniques d'optimisation possible pour la syntaxe en théorie des modèles. L'une d'elle est la construction d'automates partiels, indépendants, suivant une stratégie de type diviser-pour-régner ; une autre suppose la restriction de la grammaire a ses fragments applicables à l'énoncé à analyser – celle-ci entraîne cependant une compilation dynamique lors de l'analyse. Une solution modérée de ce type pourrait consister en la mise en cache de plusieurs grammaires « partielles », de taille raisonnable, et en le choix d'une ou plusieurs de ces grammaires précompilées en fonction du contenu de l'énoncé. Le cas échéant, il est aussi possible d'abandonner complètement l'étape de pré-compilation de la grammaire, en évaluant ses règles dynamiquement lors de l'analyse. À titre d'exemple, une

technique de ce type a été implémentée avec succès pour la vérification de certaines propriétés monadiques du second ordre sur les graphes par Courcelle et Durand [2011], sous la forme d'automates-à-la-volée (*fly-automata*).

Utilité des requêtes logiques L'un des composants les plus complexes du formalisme, compliquant considérablement le processus de compilation des linéarisations, est celui des requêtes logiques. Ces dernières ont été introduites pour faciliter le traitement du mouvement (notamment *wh*), et remplissent ce rôle efficacement ; toutefois, elles semblent constituer un ajout *ad-hoc* pour un phénomène linguistique spécifique. Le mouvement d'un fragment de réalisation peut également être traité par le lambda-calcul, en formant une paire de chaînes dont le premier élément est la chaîne localement réalisée, et le second est le composant déplacé (par exemple un pronom relatif). La réalisation du site d'insertion peut alors concaténer ces deux chaînes de la manière appropriée, comme le mentionne le paragraphe « Linéarisation alternative » page 92.

Un premier argument en leur faveur est cependant que, bien qu'elles compliquent l'implémentation du formalisme, l'emploi de ces requêtes facilite indiscutablement la modélisation du mouvement par rapport à l'emploi de paires de chaînes : or, la conception d'un outil de modélisation simple et de haut niveau reste notre objectif premier. Une autre raison de leur maintien dans le formalisme est l'existence d'autres emplois potentiels. D'une part, les phénomènes (particulièrement complexes) liés à l'ellipse pourraient bénéficier de la capacité de déplacer un composant commun dans la structure abstraite de la phrase, en fonction de certains choix de réalisation ; cette perspective semble toutefois assez incertaine. D'autre part, comme mentionné en conclusion du chapitre 4, les requêtes logiques pourraient être exploitées dans des linéarisations sémantiques pour assurer un traitement correct de la quantification sans requérir les constructions compositionnelles, à base de montée de type, actuellement dominantes dans la littérature.

Utilité de la grammaire support Une question plus subsidiaire dans la continuité de la précédente est celle de la nécessité de la grammaire support (ou grammaire d'approximation). Introduite pour faciliter la visualisation de la forme générale des structures abstraites et pour servir de point d'ancrage aux contraintes logiques et règles de linéarisation, elle n'est toutefois pas indispensable à la description des structures abstraites. Celles-ci pourraient en effet être entièrement spécifiées par des contraintes logiques. La perte en termes de simplicité visuelle semble trop importante pour justifier ce choix, mais il présente néanmoins un avantage : pour le rattachement de certaines contraintes à longue distance, telles que liens entre pronoms et antécédents ou entre site d'insertion et site d'extraction, le choix d'une production spécifique (ou plus) dans la grammaire support est quelque peu arbitraire. En un sens, ces contraintes

sont vraies à travers toute la structure abstraite, et non simplement là où une production les instancie.

Complexité du *scrambling* Une dernière interrogation porte sur la complexité algorithmique du *scrambling*. Comme souligné à l'issue du chapitre 5, celui-ci peut-être modélisé par des classes de grammaires commutatives dont la complexité est NP-difficile, et semble échapper à l'analyse par les classes polynomiales.

Il se pourrait cependant que cette difficulté soit intrinsèque au phénomène. L'exemple mentionné dans le chapitre 5 peut être assez rapidement résolu grâce au marquage des cas : les quatre arguments verbaux de la phrase ont autant de cas différents, le nominatif étant réservé à la première proposition dans une structure de contrôle, et chacun des trois compléments ne pouvant correspondre qu'à l'un des trois verbes présents. En l'absence de tels liens (ou de restrictions de sélection sémantique), la structure abstraite correspondant à un tel énoncé serait sous-spécifiée, permettant à une analyse quelconque d'être justifiable. Cependant, rien ne semble entraver en théorie la présence d'un grand nombre de restrictions grammaticales (ou sémantiques), limitant les possibilités d'appariement à certaines combinaisons de verbes et d'arguments ; dans un tel cas, la construction d'une structure abstraite satisfaisant toutes ces contraintes ressemble à la solution d'un problème de 3-couverture exacte, intrinsèquement NP-difficile.

Cet argument n'est toutefois pas entièrement satisfaisant, d'une part en raison de son côté très informel, et d'autre part parce que si le nombre de restrictions de sélection (ou de cas) est fini, ce qui est par hypothèse le cas dans une grammaire fixée, alors les regroupements effectués dans la structure abstraite (correspondant aux sous-ensembles de l'instance de X3C) possèdent une structure propre, susceptible d'altérer la complexité du problème.

Perspectives

Enfin, le formalisme tel qu'il est présenté dans le cadre de cette thèse ouvre de nombreuses perspectives de développement supplémentaires. La tâche la plus évidente est l'implémentation du formalisme, et son optimisation. Il s'agit cependant d'un travail considérable, impliquant la mise au point de nombreux composants logiciels inédits (par exemple pour l'analyse commutative) ou la réutilisation de bibliothèques (comme MONA) dont le développement est à l'arrêt. Le travail d'optimisation qui l'accompagne semble encore plus ambitieux, bien que sans doute plus prometteur sur le plan de l'intérêt des problèmes qu'il soulève.

Un autre travail, préalable, serait d'améliorer l'accessibilité du formalisme du point de vue d'un linguiste n'ayant pas d'expérience dans l'usage de la lo-

gique et du lambda-calcul. En effet, tout au long de ce document, nous n'avons pas particulièrement cherché à dissimuler le fonctionnement interne du formalisme (qui, au contraire, est au centre de la discussion) ; il paraît cependant peu concevable qu'un syntacticien souhaite s'embarrasser de lambda-abstractions, d'applications, et d'encodage des chaînes, termes, tuples et projections par le lambda-calcul simplement typé. Les primitives que représentent nos réalisations sont pourtant relativement simples : par exemple, les tuples de chaînes employés pour l'ordre des mots en néerlandais peuvent être plus aisément envisagés comme des « chaînes à trous » où les arguments et verbes s'insèrent de part et d'autre, et il en va de même des réalisations sémantiques, particulièrement complexes. Disposer d'une présentation plus accessible pour ces éléments (quitte à restreindre incidemment le pouvoir d'expression des linéarisations) pourrait faciliter l'abord de ces techniques de modélisation.

Un autre facteur de simplification des réalisations sémantiques serait, ainsi qu'évoqué précédemment, l'emploi de requêtes logiques pour simplifier le traitement de la sémantique. En effet, dans le paradigme actuel, la quantification porte sur une proposition complète, mais est décidée par les pronoms et déterminants de ses arguments. Pour remédier à ce problème, une construction à montée de type permet classiquement au déterminant de recevoir et de traiter, par le biais d'une continuation, le contenu de son groupe nominal, puis celui de l'ensemble de la phrase, ce qui alourdit inutilement le sens d'une proposition partielle. Une solution possible serait de requérir, à longue distance, une certaine propriété du déterminant (sélectionnant le type de quantification qu'il impose) et de construire la sémantique de toute la proposition d'une traite, simplifiant les types et formules intermédiaires. La clarté du résultat et la présence éventuelle d'obstacles à un tel traitement demeurent toutefois encore à déterminer.

Enfin, deux tâches restent à accomplir en rapport avec les grammaires commutatives du dernier chapitre. La première est leur intégration directe dans le formalisme dans le cadre d'une tâche de modélisation similaire à celle entreprise dans le chapitre 4. Cela permettrait de confronter à des phénomènes linguistiques réels notre représentation des ordres libres, et de garantir son bon fonctionnement. La seconde tâche consiste à explorer plus avant les propriétés de ces grammaires. Nous avons étudié en détail la complexité de l'analyse, mais certains détails restent en suspens : la complexité exacte de l'analyse universelle des CMCFG et CMG n'est pas fixée dans tous les cas, et les propriétés de clôture exactes des classes de langages commutatifs restent encore à établir.