Convertir des grammaires d'arbres adjoints à composantes multiples avec tuples d'arbres (TT-MCTAG) en grammaires à concaténation d'intervalles (RCG)

Laura Kallmeyer[*], Yannick Parmentier[**]

** SFB441 - Universität Tübingen – Nauklerstr. 35, D-72 074 Tübingen, Germany*

lk@sfs.uni-tuebingen.de

*** LORIA - Nancy Université – Campus Scientifique Victor Grignard – BP 239 F-54 506 Vandœuvre-Lès-Nancy Cedex, France*

parmenti@loria.fr

RÉSUMÉ. *Cet article étudie la relation entre le formalisme TT-MCTAG et le formalisme RCG. RCG est connu pour décrire exactement la classe PTIME. TT-MCTAG a été proposé pour modéliser les langues à ordre des mots libre. Nous montrons comment une forme restreinte de TT-MCTAG peut être convertie en une RCG « simple » équivalente. Le résultat est intéressant pour des raisons théoriques (il montre que la forme restreinte est légèrement sensible au contexte), mais également pour des raisons pratiques (la conversion proposée a été implantée dans un analyseur TT-MCTAG).*

ABSTRACT. *This paper investigates the relation between TT-MCTAG, and RCG. RCG is known to describe exactly the class PTIME. TT-MCTAG has been proposed to model free word order languages. We show that TT-MCTAG with an additional limitation can be transformed into equivalent "simple" RCG. This result is interesting for theoretical reasons (since it shows that TT-MCTAG in this limited form is mildly context-sensitive) and also for practical reasons (the transformation has been implemented in a parser for TT-MCTAG).*

MOTS-CLÉS : *grammaires d'arbres adjoints, grammaires à concaténation d'intervalles.*

KEYWORDS: *Tree Adjoining Grammars, Range Concatenation Grammars.*

TAL. Volume 50 – n°1/2009, pages 227 à 255

1. Introduction

Les grammaires d'arbres adjoints (TAG, (Joshi et Schabes, 1997)) sont un formalisme de réécriture d'arbre, introduit pour la première fois dans (Joshi *et al.*, 1975). Bien que ce formalisme ait été utilisé pour décrire certaines langues (par exemple, l'anglais (XTAG Research Group, 2001)), il s'est avéré trop limité pour traiter certains phénomènes linguistiques tels que les variations d'ordre des mots. C'est dans ce contexte que les TAG à composantes multiples (MCTAG, (Joshi, 1987 ; Weir, 1988)) ont été proposées. Leur motivation repose sur la volonté de découper la contribution d'un item lexical (*e.g.*, un verbe et ses arguments) en plusieurs arbres TAG élémentaires. Une MCTAG consiste donc en ensembles d'arbres élémentaires, appelés composantes multiples. Si une composante multiple est utilisée lors d'une dérivation, tous ses membres (arbres) doivent être utilisés. Le type particulier de MCTAG que nous considérons dans cet article est MCTAG avec tuples d'arbres et nœuds partagés (Tree-Tuple MCTAG with Shared Nodes, TT-MCTAG, (Lichte, 2007)). TT-MCTAG a été introduit pour modéliser les phénomènes d'ordre des mots libre[1]. Un exemple de ce phénomène est (1) où l'argument *es* de *reparieren* précède l'argument *der Mechaniker* de *verspricht* et est ainsi non adjacent au prédicat dont il dépend :

(1) ... dass es der Mechaniker zu reparieren verspricht
... que le le mécanicien de réparer promet
'... que le mécanicien promet de le réparer'

Une TT-MCTAG est légèrement différente d'une MCTAG standard[2] dans la mesure où les ensembles d'arbres élémentaires contiennent deux parties : la première, exactement un arbre lexicalisé γ, marqué comme l'unique *arbre tête*, et la seconde, un ensemble (éventuellement vide) d'arbres auxiliaires, les *arbres arguments*. Une telle paire est appelée tuple d'arbres. Lors de la dérivation, les arbres arguments doivent soit s'adjoindre directement à leur tête, soit être liés par une chaîne d'adjonctions aux nœuds racines des arbres qui s'attachent à leur arbre tête. En d'autres termes, dans l'arbre de dérivation TAG sous-jacent, un nœud d'arbre tête doit dominer les nœuds d'arbres arguments de telle sorte que les positions sur le chemin entre la tête et l'argument, à l'exception de la première, doivent être annotées de l'adresse ϵ (adjonction au nœud racine). Ceci correspond à la notion d'adjonction avec nœud partagé de (Kallmeyer, 2005)[3].

1. Voir (Lichte, 2007 ; Lichte et Kallmeyer, 2008) pour des analyses de phrases en allemand avec TT-MCTAG.

2. Cette caractérisation de TT-MCTAG repose sur des notions inhérentes au formalisme TAG (arbre auxiliaire, adjonction, arbre de dérivation, etc.). Ces notions sont définies formellement en section 2.

3. L'intuition est que si un arbre γ' s'adjoint à un arbre γ, sa racine dans l'arbre dérivé résultat appartient en quelque sorte à la fois à γ et γ', elle est donc partagée par eux. Un autre arbre β s'adjoignant à ce nœud peut être considéré comme s'adjoignant à γ, et pas seulement à γ' comme c'est le cas en TAG habituellement. Notons que nous supposons que les nœuds pieds

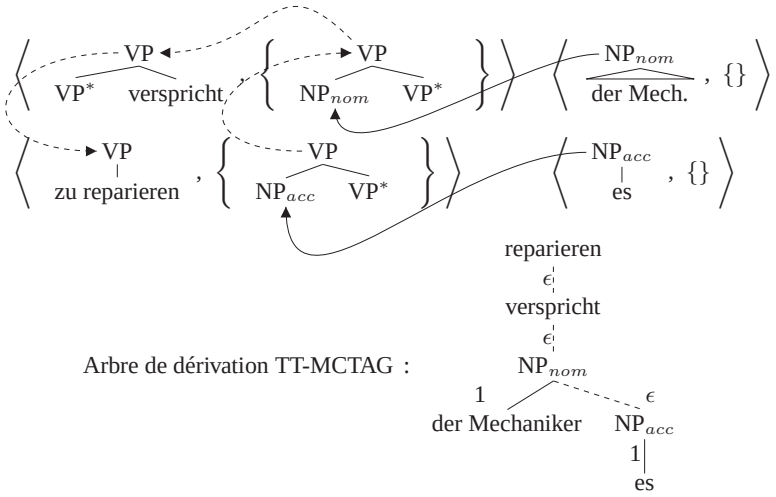


Figure 1. *Dérivation TT-MCTAG pour (1)*

La figure 1 montre une dérivation TT-MCTAG pour (1). Ici, l'arbre auxiliaire VP-NP$_{nom}$ s'adjoint directement à *verspricht* (sa tête) alors que l'arbre VP-NP$_{acc}$ s'adjoint à la racine d'un arbre qui s'adjoint à la racine d'un arbre qui s'adjoint lui-même à *reparieren*.

Afin de pouvoir mieux situer la classe des langages générés par TT-MCTAG et, aussi, afin de développer des analyseurs syntaxiques pour TT-MCTAG, nous définissons une restriction de TT-MCTAG, les TT-MCTAG de rang k (k-TT-MCTAG). Pour k-TT-MCTAG nous donnons un algorithme de conversion en grammaire à concaténation d'intervalles (*Range Concatenation Grammar*, RCG) simple. La RCG simple qu'on construit de cette façon est fortement équivalente à la TAG de départ dans le sens où les deux génèrent des structures de dérivation isomorphes. Ceci nous montre deux choses : tout d'abord, comme corollaire nous obtenons que les langages générés par k-TT-MCTAG sont légèrement sensibles au contexte. De plus, la conversion en RCG simple peut être exploitée pour une analyse syntaxique de k-TT-MCTAG.

L'algorithme de transformation d'une k-TT-MCTAG en une RCG simple a été implanté dans un analyseur pour TT-MCTAG, à savoir le *Tuebingen Linguistic Parsing Architecture* (TuLiPA, (Kallmeyer *et al.*, 2008b ; Parmentier *et al.*, 2008 ; Parmentier et Maier, 2008)). L'analyse syntaxique de TT-MCTAG avec TuLiPA est réalisée en quatre étapes. Dans un premier temps, la chaîne d'entrée est utilisée pour extraire une sous-grammaire de la TT-MCTAG. Dans une deuxième étape, cette sous-grammaire est transformée en une RCG équivalente en utilisant l'algorithme de transformation décrit dans cet article. Dans une troisième étape, la RCG ainsi produite est utilisée pour analyser la chaîne d'entrée, créant une forêt de dérivation RCG (*i.e.*, une re-

ne peuvent pas recevoir d'adjonction, dans le cas contraire, le partage de nœud s'appliquera également à eux.

présentation compacte de tous les arbres de dérivation RCG). Enfin, chaque arbre de dérivation RCG est interprété pour extraire l'arbre de dérivation TAG correspondant (*i.e.*, l'arbre de dérivation TAG pour la TT-MCTAG d'entrée).

La structure de l'article est la suivante : la section 2 définit formellement les TAG et la section 3 les RCG. Ensuite, en section 4, les TT-MCTAG sont présentées. La section 5 donne l'algorithme de conversion de TT-MCTAG de rang k en RCG simple. Finalement, la section 6 traite de la relation entre TAG et TT-MCTAG de rang k.

2. Grammaires d'arbres adjoints (*Tree Adjoining Grammars*)

Une TAG consiste en un ensemble fini d'arbres (arbres élémentaires). Les nœuds de ces arbres sont étiquetés par des symboles terminaux ou non terminaux (les symboles terminaux n'apparaissent que sur les nœuds feuilles).

À partir d'arbres élémentaires, on dérive des arbres plus grands (*i.e.*, ayant un plus grand nombre de nœuds) par *substitution* (remplacement d'un nœud feuille par un nouvel arbre) et *adjonction* (remplacement d'un nœud non feuille, appelé nœud interne, par un nouvel arbre). Dans le cas d'une adjonction, l'arbre inséré a exactement un nœud feuille défini comme étant le nœud pied (marqué par un astérisque). Un tel arbre est appelé *arbre auxiliaire*. Afin de permettre l'adjonction d'un arbre à un nœud étiqueté n, la racine et le nœud pied de cet arbre doivent être étiquetés par ce même symbole n. Lors de l'adjonction au nœud n, dans l'arbre résultat, le sous-arbre de racine n de l'arbre avant adjonction se retrouve attaché sous le nœud pied de l'arbre auxiliaire inséré. Les arbres élémentaires non auxiliaires sont appelés arbres *initiaux*. Une dérivation débute avec un arbre initial. Dans un arbre dérivé final, tous les nœuds feuilles doivent être étiquetés par un symbole terminal (ou la chaîne vide ε). Un exemple de dérivation est donné en figure 2.

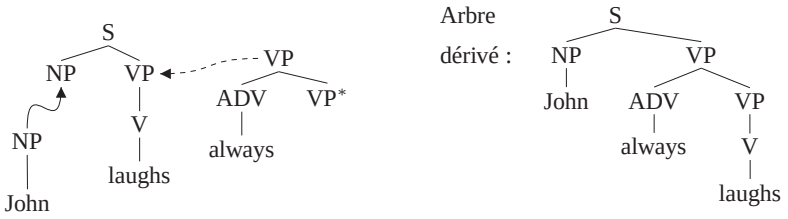


Figure 2. *Dérivation de la phrase «* John always laughs *»*

Dans une TAG, il est possible de spécifier, pour chaque nœud, a) si celui-ci doit ou peut recevoir une adjonction, et si oui b) quels arbres peuvent être adjoints.

Afin de donner une définition formelle de TAG, nous avons besoin de quelques définitions préliminaires sur les arbres :

Définition 1 (Arbre)

1) Soit un graphe orienté $\gamma = \langle V, E\rangle$, où V est l'ensemble des sommets (vertices) et E l'ensemble des arêtes (edges), et E et V sont tels que $E \in \mathcal{P}(V \times V)$. Un arbre *est un triplet $\langle V, E, r\rangle$ tel que*

- γ ne contient pas de cycle,

- seule la racine $r \in V$ n'a pas d'arête entrante,

- chaque sommet $v \in V$ est accessible depuis r, et

- tous les sommets $v \in V - \{r\}$ ont une seule arête entrante.

2) un arbre est ordonné *s'il est équipé d'une relation de* précédence linéaire $\prec \subseteq V \times V$ *telle que*

- $\prec$ est irréflexive, antisymétrique et transitive,

- pour tout v_1, v_2 avec $\{\langle v_1, v_2\rangle, \langle v_2, v_1\rangle\} \cap E^* = \emptyset$:

soit $v_1 \prec v_2$, soit $v_2 \prec v_1$,

de plus, s'il existe un $\langle v_3, v_1\rangle \in E$ avec $v_3 \prec v_2$ ou un $\langle v_4, v_2\rangle \in E$ avec $v_1 \prec v_4$, alors $v_1 \prec v_2$,

- rien d'autre n'est dans $\prec$.

Un sommet sans arête sortante est appelé une feuille. De plus, les sommets d'un arbres sont appelés nœuds.

Définition 2 (Étiquetage)

Un étiquetage *d'un graphe $\gamma = \langle V, E\rangle$ par rapport à une signature $\langle A_1, A_2\rangle$ est une paire de fonctions $l : V \to A_1$ et $g : E \to A_2$ avec A_1, A_2 pouvant être distincts.*

Pour définir une TAG, nous avons besoin d'une définition d'arbre initial et auxiliaire. Dans ce qui suit, nous présumons l'existence d'un alphabet N de symboles non terminaux et d'un alphabet T de symboles terminaux.

Définition 3 (Arbre auxiliaire et initial)

1) Un arbre syntaxique *est un arbre fini ordonné étiqueté tel que $l(v) \in N$ pour chaque sommet v ayant au moins une arête sortante, et $l(v) \in (N \cup T \cup \{\epsilon\})$ pour chaque feuille v.*

2) Un arbre auxiliaire *est un arbre syntaxique $\langle V, E, r\rangle$ tel qu'il existe une unique feuille f marquée comme pied avec $l(r) = l(f)$. Nous notons de tels arbres $\langle V, E, r, f\rangle$.*

3) Un arbre initial *est un arbre syntaxique non auxiliaire.*

À présent, nous pouvons définir une TAG.

Définition 4 (Grammaire d'arbres adjoints, TAG)

Une grammaire d'arbres adjoints *(TAG) est un 7-uplet* $G = \langle N, T, I, A, S, f_{OA}, f_{SA} \rangle$ *où*

– N, T *sont des alphabets disjoints de symboles non terminaux et terminaux,*

– I *et* A *sont des ensembles finis d'arbres initiaux et auxiliaires respectivement,*

– $S \in N$ *est le symbole initial,*

– $f_{OA} : \{v \,|\, v \text{ sommet de } \gamma \in I \cup A\} \rightarrow \{0, 1\}$ *et* $f_{SA} : \{v \,|\, v \text{ sommet de } \gamma \in I \cup A\} \rightarrow \mathcal{P}(A)$ *sont des fonctions telles que* $f_{OA}(v) = 0$ *et* $f_{SA}(v) = \emptyset$ *pour chaque feuille* v*.*

Tout arbre de $I \cup A$ *est appelé un* arbre élémentaire.

Pour un nœud donné, la fonction f_{OA} spécifie si l'adjonction est obligatoire (valeur 1) ou non (valeur 0) et f_{SA} donne l'ensemble des arbres auxiliaires qui peuvent être adjoints. Seuls les nœuds internes peuvent recevoir une adjonction, l'adjonction à un nœud feuille n'est pas autorisée. Par convention de notation, nous omettons souvent les fonctions f_{OA} et f_{SA} dans la notation de 7-uplet, *i.e.*, nous notons les TAG sous forme de quintuplets $\langle N, T, I, A, S \rangle$.

Dans le formalisme TAG, des arbres plus grands (*i.e.*, ayant un plus grand nombre de nœuds) sont dérivés à partir de $I \cup A$ en appliquant successivement des opération de substitution et d'adjonction. L'opération de substitution combine un arbre syntaxique et un arbre initial pour former un nouvel arbre syntaxique, alors que l'adjonction combine un arbre syntaxique et un arbre auxiliaire pour former un nouvel arbre syntaxique.

Définition 5 (Substitution)

Soient $\gamma = \langle V, E, r \rangle$ *un arbre syntaxique,* $\gamma' = \langle V', E', r' \rangle$ *un arbre initial et* $v \in V$*.* $\gamma[v, \gamma']$*, le résultat de la* substitution *de* γ' *dans* γ *au nœud* v *est défini comme suit :*

– si v *n'est pas une feuille ou si* v *est un nœud pied ou si* $l(v) \neq l(r')$*, alors* $\gamma[v, \gamma']$ *n'est pas défini ;*

– sinon, $\gamma[v, \gamma'] = \langle V'', E'', r'' \rangle$ *avec* $V'' = V \cup V' \setminus \{v\}$ *et* $E'' = (E \setminus \{\langle v_1, v_2 \rangle \,|\, v_2 = v\}) \cup E' \cup \{\langle v_1, r' \rangle \,|\, \langle v_1, v \rangle \in E\}$*.*

Une feuille qui est étiquetée par un symbole non terminal et qui n'est pas un nœud pied est appelée nœud de substitution.

Définition 6 (Adjonction)

Soient $\gamma = \langle V, E, r \rangle$ *un arbre syntaxique,* $\gamma' = \langle V', E', r', f \rangle$ *un arbre auxiliaire et* $v \in V$*.* $\gamma[v, \gamma']$*, le résultat de l'*adjonction *de l'arbre* γ' *dans* γ *au nœud* v *est défini comme suit :*

– si v *est une feuille ou si* $l(v) \neq l(r')$*, alors* $\gamma[v, \gamma']$ *n'est pas défini ;*

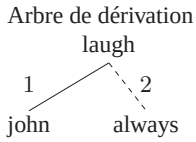


Figure 3. *Arbre de dérivation pour la dérivation donnée en figure 2*

– sinon, $\gamma[v,\gamma'] = \langle V'', E'', r''\rangle$ *avec* $V'' = V \cup V' \setminus \{v\}$ *et* $E'' = (E \setminus \{\langle v_1, v_2\rangle \mid v_1 = v$ *ou* $v_2 = v\}) \cup E' \cup \{\langle v_1, r'\rangle \mid \langle v_1, v\rangle \in E\} \cup \{\langle f, v_2\rangle \mid \langle v, v_2\rangle \in E\}$.

Les dérivations TAG sont représentées par des *arbres de dérivation* qui enregistrent l'historique des combinaisons d'arbres élémentaires. Un *arbre dérivé* est l'arbre syntaxique résultant de l'application des substitutions et adjonctions définies par l'arbre de dérivation. En d'autres termes, l'arbre de dérivation décrit de manière unique l'arbre dérivé[4].

Chaque arête de l'arbre de dérivation représente une adjonction ou une substitution. Les arêtes sont étiquetées par les adresses de Gorn[5] des nœuds où ont eu lieu les adjonctions et substitutions en question. Pour illustrer cela, la figure 3 représente la dérivation de la phrase « *John always laughs* », plus précisément, l'arbre de dérivation indique que l'arbre élémentaire pour *John* est substitué au nœud d'adresse 1 de l'arbre pour *laughs* et celui pour *always* est adjoint au nœud d'adresse 2 de l'arbre pour *laughs* (le fait que cette dernière opération soit une adjonction, et la précédente une substitution, peut être inféré par le fait que le nœud d'adresse 1 de l'arbre pour *laughs* est une feuille qui n'est pas un nœud pied, et le nœud 2 un nœud interne)[6].

Nous appelons un arbre γ' qui est isomorphe à un arbre γ en préservant l'étiquetage, une instance de γ. De plus, nous considérons que deux arbres sont disjoints si leurs ensembles de nœuds sont disjoints.

Définition 7 (Arbre dérivé et arbre de dérivation)

Soit $G = \langle N, T, I, A, S\rangle$ *une TAG.*

1) Toute instance γ *d'un* $\gamma_e \in I \cup A$ *est un arbre dérivé dans* G.

L'arbre de dérivation correspondant est $\langle\{v\}, \emptyset, v\rangle$ *avec* $l(v) = \gamma_e$.

4. Nous supposons qu'une fois une adjonction, ou substitution, appliquée à un nœud, ce dernier disparaît. Ce qui veut dire que des adjonctions multiples sont impossibles. En conséquence de quoi, l'ordre d'application des adjonctions et substitutions n'a pas d'influence sur l'arbre dérivé résultat, les nœuds frères de l'arbre de dérivation sont donc non ordonnés.

5. Dans ce système d'adressage, l'adresse de la racine est ϵ, et l'adresse du j^{e} fils d'un nœud d'adresse p est pj.

6. Par souci de clarté, nous représenterons les arêtes référant à des adjonctions avec des lignes en pointillée.

2) Considérons (i) les arbres dérivés $\gamma_1, \ldots, \gamma_n$ de G ayant pour arbre de dérivation respectif $D_i = \langle V_i, E_i, r_i \rangle$ ($1 \leq i \leq n$) et (ii) $\gamma = \langle V, E, r \rangle$ l'instance d'un $\gamma_e \in I \cup A$, $v_1, \ldots, v_n \in V$ avec des adresses de Gorn $p_1, \ldots, p_n$ tel que $\gamma_1, \ldots, \gamma_n, \gamma$ sont deux à deux disjoints.

Si

- $\gamma' = \gamma[v_1, \gamma_1] \ldots [v_n, \gamma_n]$ est définie,

- $l(r_i) \in f_{SA}(v_i)$ pour tout $\gamma_i \in A$,

alors γ' est un arbre dérivé dans G avec pour arbre de dérivation $D = \langle V_D, E_D, r \rangle$ tel que $V_D = \bigcup_{i=1}^{n} V_i \cup \{r\}$, $E_D = \bigcup_{i=1}^{n} E_i \cup \{\langle r, r_1 \rangle, \ldots, \langle r, r_n \rangle\}$ et $l(r) = \gamma_e$ et $g(\langle r, r_i \rangle) = p_i$ pour $1 \leq i \leq n$.

3) Rien d'autre n'est un arbre dérivé ou de dérivation dans G.

Un arbre dérivé qui ne contient aucun nœud de substitution ou nœud interne v vérifiant $f_{OA}(v) = 1$ est appelé arbre dérivé saturé *et l'arbre de dérivation correspondant arbre de dérivation* saturé.

Définition 8 (Langage d'arbres)

Soit $G = \langle N, T, I, A, S \rangle$ une TAG. Le langage d'arbres *de G est $L_T(G) = \{\gamma = \langle V, E, r \rangle \mid \gamma$ est un arbre initial dérivé saturé dans $G, l(r) = S\}$. Le* langage de chaînes *de G est l'ensemble des productions*[7] *des arbres dans $L_T(G)$.*

3. Grammaires à concaténation d'intervalles (*Range Concatenation Grammar*)

Ce paragraphe introduit les grammaires à concaténation d'intervalles (RCG, (Boullier, 1999 ; Boullier, 2000)).

3.1. *Définition des RCG*

Définition 9 (Grammaire à concaténation d'intervalles)

Une grammaire à concaténation d'intervalles positive *est un quintuplet $G = \langle N, T, V, S, P \rangle$ tel que*

– N est un ensemble fini de prédicats, chacun ayant une arité fixée ;

– T et V sont des alphabets disjoints de symboles terminaux et de variables ;

– $S \in N$ est le prédicat de départ (axiome), ayant pour arité 1 ;

– P est un ensemble fini de clauses *de la forme*
$A_0(x_{01}, \ldots, x_{0a_0}) \rightarrow \epsilon$, ou

7. Dans ce qui suit, nous entendons par *production* la liste ordonnée des symboles terminaux étiquetant les nœuds feuilles d'un arbre dérivé, c'est-à-dire sa frontière.

$A_0(x_{01}, \ldots, x_{0a_0}) \rightarrow A_1(x_{11}, \ldots, x_{1a_1}) \ldots A_n(x_{n1}, \ldots, x_{na_n})$ *avec* $n \geq 1$
où $A_i \in N, x_{ij} \in (T \cup V)^*$ *et* a_i *est l'arité de* A_i.

Puisque tout au long de cet article, nous n'utiliserons que les RCG positives, chaque fois que nous emploierons les initiales « RCG », nous entendrons par là, « RCG positive »[8]. Une RCG dont l'arité maximale d'un prédicat est n, est appelée RCG d'arité n.

Lorsque nous appliquons une clause à une chaîne $w = t_1 \ldots t_n$, les arguments des prédicats sont instanciés par les occurrences de sous-chaînes de w, plus précisément par les intervalles correspondants. Un intervalle $\langle i, j \rangle$ avec $0 \leq i < j \leq n$ correspond à la sous-chaîne située entre les positions i et j, *i.e.*, à la sous-chaîne $t_{i+1} \ldots t_j$. Si $i = j$, alors $\langle i, j \rangle$ correspond à la chaîne vide ϵ. Si $i > j$, alors $\langle i, j \rangle$ n'est pas défini.

Définition 10 (Instanciation de clause)

Pour une clause c *donnée, une* instanciation *par rapport à une chaîne* $w = t_1 \ldots t_n$ *consiste en la fonction* $f : \{t' \mid t'$ *est une occurrence d'un* $t \in T$ *dans la clause*$\} \cup V \rightarrow \{\langle i, j \rangle \mid 0 \leq i \leq j \leq n\}$ *telle que*

a) pour toutes les occurrences t' *d'un symbole* $t \in T$ *dans* c *:* $f(t') = \langle i, i+1 \rangle$ *pour un* $i, 0 \leq i < n$ *tel que* $t_{i+1} = t$ *(*i.e.*, un symbole terminal a pour longueur* 1*),*

b) pour tout $v \in V$ *dans* c *:* $f(v) = \langle k, l \rangle$ *avec* $0 \leq k \leq l \leq n$ *(*i.e.*, toute variable est associée à un intervalle éventuellement vide de la chaîne d'entrée), et*

c) si des variables et occurrences de symboles terminaux consécutifs dans un argument du prédicat du membre gauche de la clause sont associés à $\langle i_1, j_1 \rangle, \ldots, \langle i_k, j_k \rangle$ *pour un certain* k*, alors* $j_m = i_{m+1}$ *pour tout* m *tel que* $1 \leq m < k$*. Par définition, nous disons alors que* f *associe tout l'argument avec* $\langle i_1, j_k \rangle$ *(i.e., les symboles apparaissant de manière contiguë au sein d'un argument d'un prédicat gauche d'une clause réfèrent à des intervalles contigus de la chaîne d'entrée).*

Pour chaque clause c *de la forme*
$A_0(x_{01}, \ldots, x_{0a_0}) \rightarrow A_1(x_{11}, \ldots, x_{1a_1}) \ldots A_n(x_{n1}, \ldots, x_{na_n})$ *avec* f *une instanciation de* c *par rapport à* w*,*
$A_0(f(x_{01}), \ldots, f(x_{0a_0})) \rightarrow A_1(f(x_{11}), \ldots, f(x_{1a_1})) \ldots A_n(f(x_{n1}), \ldots, f(x_{na_n}))$
est une clause instanciée.

La relation de dérivation est définie comme suit.

8. La variante négative des RCG autorise des appels de prédicats de la forme $\overline{A(\alpha_1, \ldots, \alpha_n)}$. Un tel prédicat est utilisé pour reconnaître le langage complémentaire de celui reconnu par la forme non surlignée, *cf.* (Boullier, 2000).

Définition 11 (Dérivation RCG et langage de chaînes)

– Soient $\phi_1, \phi_2 \in \{A(\vec{\rho}) \mid A \in N$, $\vec{\rho}$ un vecteur de k intervalles, k l'arité de $A\}^+$.

$\phi_1 \Rightarrow \phi_2$ par rapport à une chaîne donnée w ssi (i) $\phi_1 = \psi_1 A_0(\vec{\rho}_0)\psi_2$, (ii) $\phi_2 = \psi_1 A_1(\vec{\rho}_1) \ldots A_n(\vec{\rho}_n)\psi_2$ et (iii) il existe une clause instanciée par rapport à w s'écrivant $A_0(\vec{\rho}_0) \rightarrow A_1(\vec{\rho}_1) \ldots A_n(\vec{\rho}_n)$.

$\stackrel{*}{\Rightarrow}$ est la clôture réflexive transitive de $\Rightarrow$.

*– Le langage généré par une RCG G est $L(G) = \{w \mid S(\langle 0, |w| \rangle) \stackrel{*}{\Rightarrow} \epsilon$ par rapport à la chaîne $w\}$ (*i.e.*, toute chaîne pouvant se réécrire en ϵ au moyen de la RCG G appartient à $L(G)$).*

En guise d'illustration, considérons la RCG suivante[9] :
$G = \langle \{S, A, B\}, \{a, b\}, \{X, Y, Z\}, S, P \rangle$ avec $P = \{S(X\,Y\,Z) \rightarrow A(X, Z)\,B(Y)$, $A(a\,X, a\,Y) \rightarrow A(X, Y)$, $B(b\,X) \rightarrow B(X)$, $A(\epsilon, \epsilon) \rightarrow \epsilon$, $B(\epsilon) \rightarrow \epsilon\}$.

$L(G) = \{a^n b^k a^n \mid k, n \in \mathbb{N}\}$. Prenons $w = aabaa$. La dérivation commence par $S(\langle 0, 5 \rangle)$. En premier, nous appliquons l'instanciation de clause suivante :

$$\begin{array}{ccccccccc} S(X & & Y & & Z) & \rightarrow & A(X, & Z) & B(Y) \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & \downarrow & \downarrow \\ \langle 0,2 \rangle & & \langle 2,3 \rangle & & \langle 3,5 \rangle & & \langle 0,2 \rangle & \langle 3,5 \rangle & \langle 2,3 \rangle \\ aa & & b & & aa & & aa & aa & b \end{array}$$

Avec cette instanciation, $S(\langle 0, 5 \rangle) \Rightarrow A(\langle 0, 2 \rangle, \langle 3, 5 \rangle)B(\langle 2, 3 \rangle)$. Ensuite

$$\begin{array}{ccccc} B(b & & X) & \rightarrow & B(X) \\ \downarrow & & \downarrow & & \downarrow \\ \langle 2,3 \rangle & & \langle 3,3 \rangle & & \langle 3,3 \rangle \\ b & & \epsilon & & \epsilon \end{array} \quad \text{et } B(\epsilon) \rightarrow \epsilon$$

mènent à $A(\langle 0, 2 \rangle, \langle 3, 5 \rangle)B(\langle 2, 3 \rangle) \Rightarrow A(\langle 0, 2 \rangle, \langle 3, 5 \rangle)B(\langle 3, 3 \rangle) \Rightarrow A(\langle 0, 2 \rangle, \langle 3, 5 \rangle)$.

$$\begin{array}{ccccccccc} A(a & & X, & & a & & Y) & \rightarrow & A(X, & Y) \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & \downarrow \\ \langle 0,1 \rangle & & \langle 1,2 \rangle & & \langle 3,4 \rangle & & \langle 4,5 \rangle & & \langle 1,2 \rangle & \langle 4,5 \rangle \\ a & & a & & a & & a & & a & a \end{array}$$

mène à $A(\langle 0, 2 \rangle, \langle 3, 5 \rangle) \Rightarrow A(\langle 1, 2 \rangle, \langle 4, 5 \rangle)$. Alors

9. Notons que cette RCG traite les portions $a^n ... a^n$ comme étant une copie l'une de l'autre alors qu'une grammaire similaire à celle-ci à l'exception de la clause de membre gauche A(aX, aY) qui serait remplacée par la clause A(aX, Ya) -> A(X, Y) définirait le même langage, mais traiterait $a^n ... a^n$ comme une chaîne bien parenthésée.

$$\begin{array}{cccccccc} A(a & X, & a & Y) & \rightarrow & A(X, & Y) & \\ \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \\ \langle 1,2\rangle & \langle 2,2\rangle & \langle 4,5\rangle & \langle 5,5\rangle & & \langle 2,2\rangle & \langle 5,5\rangle & \\ a & \epsilon & a & \epsilon & & \epsilon & \epsilon & \end{array} \qquad \text{et } A(\epsilon,\epsilon) \rightarrow \epsilon$$

mènent à $A(\langle 1,2\rangle, \langle 4,5\rangle) \Rightarrow A(\langle 2,2\rangle, \langle 5,5\rangle) \Rightarrow \epsilon$

Définition 12 (RCG simple)

Une RCG est :

– non combinatorielle *si chacun des arguments d'un prédicat du membre droit d'une clause est constitué d'une unique variable ;*

– linéaire *s'il n'y a pas de variable apparaissant plus d'une fois dans le membre de gauche ou de droite d'une clause ;*

– non effaçante *si chaque variable apparaissant dans le membre de gauche d'une clause apparaît aussi dans son membre de droite et vice versa ;*

– simple *si elle est non combinatorielle, linéaire et non effaçante.*

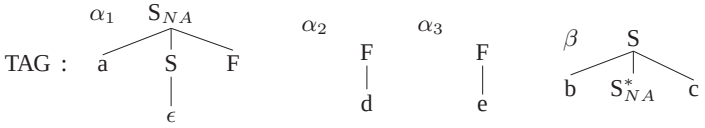
Les RCG simples et les systèmes de réécriture non contextuels linéaires (LCFRS, (Weir, 1988)) sont équivalents (voir (Boullier, 1998)). En conséquence, les RCG simples sont légèrement sensibles au contexte (Joshi, 1985).

Pour les RCG simples, différents algorithmes d'analyse ont été proposés : l'algorithme *top-down* de (Boullier, 2000) qui peut gérer même des RCG non simples, les algorithmes CYK introduits dans (Burden et Ljunglöf, 2005) pour les grammaires non contextuelles multiples (MCFG, (Seki *et al.*, 1991)), un formalisme équivalent aux RCG simples, et les *Thread Automata* pour RCG simples introduits dans (Villemonte de La Clergerie, 2002). Cette dernière approche représente un algorithme incrémental de style *Earley*.

3.2. *Convertir une TAG en RCG*

À présent, nous décrivons l'idée générale de la transformation d'une TAG en RCG, en suivant l'algorithme de (Boullier, 1999) : la RCG contient des prédicats $\langle\alpha\rangle(X)$ et $\langle\beta\rangle(L,R)$ respectivement associés aux arbres initiaux et auxiliaires. X couvre la production de α et tous les arbres insérés dans α, alors que L et R couvrent les parties de la production de β (incluant tous les arbres insérés dans β) situées respectivement à gauche et à droite du nœud pied de β. Les clauses de la RCG « consomment » les arguments de ces prédicats en identifiant les parties qui proviennent de l'arbre élémentaire α (respectivement β) lui-même et celles qui proviennent de l'un des arbres

élémentaires insérés par substitution ou adjonction. Cette transformation est illustrée par la figure 4[10].



RCG équivalente :
$S(X) \rightarrow \langle\alpha_1\rangle(X) \qquad S(X) \rightarrow \langle\alpha_2\rangle(X) \qquad S(X) \rightarrow \langle\alpha_3\rangle(X)$
(chaque mot du langage appartient à une production d'un $\alpha \in I$)
$\langle\alpha_1\rangle(aF) \rightarrow \langle\alpha_2\rangle(F) \qquad \langle\alpha_1\rangle(aF) \rightarrow \langle\alpha_3\rangle(F)$
(la production de α_1 est a suivie de l'arbre qui se substitue à F)
$\langle\alpha_1\rangle(aB_1B_2F) \rightarrow \langle\beta\rangle(B_1, B_2)\langle\alpha_2\rangle(F) \qquad \langle\alpha_1\rangle(aB_1B_2F) \rightarrow \langle\beta\rangle(B_1, B_2)\langle\alpha_3\rangle(F)$
(si β s'adjoint à S dans α_1 ; alors la production est a suivie par la partie gauche de β, la partie droite de β et l'arbre substitué à F)
$\langle\beta\rangle(B_1b, cB_2) \rightarrow \langle\beta\rangle(B_1, B_2)$
(β peut s'adjoindre à sa racine ; alors la partie gauche est la partie gauche du β adjoint suivie de b ; la partie droite est c suivie de la partie droite du β adjoint)
$\langle\alpha_2\rangle(d) \rightarrow \epsilon \quad \langle\alpha_3\rangle(e) \rightarrow \epsilon \quad \langle\beta\rangle(b, c) \rightarrow \epsilon$
(la production de α_2, α_3 et β peut être d, e et la paire b (gauche) et c (droite) respectivement)

Figure 4. *Une TAG exemple et une RCG associée*

4. TT-MCTAG

Cette section définit les TT-MCTAG, introduites par (Lichte, 2007).

Définition 13 (TT-MCTAG)

Une MCTAG avec tuples d'arbres et nœuds partagés (TT-MCTAG) *est un quintuple* $G = \langle N, T, I, A, S, \mathcal{A}\rangle$ *tel que* $\langle N, T, I, A, S\rangle$ *est une TAG, et* $\mathcal{A}$ *est une partition de* $\mathcal{P}(I \cup A)$[11]*, un ensemble des ensembles d'arbres élémentaires tel que chaque* $\Gamma \in \mathcal{A}$ *est de la forme* $\{\gamma, \beta_1, \ldots, \beta_n\}$ *où* γ *est un arbre ayant au moins un nœud étiqueté par un symbole terminal, l'arbre* tête*, et* $\beta_1, \ldots, \beta_n$ *sont des arbres auxiliaires, les arbres* arguments*. Nous notons un tel ensemble sous forme de tuple* $\langle\gamma, \{\beta_1, \ldots, \beta_n\}\rangle$.

Dans ce qui suit, nous notons, pour un arbre argument β donné, $h(\beta)$ la tête de β. Pour un arbre $\gamma \in I \cup A$ donné, $a(\gamma)$ représente l'ensemble des arbres arguments

10. L'étiquette NA sur un nœud spécifie que ce nœud est non adjoignable. En cas d'absence d'étiquette, le nœud est considéré comme optionnellement adjoignable.

11. Notons que cette définition n'exclut pas qu'un même arbre apparaisse dans différents ensembles ou même plusieurs fois dans le même ensemble. Dans ce cas, nous considérons qu'il y a différents arbres de même structure (*i.e.*, qu'ils sont isomorphes, avec des étiquettes identiques).

de γ s'il y en a, l'ensemble vide dans le cas contraire. De plus, pour une TT-MCTAG G donnée, $H(G)$ est l'ensemble des arbres têtes et $A(G)$ l'ensemble des arbres arguments. Enfin, nous appelons un nœud d'étiquette A, un A-nœud.

Définition 14 (Dérivation TT-MCTAG)

Soit G une TT-MCTAG. Un arbre de dérivation D dans la TAG sous-jacente G_{TAG} est valide pour G si et seulement si $D = \langle V, E, r\rangle$ satisfait les conditions suivantes :

– **(MC)** *(Multicomponent Condition) Si γ_1, γ_2 font partie du même tuple,*

alors $|\{v \mid v \in V$ et $l(v) = \gamma_1\}| = |\{v \mid v \in V$ et $l(v) = \gamma_2\}|$. Cette condition stipule que tous les arbres d'un tuple doivent être utilisés dans une dérivation ;

– **(SN-TTL)** *(Tree-Tuple Locality with Shared Nodes) Pour tout $\beta \in A(G)$:*
*Si $v_1, \ldots, v_n \in V$ sont distincts deux à deux avec $l(v_i) = h(\beta)$ pour $1 \leq i \leq n$, alors il existe des $u_1, \ldots, u_n \in V$ deux à deux distincts avec $l(u_i) = \beta$ pour $1 \leq i \leq n$ tels que pour tout $i, 1 \leq i \leq n$: soit $\langle v_i, u_i\rangle \in E$ ou alors il y a $u_{i,1}, \ldots, u_{i,k}$ avec des étiquettes d'arbres auxiliaires tels que $u_i = u_{i,k}$, $\langle v_i, u_{i,1}\rangle \in E$ et pour $1 \leq j \leq k-1$: $\langle u_{i,j}, u_{i,j+1}\rangle \in E$ et $g(\langle u_{i,j}, u_{i,j+1}\rangle) = \epsilon$ (*i.e.*, adjonction racine). Cette condition stipule qu'un arbre argument doit être lié indirectement à l'arbre tête de son tuple, et que dans l'arbre de dérivation, le chemin partant de la tête et représentant ce lien est constitué d'une opération quelconque suivie d'adjonctions à la racine.*

Voir la figure 1 pour une dérivation TT-MCTAG.

Notons que, dans le cas général, le problème de la reconnaissance universelle pour TT-MCTAG est NP-difficile (Søgaard *et al.*, 2007). Aussi, à ce stade, on ne sait pas encore si la complexité d'analyse pour les langages TT-MCTAG est polynomiale par rapport à la chaîne d'entrée. Dans ce qui suit, nous définissons une limitation de TT-MCTAG, limitation fondée sur une suggestion de (Søgaard *et al.*, 2007) : TT-MCTAG est de rang k si, à un moment donné au cours de la dérivation, au plus k arbres arguments dépendent d'arbres têtes actuellement utilisés dans la dérivation (notion d'argument en attente d'adjonction). Même pour TT-MCTAG de rang k, le problème de la reconnaissance universelle est très probablement toujours NP-difficile. Mais, comme nous allons le montrer dans la suite de cet article, la complexité d'analyse des langages générés par k-TT-MCTAG est polynomiale par rapport à la chaîne d'entrée.

Définition 15 (k-TT-MCTAG)

*Soit G une TT-MCTAG. G est de rang k (ou encore est une k-*TT-MCTAG*) si et seulement si pour chaque arbre de dérivation $D = \langle V, E, r\rangle$ défini pour G, la condition suivante est valide :*
(TT-k) *Il n'existe pas de v tel que*

$$\Sigma_{\beta \in A(G)}(|\{v' \mid v' \text{ est un } \beta\text{-nœud avec } \langle v, v'\rangle \in E^+\}| - |\{v' \mid v' \text{ est un } h(\beta)\text{-nœud avec } \langle v, v'\rangle \in E^*\}|) > k$$

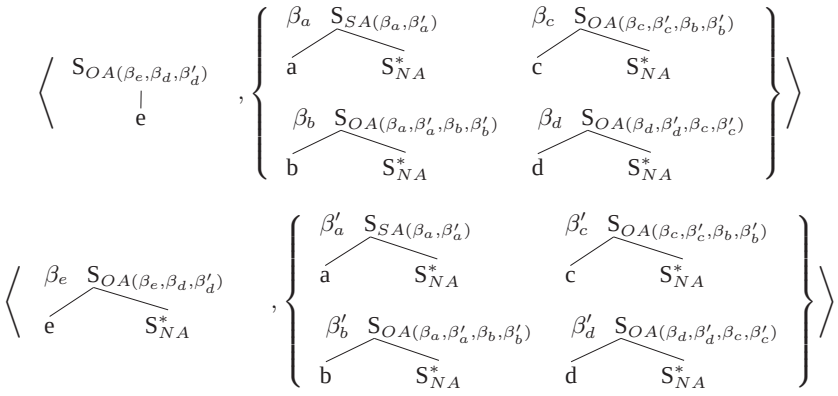


Figure 5. *TT-MCTAG pour* $\{a^n b^n c^n d^n e^n \mid n \geq 1\}$

Avec les analyses proposées dans (Lichte, 2007), cela conduit à un arbre de dérivation dont les nœuds sont binaires, et axé sur une ligne de projection verbale. La signification linguistique de cette restriction est grossièrement que pour chaque nœud VP sur la ligne de projection verbale, au plus k NPs peuvent être enchâssés (notion de *scrambling*) autour de ce nœud. Il est difficile de dire si une telle restriction est valide empiriquement. Cependant, il convient de noter que le nombre de verbes autorisant un enchâssement non local de leurs arguments est limité. De plus, le nombre d'arguments de ces verbes est fixé. Ceci semble indiquer que la limite k existe, bien qu'elle soit plus motivée par des raisons sémantiques et pragmatiques que par des raisons syntaxiques.

Comme chaque TAG peut être lexicalisée et comme chaque LTAG est une TT-MCTAG sans argument, nous obtenons immédiatement que $\mathcal{L}(\text{TAG}) \subseteq \mathcal{L}(k\text{-TT-MCTAG})$ pour chaque k. De plus $\mathcal{L}(\text{TAG}) \subset \mathcal{L}(\text{TT-MCTAG})$. La figure 5 donne un exemple de TT-MCTAG qui génère un langage ne pouvant pas être généré par une TAG, ainsi $\mathcal{L}(\text{TAG}) \neq \mathcal{L}(\text{TT-MCTAG})$.

5. Convertir une k-TT-MCTAG en RCG

5.1. *Idée principale*

Nous construisons une RCG simple équivalente à une k-TT-MCTAG de manière similaire à la transformation de TAG vers RCG. Nous utilisons des prédicats $\langle\gamma\rangle$ pour les arbres élémentaires (et non les ensembles d'arbres) caractérisant la contribution de γ. Rappelons que chaque TT-MCTAG est aussi une TAG, une dérivation TT-MCTAG est une dérivation dans la TAG sous-jacente (c'est ainsi que nous avons défini TT-MCTAG). En conséquence de quoi, nous pouvons construire la RCG pour la TAG sous-jacente, en enrichissant les prédicats de manière à permettre une représentation des arbres arguments qui doivent encore être adjoints et ainsi contraindre les

clauses RCG utilisables dans une dérivation. Dans ce cas, la production d'un prédicat correspondant à un arbre γ contient non seulement γ et ses arguments mais aussi les arguments des prédicats qui sont plus haut dans l'arbre de dérivation et qui sont adjoints sous γ par partage de nœud[12].

Notre construction conduit à une RCG d'arité 2 avec noms de prédicats complexes. Afin de conserver un nombre fini de prédicats, la limite k est cruciale.

Un prédicat $\langle\gamma\rangle$ doit encoder l'ensemble des arbres arguments qui dépendent d'un arbre tête rencontré précédemment et qui doivent encore être adjoints. Nous appelons cet ensemble la liste des arguments en attente (*list of pending arguments*, LPA). Ces arbres doivent soit être adjoints à la racine ou bien passés à la LPA de l'arbre adjoint à la racine. La LPA est un ensemble dont les éléments appartiennent à l'ensemble des parties de A (arbres auxiliaires arguments) puisque nous autorisons plusieurs occurrences d'un même arbre.

Afin de réduire le nombre de clauses, nous distinguons les clauses d'arbres (prédicats de la forme $\langle\gamma...\rangle$) des clauses de branchements (prédicats de la forme $\langle adj...\rangle$ et $\langle sub...\rangle$) comme dans (Boullier, 1999). Nous avons donc trois types de prédicats :

1) $\langle\gamma, LPA\rangle$ avec LPA la liste des arguments en attente, provenant d'arbres têtes rencontrés précédemment (sans les arguments de γ). Ce prédicat a une arité de deux si γ est un arbre auxiliaire, et une arité de un sinon, comme nous allons le voir dans l'algorithme de conversion ci-dessous.

Les clauses $\langle\gamma, LPA\rangle$ distribuent les variables de la production des arbres qui se substituent ou s'adjoignent dans γ aux prédicats sub et adj correspondants. De plus, elles transmettent la LPA aux prédicats adj en position racine et distribuent les arguments de γ aux LPAs de tous les prédicats adj.

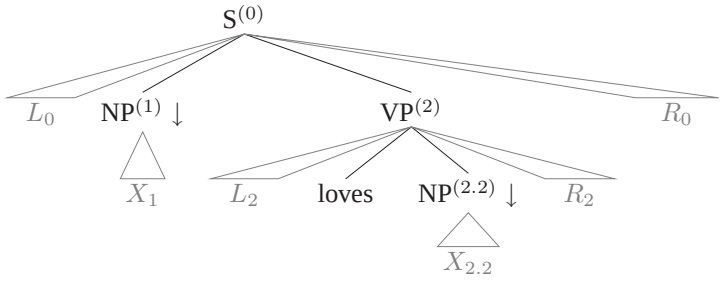
2) $\langle adj, \gamma, dot, LPA\rangle$ est un prédicat intermédiaire (d'arité deux). Ici, LPA contient a) la liste des arguments rencontrés précédemment si $dot = \epsilon$, et b) les arguments de γ. Nous prenons comme condition qu'elle ne contienne que les arbres qui peuvent s'adjoindre en position dot à γ.

Les clauses $\langle adj, \gamma, dot, LPA\rangle$ adjoignent un arbre γ' à la position dot dans γ. Si $\gamma' \in LPA$, alors le nouveau prédicat reçoit $LPA \setminus \{\gamma'\}$. Sinon, γ' doit être une tête et LPA est transmis tel quel.

3) $\langle sub, \gamma, dot\rangle$ est un prédicat intermédiaire (d'arité un).

Les clauses $\langle sub, \gamma, dot\rangle$ substituent un arbre γ' à la position dot dans γ.

12. Une alternative serait de considérer seulement γ et ses arguments comme la production de γ. Ce qui mènerait à une RCG avec des noms de prédicats plus simples (les arguments en attente ne seraient pas nécessaires) mais avec des prédicats d'arité plus grande puisque la contribution de γ peut être discontinue : chaque argument d'une tête rencontrée précédemment, et s'adjoignant sous γ, interrompt la contribution de γ. Cette construction est plus complexe que celle que nous exposons ici.



Chaîne de décoration : $L_0\, X_1\, L_2\, loves\, X_{2.2}\, R_2\, R_0$

Figure 6. *Exemple d'arbre TAG et de chaîne de décoration associée*

Pour une description plus précise de l'algorithme de construction, nous avons besoin de la notion de chaîne de décoration σ_γ d'un arbre élémentaire γ telle que définie par (Boullier, 1999) :

Définition 16 (Chaîne de décoration)

Pour chaque arbre élémentaire $\gamma = \langle V, E, r\rangle$: pour chaque nœud $v \in V$ vérifiant $f_{SA}(v) \neq \emptyset$ (nœud d'adjonction), on définit les variables L_v et R_v, et pour chaque nœud de substitution $v \in V$, on définit la variable X_v. Ces variables L_v, R_v et X_v réfèrent à des intervalles de la chaîne d'entrée, i.e., *à des étiquettes des nœuds feuilles de l'arbre dérivé qui sont dominés par les nœuds auxquels ces variables sont associées.*

La chaîne de décoration *σ_γ de γ est alors obtenue* via *un parcours descendant de gauche à droite de γ (parcours en profondeur). Lors de ce parcours, chaque nœud interne est rencontré deux fois (descente et montée), et chaque nœud feuille une fois. On collecte les variables L_v lorsque v est traversé en descendant, R_v lorsque v est traversé en remontant (pour les nœuds d'adjonction), X_v lorsque v est un nœud de substitution et $l(v)$ lorsque v est une feuille telle que $l(v) \in T \cup \{\epsilon\}$. De plus, lors du passage dans un nœud pied, une virgule (« , ») est insérée, séparant ainsi la contribution gauche de la contribution droite d'un arbre auxiliaire. Un exemple de chaîne de décoration est donné en figure 6.*

5.2. *Algorithme de transformation*

À partir de ces définitions, nous pouvons décrire la construction d'une 2-RCG simple à partir d'une k-TT-MCTAG :

1) On crée un prédicat de départ S et des clauses $S(X) \rightarrow \langle \alpha, \emptyset \rangle(X)$ pour chaque $\alpha = \langle V, E, r \rangle \in I$ avec $l(r) = S$.

2) Pour chaque $\gamma \in I \cup A$: soient L_p, R_p les symboles représentant les contributions gauche et droite dans σ_γ du nœud en position p (si ce nœud n'est pas un nœud de substitution). Soit X_p le symbole représentant la contribution du nœud en position p (si ce nœud est un nœud de substitution).

Nous supposons que $p_1, \ldots, p_k$ sont les nœuds d'adjonction de γ, et $p_{k+1}, \ldots, p_l$ ses nœuds de substitution. Alors la RCG résultant de la transformation contient toutes les clauses de la forme :

$$\langle \gamma, LPA \rangle(\sigma_\gamma) \rightarrow \langle adj, \gamma, p_1, LPA_{p_1} \rangle(L_{p_1}, R_{p_1}) \ldots \langle adj, \gamma, p_k, LPA_{p_k} \rangle(L_{p_k}, R_{p_k})$$
$$\langle sub, \gamma, p_{k+1} \rangle(X_{p_{k+1}}) \ldots \langle sub, \gamma, p_l \rangle(X_{p_l})$$

telles que (on rappelle, *cf.* section 5.1, que les LPA sont des ensembles)

- Si $LPA \neq \emptyset$, alors $\epsilon \in \{p_1, \ldots, p_k\}$ et $LPA \subseteq LPA_\epsilon$, et
- $\bigcup_{i=0}^{k} LPA_{p_i} = LPA \cup a(\gamma)$.

3) Pour chaque prédicat $\langle adj, \gamma, dot, LPA \rangle$ la RCG contient toutes les clauses $\langle adj, \gamma, dot, LPA \rangle(L, R) \rightarrow \langle \gamma', LPA' \rangle(L, R)$ telles que γ' peut être adjoint en position dot dans γ et

- soit $\gamma' \in LPA$ et $LPA' = LPA \setminus \{\gamma'\}$,
- soit $\gamma' \notin LPA$, γ' est une tête (*i.e.*, un arbre tête), et $LPA' = LPA$.

4) Pour chaque prédicat $\langle adj, \gamma, dot, \emptyset \rangle$ où dot dans γ n'est pas un nœud d'adjonction obligatoire (appelé *OA-node*), la RCG contient une clause $\langle adj, \gamma, dot, \emptyset \rangle(\epsilon, \epsilon) \rightarrow \epsilon$.

5) Pour chaque prédicat $\langle sub, \gamma, dot \rangle$ et chaque γ' pouvant être substitué en position dot dans γ, la RCG contient une clause $\langle sub, \gamma, dot \rangle(X) \rightarrow \langle \gamma', \emptyset \rangle(X)$.

Notons que cet algorithme de transformation (i) a une complexité exponentielle par rapport au nombre maximal d'arbres auxiliaires pouvant s'adjoindre à un nœud d'une certaine catégorie syntaxique, dans la mesure où l'algorithme calcule toutes les évolutions possibles de la LPA, et (ii) cependant il termine car la liste des arguments en attente est contrainte en termes de taille maximale.

Pour illustrer cet algorithme, considérons la TT-MCTAG de la figure 7. Pour cette TT-MCTAG, nous obtenons (entre autres) les clauses RCG suivantes :

– $\langle \alpha_v, \emptyset \rangle(L\ v_0\ R) \rightarrow \langle adj, \alpha_v, \epsilon, \emptyset \rangle(L, R)$
(une seule adjonction au nœud racine d'adresse ϵ)

– $\langle adj, \alpha_v, \epsilon, \emptyset \rangle(L, R) \rightarrow \langle \beta_{v_1}, \emptyset \rangle(L, R) \mid \langle \beta_{v_2}, \emptyset \rangle(L, R)$
(β_{v_1} ou β_{v_2} peut être adjoint en ϵ dans α_v, la LPA (ici vide) est transmise au membre

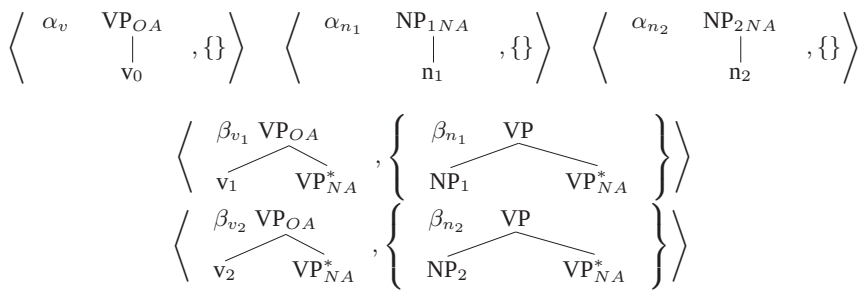


Figure 7. *TT-MCTAG*

de droite)

– $\langle \beta_{v_1}, \emptyset \rangle (L\ v_1, R) \rightarrow \langle adj, \beta_{v_1}, \epsilon, \{\beta_{n_1}\} \rangle (L, R)$
(dans β_{v_1}, il n'y a qu'un seul site d'adjonction d'adresse ϵ ; l'argument est transmis à la nouvelle LPA)

– $\langle adj, \beta_{v_1}, \epsilon, \{\beta_{n_1}\} \rangle (L, R) \rightarrow$
$\langle \beta_{n_1}, \emptyset \rangle (L, R)\ \mid\ \langle \beta_{v_1}, \{\beta_{n_1}\} \rangle (L, R)\ \mid\ \langle \beta_{v_2}, \{\beta_{n_1}\} \rangle (L, R)$
(soit β_{n_1} est adjoint et retiré de la LPA, soit un autre arbre (β_{v_1} ou β_{v_2}) est adjoint ; dans ce cas, la LPA reste inchangée)

– $\langle \beta_{v_1}, \{\beta_{n_1}\} \rangle (L\ v_1, R) \rightarrow \langle adj, \beta_{v_1}, \epsilon, \{\beta_{n_1}, \beta_{n_1}\} \rangle (L, R)$
(là encore, il y a un seul site d'adjonction dans β_{v_1} ; l'argument β_{n_1} est ajouté à la LPA)

– $\langle \beta_{n_1}, \emptyset \rangle (L\ X, R) \rightarrow \langle adj, \beta_{n_1}, \epsilon, \emptyset \rangle (L, R)\ \langle sub, \beta_{n_1}, 1, \emptyset \rangle (X)$
(adjonction à la racine et substitution en 1 dans β_{n_1})

– $\langle adj, \beta_{n_1}, \epsilon, \emptyset \rangle (\epsilon, \epsilon) \rightarrow \epsilon$
(adjonction à la racine de β_{n_1} non obligatoire, tant que la LPA est vide)

– $\langle sub, \beta_{n_1}, 1, \emptyset \rangle (X) \rightarrow \langle \alpha_{n_1}, \emptyset \rangle (X)$
(substitution de α_{n_1} à l'adresse 1)

– $\langle \alpha_{n_1}, \emptyset \rangle (n_1) \rightarrow \epsilon$
(pas d'adjonction ou de substitution dans α_{n_1})

...

5.3. *Analyse de la dérivation pour la RCG résultant de la transformation*

Prenons la chaîne d'entrée $n_1 n_2 n_1 v_2 v_1 v_1 v_0$ et la RCG calculée à partir de la TT-MCTAG de la figure 7. La dérivation RCG se déroule comme suit[13].

13. Dans cet exemple, nous remplaçons les intervalles de la RCG avec les portions correspondantes de la chaîne d'entrée afin de rendre l'exemple plus lisible.

$S(n_1\ n_2\ n_1\ v_2\ v_1\ v_1\ v_0)$
$\Rightarrow \langle \alpha_v, \emptyset \rangle (n_1\ n_2\ n_1\ v_2\ v_1\ v_1\ v_0)$
$\Rightarrow \langle adj, \alpha_v, \epsilon, \emptyset \rangle (n_1\ n_2\ n_1\ v_2\ v_1\ v_1, \epsilon)$ (adjonction en ϵ, analyse de v_0)
$\Rightarrow \langle \beta_{v_1}, \emptyset \rangle (n_1\ n_2\ n_1\ v_2\ v_1\ v_1, \epsilon)$ (adjonction de β_{v_1})
$\Rightarrow \langle adj, \beta_{v_1}, \epsilon, \{\beta_{n_1}\} \rangle (n_1\ n_2\ n_1\ v_2\ v_1, \epsilon)$ (adj. en ϵ, v_1 analysé, β_{n_1} ajouté à la LPA)
$\Rightarrow \langle \beta_{v_1}, \{\beta_{n_1}\} \rangle (n_1\ n_2\ n_1\ v_2\ v_1, \epsilon)$ (adjonction de β_{v_1})
$\Rightarrow \langle adj, \beta_{v_1}, \epsilon, \{\beta_{n_1}, \beta_{n_1}\} \rangle (n_1\ n_2\ n_1\ v_2, \epsilon)$ (adj. en ϵ, v_1 analysé, β_{n_1} ajouté à la LPA)
$\Rightarrow \langle \beta_{v_2}, \{\beta_{n_1}, \beta_{n_1}\} \rangle (n_1\ n_2\ n_1\ v_2, \epsilon)$ (adjonction de β_{v_2})
$\Rightarrow \langle adj, \beta_{v_2}, \epsilon, \{\beta_{n_2}, \beta_{n_1}, \beta_{n_1}\} \rangle (n_1\ n_2\ n_1, \epsilon)$ (adj. en ϵ, v_2 analysé, β_{n_2} ajouté à la LPA)
$\Rightarrow \langle \beta_{n_1}, \{\beta_{n_2}, \beta_{n_1}\} \rangle (n_1\ n_2\ n_1, \epsilon)$ (adjonction de β_{n_1} issu de la LPA)
$\Rightarrow \langle adj, \beta_{n_1}, \epsilon, \{\beta_{n_2}, \beta_{n_1}\} \rangle (n_1\ n_2, \epsilon)\ \langle sub, \beta_{n_1}, 1, \rangle (n_1)$ (adj. en ϵ, sub. en 1)
$\Rightarrow \langle adj, \beta_{n_1}, \epsilon, \{\beta_{n_2}, \beta_{n_1}\} \rangle (n_1\ n_2, \epsilon)\ \langle \alpha_{n_1}, \emptyset \rangle (n_1)$ (substitution de α_{n_1})
$\Rightarrow \langle adj, \beta_{n_1}, \epsilon, \{\beta_{n_2}, \beta_{n_1}\} \rangle (n_1\ n_2, \epsilon)\ \epsilon$ (analyse de n_1)
$\Rightarrow \langle \beta_{n_2}, \{\beta_{n_1}\} \rangle (n_1\ n_2, \epsilon)$ (adjonction de β_{n_2} issu de la LPA)
$\Rightarrow \langle adj, \beta_{n_2}, \epsilon, \{\beta_{n_1}\} \rangle (n_1, \epsilon)\ \langle sub, \beta_{n_2}, 1, \rangle (n_2)$ (adjonction en ϵ, substitution en 1)
$\Rightarrow \langle adj, \beta_{n_2}, \epsilon, \{\beta_{n_1}\} \rangle (n_1, \epsilon)\ \langle \alpha_{n_2}, \emptyset \rangle (n_2)$ (substitution de α_{n_2})
$\Rightarrow \langle adj, \beta_{n_2}, \epsilon, \{\beta_{n_1}\} \rangle (n_1, \epsilon)\ \epsilon$ (analyse de n_2)
$\Rightarrow \langle \beta_{n_1}, \emptyset \rangle (n_1, \epsilon)$ (adjonction de β_{n_1} issu de la LPA)
$\stackrel{*}{\Rightarrow} \langle adj, \beta_{n_1}, \epsilon, \emptyset \rangle (\epsilon, \epsilon)\ \langle \alpha_{n_1}, \emptyset \rangle (n_1)$ (substitution de α_{n_1})
$\stackrel{*}{\Rightarrow} \epsilon$ (analyse de n_1)

Notons que cet exemple nécessite une LPA de taille maximale 3, *i.e.*, une 3-TT-MCTAG.

Notons, en outre, qu'avec cette construction, le regroupement des arbres en ensembles est perdu. En effet, dans notre exemple, nous ne savons pas quels sont les arbres n_1 et v_1 provenant initialement du même ensemble. Cependant, dans notre implantation, nous construisons la RCG uniquement pour la TT-MCTAG sélectionnée par la chaîne d'entrée. Aussi, si un symbole terminal apparaît plus d'une fois dans la chaîne d'entrée, nous utilisons différentes occurrences des tuples d'arbres qui lui sont associés. Ainsi, nous évitons l'utilisation du même arbre élémentaire plusieurs fois, et le regroupement des arbres peut être inféré à partir de l'identifiant de tuple présent dans les noms des arbres manipulés.

À partir de la construction donnée ci-dessus, nous pouvons démontrer le théorème suivant.

Théorème 1

Pour chaque k-TT-MCTAG G, il existe une RCG simple G' telle que $L(G) = L(G')$.

Puisque $\mathcal{L}(\text{CFG}) \subset \mathcal{L}(\text{TAG}) \subseteq \mathcal{L}(k\text{-TT-MCTAG})$ pour chaque k, nous obtenons comme corollaire que les langages de chaînes générés par une k-TT-MCTAG sont légèrement sensibles au contexte. L'inverse de ce théorème n'est pas vrai puisque, commme nous allons le montrer en section 6, pour chaque k-TT-MCTAG on peut

même construire une TAG équivalente. Par conséquent, les k-TT-MCTAG génèrent exactement les mêmes langages que les TAG. L'ensemble des langages de chaînes générés par les TAG est strictement contenu dans l'ensemble des langages de chaînes générés par les RCG simples.

Pour démontrer le théorème ci-dessus, nous introduisons la notion d'arbre de dérivation TT-RCG. Ces arbres sont obtenus à partir de la dérivation RCG en remplaçant chaque prédicat d'arbre élémentaire ($\langle \gamma, LPA \rangle$) par un nœud et chaque prédicat de branchement ($\langle adj|sub, \gamma, dot, LPA \rangle$) par une arête. De plus, les productions des prédicats sont ajoutées aux étiquettes $\langle \gamma, LPA \rangle$.

Définition 17 (Arbre de dérivation TT-RCG)

Soit G' une RCG construite à partir d'une k-TT-MCTAG G comme précédemment.

1) S'il existe une clause $\langle \gamma, LPA \rangle(w) \rightarrow \epsilon$ (ou $\langle \gamma, LPA \rangle(w_1, w_2) \rightarrow \epsilon$ respectivement), alors $\langle \{v\}, \emptyset, v \rangle$ avec $l(v) = \langle \gamma, LPA, w \rangle$ (ou $l(v) = \langle \gamma, LPA, w_1, w_2 \rangle$ respectivement) est un arbre de dérivation TT-RCG dans G'.

2) Pour chaque clause

$\langle \gamma, LPA \rangle(\sigma_\gamma) \rightarrow \langle adj, \gamma, p_1, LPA_{p_1} \rangle(L_{p_1}, R_{p_1}) \ldots$

$\ldots \langle adj, \gamma, p_k, LPA_{p_k} \rangle(L_{p_k}, R_{p_k}) \langle sub, \gamma, p_{k+1} \rangle(X_{p_{k+1}}) \ldots \langle sub, \gamma, p_l \rangle(X_{p_l})$

et chaque $P_{adj} \subseteq \{p_1, \ldots, p_k\}$:

Si

- il existe une clause $\langle adj, \gamma, p, \emptyset \rangle(\epsilon, \epsilon) \rightarrow \epsilon$ pour toute position $p \in \{p_1, \ldots, p_k\} \setminus P_{adj}$,

- pour toute position $p \in P_{adj}$, il existe une clause $\langle adj, \gamma, p, LPA_p \rangle(L, R) \rightarrow \langle \gamma_p, LPA'_p \rangle(L, R)$ et il existe un arbre de dérivation TT-RCG $D_p = \langle V_p, E_p, r_p \rangle$ avec $l(r_p) = \langle \gamma, LPA_p, w^l_p, w^r_p \rangle$ pour $w^l_p, w^r_p \in T^*$, et

- pour toute position $p \in P_{subst}$ telle que $P_{subst} = \{p_{k+1}, \ldots, p_l\}$ il existe une clause $\langle sub, \gamma, p \rangle(X) \rightarrow \langle \gamma_p, \emptyset \rangle(X)$ et il existe un arbre de dérivation TT-RCG $D_p = \langle V_p, E_p, r_p \rangle$ tel que $l(r_p) = \langle \gamma, w_p \rangle$ pour $w_p \in T^*$

alors il existe un arbre de dérivation TT-RCG $\langle V, E, r \rangle$ dans G' tel que

- $r \notin \bigcup_{p \in P_{adj} \cup P_{subst}} V_p$;

- $V = \bigcup_{p \in P_{adj} \cup P_{subst}} V_p \cup \{r\}$;

- $E = \bigcup_{p \in P_{adj} \cup P_{subst}} E_p \cup \{\langle r, r_p \rangle \mid p \in P_{adj} \cup P_{subst}\}$;

- $l(r) = \langle \gamma, LPA', f(\sigma_\gamma) \rangle$ avec $LPA' = LPA \cup a(\gamma)$),

f étant une instanciation des clauses de membre gauche $\langle \gamma, LPA \rangle$ telle que $f(L_p) = w^l_p, f(R_p) = w^r_p$ pour toute $p \in P_{adj}$, $f(X_p) = w_p$ pour toute position $p \in P_{subst}$ et $f(L_p) = f(R_p) = \epsilon$ pour toute position $p \in \{p_1, \ldots, p_k\} \setminus P_{adj}$;

- $g(\langle r, r_p \rangle) = p$ pour toute position $p \in P_{adj} \cup P_{subst}$.

3) Rien d'autre n'est un arbre de dérivation TT-RCG dans G.

La figure 8 montre un exemple d'arbre de dérivation TT-RCG. Il s'agit de celui correspondant à la dérivation RCG pour la chaîne $w = n_1n_2n_1v_2v_1v_1v_0$, donnée précédemment.

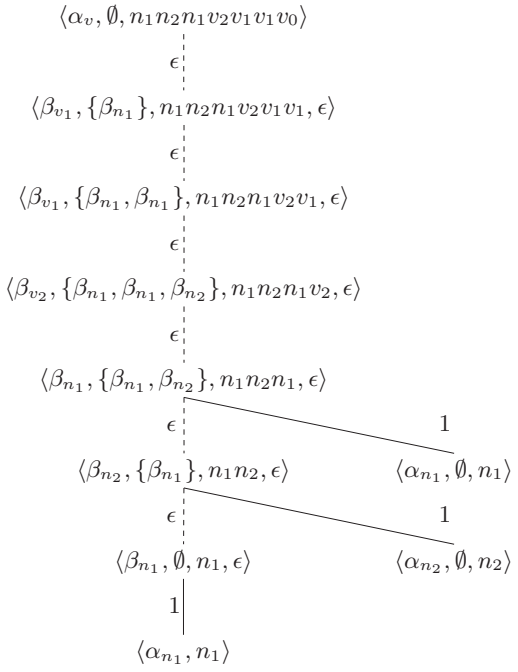


Figure 8. *Un arbre de dérivation TT-RCG pour la RCG obtenue à partir de la TT-MCTAG donnée en figure 7*

Il est clair que pour chaque $\alpha \in I, w \in T^*$ il existe une dérivation RCG $S(w) \Rightarrow \langle \alpha, \emptyset \rangle(w) \stackrel{*}{\Rightarrow} \epsilon$ si et seulement si il existe un arbre de dérivation TT-RCG $\langle V, E, r \rangle$ avec $l(r) = \langle \alpha, \emptyset, w \rangle$.

De plus, pour une TT-MCTAG donnée, nous appelons *arbre de dérivation TAG décoré* chaque sous-arbre d'un arbre de dérivation TAG dont les nœuds sont étiquetés par (i) les productions de l'arbre dérivé qu'ils représentent (une production pour un arbre initial, deux productions pour un arbre auxiliaire) et (ii) l'ensemble des arguments qu'ils dominent et qui dépendent de têtes plus hautes dans l'arbre de dérivation.

Définition 18 (Arbre de dérivation TAG décoré)

Soient $G = \langle I, A, N, T, \mathcal{A} \rangle$ une TT-MCTAG et $D = \langle V, E, r \rangle$ avec étiquetage l un sous-arbre d'un arbre de dérivation saturé D dans G.

*L'*arbre de dérivation TAG décoré *correspondant est $D' = \langle V, E, r \rangle$ avec étiquetage l' tel que*

– pour chaque $v \in V$ avec $l(v) = \gamma \in I$:

$l'(v) = \langle \gamma, LPA, x_\gamma \rangle$ où $x_\gamma \in T^*$ est la production de l'arbre dérivé correspondant au sous-arbre de D enraciné en v et $LPA = a(\gamma)$.

– pour chaque $v \in V$ avec $l(v) = \gamma \in A$:

$l'(v) = \langle \gamma, LPA, l_\gamma, r_\gamma \rangle$ où l_γ et r_γ sont les deux parties de la production de l'arbre dérivé correspondant au sous-arbre de D enraciné en v (parties à gauche et à droite du nœud pied respectivement) ; LPA est un ensemble d'ensembles contenant les éléments de $A(G)$ tels que le nombre d'occurrences d'un arbre $\beta \in A(G)$ dans LPA est $|\{v' \mid v'$ est un β-nœud, $\langle v, v' \rangle \in E^+\}| - |\{v' \mid v'$ est un $h(\beta)$-nœud, $\langle v, v' \rangle \in E^+\}|$.

Notons que l'arbre de dérivation TAG décoré est exactement l'arbre de dérivation TT-RCG (*cf.* preuve ci-dessous).

De plus, il est évident que les informations additionnelles dans un arbre de dérivation TAG décoré, comparé avec l'arbre de dérivation sous-jacent, peuvent être obtenues à partir de ce dernier, *i.e.*, ces informations y sont déjà présentes (voir figure 9 pour les arbres de dérivation et dérivés TAG sous-jacents à l'arbre de dérivation TT-RCG de la figure 8)[14]. En conséquence de quoi, chaque arbre de dérivation TAG dans une TT-MCTAG correspond à exactement un arbre de dérivation TAG décoré, et *vice versa*.

Une fois ces structures définies, nous pouvons prouver la correspondance entre les arbres de dérivation TAG décorés définis par la k-TT-MCTAG G et les arbres de dérivation TT-RCG de la RCG G' correspondante obtenue à partir de notre algorithme de construction. Plus précisément, nous montrons que chaque arbre de dérivation TAG décoré dans G est un arbre de dérivation TT-RCG par rapport à G' et *vice versa*.

Lemme 1

Soient $G = \langle I, A, N, T, \mathcal{A} \rangle$ une TT-MCTAG et G' la RCG obtenue à partir de l'algorithme de construction décrit précédemment. Un arbre $D = \langle V, E, r \rangle$ est un arbre de dérivation TAG décoré dans G si et seulement si D est un arbre de dérivation TT-RCG dans G'.

Nous pouvons montrer cela par induction sur la hauteur de l'arbre de dérivation. La preuve est donnée en appendice.

Grâce à cette correspondance entre les structures de dérivation dans la RCG et la TAG équivalente, on peut facilement obtenir les analyses TAG à partir de la sortie d'un analyseur syntaxique pour RCG.

14. Comme mentionné précédemment, nous ne distinguons pas les occurrences multiples d'un même tuple. Cependant, en pratique, nous convertissons la TT-MCTAG sélectionnée par la chaîne d'entrée, ainsi les doublons sont renommés.

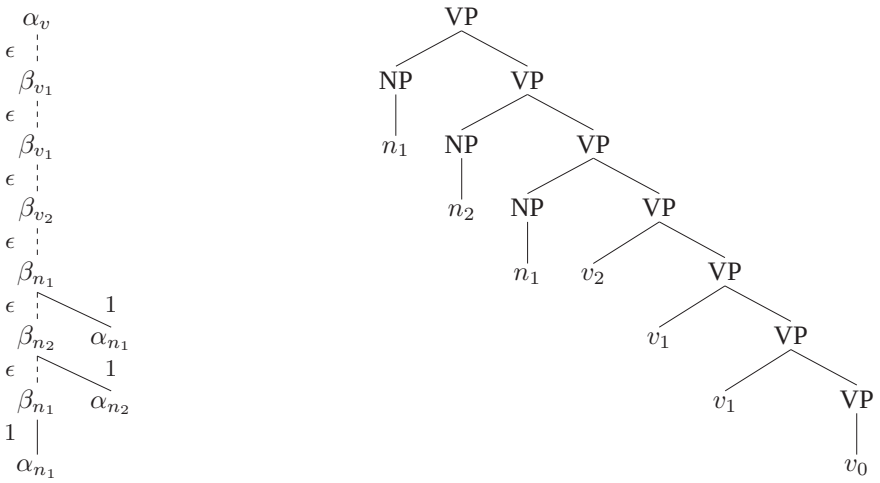


Figure 9. *Arbres de dérivation et dérivé TAG sous-jacents à l'arbre de dérivation TT-RCG de la figure 8*

6. Relation entre k-TT-MCTAG et TAG

En fait, la transformation d'une k-TT-MCTAG en RCG simple décrite ci-dessus nous donne tout de suite un algorithme pour construire une TAG équivalente à une k-TT-MCTAG donnée. Pour chaque arbre γ et chaque LPA, on introduit des arbres $\langle \gamma, LPA \rangle$ qui se distinguent de γ uniquement par leurs contraintes d'adjonction. Les contraintes f_{OA} et f_{SA} font attention à ce que les LPAs soient correctement transmises le long de l'arbre de dérivation. Cette idée est illustrée sur la figure 10.

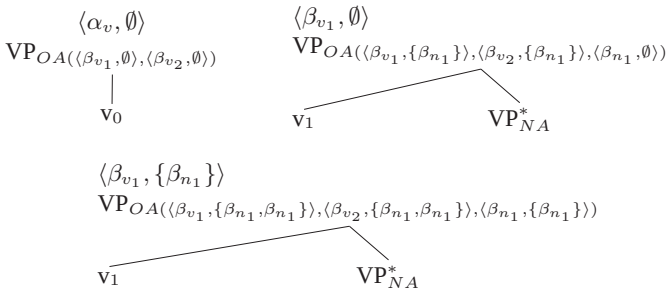


Figure 10. *Quelques arbres de la TAG équivalente à la TT-MCTAG de la figure 7*

La construction générale procède comme suit.

Pour chaque $\gamma \in I \cup A, \gamma = \langle V, E, r \rangle$ et chaque LPA possible pour γ on construit tous les arbres $\gamma' = \langle V', E', r' \rangle$ tels que γ' soit isomorphe à γ avec les mêmes étiquettes, et qu'il existe une fonction $f : V' \rightarrow \mathcal{P}(A(\gamma))$ telle que

1) $f(v') \subseteq f_{SA}(v)$ pour tous $v' \in V'$, v le nœud isomorphe dans γ, et $\{f(v') \mid v' \in V'\}$ est une partition de $A(\gamma)$,

2) pour tout nœud $v' \in V'$, $v' \neq r'$ isomorphe à $v \in V$: $f_{OA}(v') = f_{OA}(v)$ si $f(v') = \emptyset$, sinon $f_{OA}(v') = 1$; et $f_{SA}(v') = \{\beta' \mid$ il existe un $\beta \in f_{SA}(v)$ tel que β' est un nouvel arbre obtenu à partir de β avec LPA $f(v') \setminus \{\beta\}\}$,

3) et si $\Gamma = LPA \setminus \{\gamma\} \cup f(r')$, les contraintes pour la racine sont définies de telle manière que $f_{OA}(r') = f_{OA}(r)$ si $\Gamma = \emptyset$, $f_{OA}(r') = 1$ sinon ; et $f_{SA}(r') = \{\beta' \mid$ il existe un $\beta \in f_{SA}(r)$ tel β' est un nouvel arbre obtenu à partir de β avec LPA $\Gamma \setminus \{\beta\}\}$.

Avec cette construction, nous obtenons le théorème suivant.

Théorème 2

Pour chaque k-TT-MCTAG G, il existe une TAG G' telle que $L(G) = L(G')$.

La preuve est omise pour des raisons d'espace.

En fait, les deux grammaires génèrent non seulement les mêmes chaînes mais aussi les mêmes arbres dérivés. Concernant l'autre direction, nous savons que pour chaque TAG il existe une TAG lexicalisée (LTAG) qui est fortement équivalente (Joshi et Schabes, 1997) et chaque LTAG est une TT-MCTAG sans arguments. En conséquence, pour chaque TAG il existe une TT-MCTAG (et aussi une k-TT-MCTAG pour chaque k) qui est fortement équivalente. Ceci nous donne alors une équivalence forte entre k-TT-MCTAG et TAG.

Dans ce contexte, au lieu de passer par les RCG simples pour construire la TAG équivalente, on aurait pu transformer une k-TT-MCTAG directement en TAG. Pourtant, le passage par la RCG permet de se concentrer sur les arbres de dérivation TAG valides en TT-MCTAG car les clauses de la RCG représentent d'une manière directe les arcs dans un arbre de dérivation TAG. De plus, on peut envisager d'étendre la stratégie consistant à utiliser des RCG simples pour l'analyse syntaxique à d'autres formalismes, *i.e.*, d'intégrer d'autres formalismes dans une architecture d'analyse syntaxique pour formalismes grammaticaux faiblement sensibles au contexte.

Aussi, TT-MCTAG n'est pas le seul type de TAG à composantes multiples qui devient faiblement équivalent à TAG si on impose une limite sur le nombre d'arbres qui attendent d'être adjoints. Une idée similaire est utilisée dans les *k-delayed tree-local MCTAG* définies par (Chiang et Scheffler, 2008) : dans ce formalisme, les membres d'un ensemble élémentaire ne sont pas forcément adjoints tout de suite, on peut retarder les adjonctions. (Chiang et Scheffler, 2008) montrent que si on limite le nombre d'ensembles élémentaires dont des arbres attendent d'être adjoints à un certain k, on obtient un formalisme faiblement équivalent à TAG.

Enfin, les résultats obtenus dans cet article montrent que les langages des k-TT-MCTAG sont polynomiaux, autrement dit la complexité de l'analyse syntaxique en k-TT-MCTAG est polynomiale par rapport à la longueur de la chaîne d'entrée. Avec l'équivalence forte aux TAG, on obtient plus précisément une complexité d'analyse de $\mathcal{O}(n^6)$. Pourtant, comme énoncé plus haut, le problème de la reconnaissance universelle est probablement NP-difficile. Autrement dit, l'analyse syntaxique en k-TT-MCTAG est probablement NP-difficile par rapport à la taille de la grammaire et à la longueur de la chaîne d'entrée. La difficulté réside dans l'explosion du nombre de LPA possibles au moment de la transformation qui mène à une explosion de la taille de la RCG par rapport à la taille de la TT-MCTAG de départ. Une stratégie pour éviter au moins en partie cette explosion serait de construire d'abord une RCG sous-spécifiée avec des noms de prédicats contenant des variables pour les LPA. Plus concrètement, dans une clause on spécifie uniquement l'arbre argument manipulé dans cette clause en laissant sous-spécifié le reste de la LPA de cette clause. Une implantation de cette stratégie est planifiée dans des travaux futurs.

7. Conclusion

Cet article étudie la relation entre deux formalismes grammaticaux, TT-MCTAG et RCG. TT-MCTAG est un formalisme de réécriture d'arbres qui permet de modéliser les phénomènes d'ordre des mots libre dans certains langages, comme l'allemand, par exemple. RCG, de son côté, est connue pour ses propriétés formelles particulièrement intéressantes : RCG couvre la classe des langages qui sont analysables en temps polynomial (PTIME), les RCG simples sont légèrement sensibles au contexte. De plus, il existe des algorithmes d'analyse pour RCG.

Ici, nous avons montré comment construire, pour une TT-MCTAG donnée et avec une certaine limitation (en d'autres termes pour une certaine k-TT-MCTAG), une RCG équivalente. Ce résultat est intéressant d'un point de vue formel, puisqu'il montre que la classe des langages de chaînes générés par k-TT-MCTAG est contenue dans la classe des langages générés par les RCG simples. En particulier, k-TT-MCTAG est légèrement sensible au contexte. En transférant l'idée principale de la construction d'une RCG à la TAG sous-jacente de la k-TT-MCTAG, on peut même montrer que les k-TT-MCTAG sont fortement équivalentes aux TAG.

D'un point de vue pratique, nous pouvons utiliser cette transformation de k-TT-MCTAG vers RCG simple pour une analyse en deux temps de k-TT-MCTAG. Dans une première étape, nous transformons la k-TT-MCTAG en RCG, et dans une seconde étape, nous analysons la chaîne d'entrée avec la RCG ainsi produite. Aussi, nous avons vu que l'arbre de dérivation k-TT-MCTAG peut être extrait de la dérivation RCG directement. Nous avons implanté cet analyseur dans le cadre du développement d'une grammaire d'arbres de l'allemand (Kallmeyer *et al.*, 2008a).

8. Bibliographie

Boullier P., A Proposal for a Natural Language Processing Syntactic Backbone, Technical Report n° 3342, INRIA, 1998.

Boullier P., « On TAG Parsing », *TALN 99, 6e conférence annuelle sur le Traitement Automatique des Langues Naturelles*, Cargèse, Corse, p. 75-84, July, 1999.

Boullier P., « Range Concatenation Grammars », *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT2000)*, Trento, Italy, p. 53-64, February, 2000.

Burden H., Ljunglöf P., « Parsing Linear Context-Free Rewriting Systems », *IWPT'05, 9th International Workshop on Parsing Technologies*, Vancouver, Canada, October, 2005.

Chiang D., Scheffler T., « Flexible Composition and Delayed Tree-Locality », *TAG+9 Proceedings of the ninth International Workshop on Tree-Adjoining Grammar and Related Formalisms (TAG+9)*, Tübingen, p. 17-24, June, 2008.

Joshi A. K., « Tree adjoining grammars : How much context-sensitivity is required ro provide reasonable structural descriptions ? », *in* D. Dowty, L. Karttunen, A. Zwicky (eds), *Natural Language Parsing*, Cambridge University Press, p. 206-250, 1985.

Joshi A. K., « An introduction to Tree Adjoining Grammars », *in* A. Manaster-Ramer (ed.), *Mathematics of Language*, John Benjamins, Amsterdam, p. 87-114, 1987.

Joshi A. K., Levy L. S., Takahashi M., « Tree Adjunct Grammars », *Journal of Computer and System Science*, vol. 10, p. 136-163, 1975.

Joshi A. K., Schabes Y., « Tree-Adjoning Grammars », *in* G. Rozenberg, A. Salomaa (eds), *Handbook of Formal Languages*, Springer, Berlin, p. 69-123, 1997.

Kallmeyer L., « Tree-local Multicomponent Tree Adjoining Grammars with Shared Nodes », *Computational Linguistics*, vol. 31, n° 2, p. 187-225, 2005.

Kallmeyer L., Lichte T., Maier W., Parmentier Y., Dellert J., « Developing a TT-MCTAG for German with an RCG-based parser », *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC 2008)*, Marrakech, Morocco, 2008a.

Kallmeyer L., Lichte T., Maier W., Parmentier Y., Dellert J., Evang K., « TuLiPA : Towards a Multi-Formalism Parsing Environment for Grammar Engineering », *Proceedings of the GEAF08 Workshop*, 2008b.

Lichte T., « An MCTAG with Tuples for Coherent Constructions in German », *Proceedings of the 12th Conference on Formal Grammar 2007*, Dublin, Ireland, 2007.

Lichte T., Kallmeyer L., « Factorizing Complementation in a TT-MCTAG for German », *Proceedings of the Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+9)*, Tübingen, p. 57-64, June, 2008.

Parmentier Y., Kallmeyer L., Maier W., Lichte T., Dellert J., « TuLiPA : A syntax-semantics parsing environment for mildly context-sensitive formalisms », *Proceedings of the Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+9)*, Tübingen, p. 121-128, June, 2008.

Parmentier Y., Maier W., « Using constraints over finite sets of integers for range concatenation grammar parsing », *Proceedings of 6th International Conference on Natural Language Processing, GoTAL 2008*, Gothenburg, Sweden, 2008.

Seki H., Matsumura T., Fujii M., Kasami T., « On multiple context-free grammars », *Theoretical Computer Science*, vol. 88, n° 2, p. 191-229, 1991.

Søgaard A., Lichte T., Maier W., « The complexity of linguistically motivated extensions of tree-adjoining grammar », *Recent Advances in Natural Language Processing 2007*, Borovets, Bulgaria, 2007.

Villemonte de La Clergerie E., « Parsing Mildly Context-Sensitive Languages with Thread Automata », *Proceedings of the 19th International Conference on Computational Linguistics (COLING'02)*, August, 2002.

Weir D. J., Characterizing mildly context-sensitive grammar formalisms, PhD thesis, University of Pennsylvania, 1988.

XTAG Research Group, A Lexicalized Tree Adjoining Grammar for English, Technical report, Institute for Research in Cognitive Science, Philadelphia, 2001. Available from `ftp ://ftp.cis.upenn.edu/pub/xtag/release-2.24.2001/tech-report.pdf`.

Appendice : preuves

Preuve du lemme 1

Soit $G = \langle I, A, N, T, \mathcal{A}\rangle$ une TT-MCTAG avec G' étant la RCG obtenue à partir de G en utilisant l'algorithme de transformation décrit précédemment.

À montrer : un arbre $D = \langle V, E, r\rangle$ est un arbre de dérivation TAG décoré dans G si et seulement si D est un arbre de dérivation TT-RCG dans G'.

Induction sur le nombre maximal de nœuds sur le chemin de r à une feuille (ce que nous appelons *hauteur* d'un arbre) :

1) $n = 1$.
$D = \langle\{r\}, \emptyset, r\rangle$. $l(r) = \langle\alpha, LPA, w\rangle$ avec $\alpha \in I, w \in T^*$ (ou $l(r) = \langle\beta, LPA, w_1, w_2\rangle$ avec $\beta \in I, w_1, w_2 \in T^*$ respectivement). Ce qui suit est vrai pour D :
D est un arbre de dérivation TAG décoré dans G
si et seulement si

- α (β respectivement) n'a pas de nœud de substitution ni de nœud v avec $f_{OA}(v) = 1$,

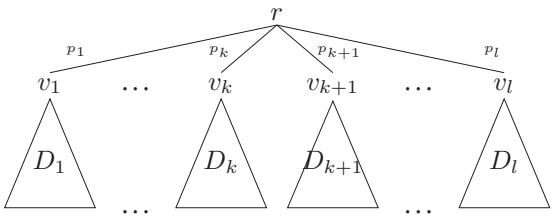
- il n'y a pas de $\beta' \in A$ avec $\alpha = h(\beta')$ ($\beta = h(\beta')$ respectivement) puisque sinon, *via* (SN-TTL), un β'-nœud doit être dominé par r, et

- $LPA = \emptyset$ puisque sinon r dominerait les nœuds étiquetés par les éléments de LPA

si et seulement si il existe une clause $\langle\alpha, \emptyset\rangle(w) \rightarrow \epsilon$ ($\langle\beta, \emptyset\rangle(w_1, w_2) \rightarrow \epsilon$ respectivement)
si et seulement si D est un arbre de dérivation TT-RCG dans G'.

2) $n \rightarrow n + 1$
Nous supposons que D est tel que : $D = \langle V, E, r\rangle$ a pour hauteur $n + 1$. Soient $\langle r, v_1\rangle, \ldots, \langle r, v_l\rangle \in E$ les arêtes sortantes de r telles qu'il y a un $k(1 \leq k \leq l)$ avec $g(\langle r, v_i\rangle) = p_i$ une adresse de nœud interne (non feuille) pour $1 \leq i \leq k$

et $g(\langle r, v_i \rangle) = p_i$ une adresse de nœud de substitution pour $k+1 \leq i \leq l$. Soit $D_i = \langle V_i, E_i, v_i \rangle$ le sous-arbre de D enraciné en v_i pour $1 \leq i \leq l$.



Pour chaque D_i, $1 \leq i \leq l$, il est vrai par induction qu'ils sont des arbres de dérivation TAG décorés dans G si et seulement s'ils sont des arbres de dérivation TT-RCG dans G' puisqu'ils sont de hauteur $\leq n$.

$l(r) = \langle \alpha, LPA, w \rangle$ avec $\alpha \in I, w \in T^*$ (ou $l(r) = \langle \beta, LPA, w_1, w_2 \rangle$ avec $\beta \in I, w_1, w_2 \in T^*$ respectivement) et $l(v_i) = \langle \beta_i, LPA_i, w_i^1, w_i^2 \rangle$ pour $1 \leq i \leq k$ et $l(v_i) = \langle \alpha_i, LPA_i, w_i \rangle$ pour $k+1 \leq i \leq l$.

Alors, ce qui suit est vrai pour D :
D est un arbre de dérivation TAG décoré dans G

si et seulement si
$D_1, \ldots, D_l$ sont des arbres de dérivation TT-RCG dans G' et

a) puisque D est un arbre de dérivation TAG saturé :

- $\gamma' = \gamma[n_1, \gamma_1] \ldots [n_l, \gamma_l]$ est défini avec $\gamma = \alpha$ ($\gamma = \beta$ respectivement) et pour $1 \leq i \leq l$, n_i est le nœud d'adresse p_i dans γ et γ_i est l'arbre dérivé obtenu à partir de l'arbre de dérivation D_i,
- $\beta_i \in f_{SA}(n_i)$ pour tout $1 \leq i \leq k$,
- $f_{OA}(n) = 0$ pour tous les nœuds n dans γ avec $n \notin \{n_1, \ldots, n_k\}$, et
- $n_{k+1}, \ldots, n_l$ sont tous les nœuds de substitution dans γ

b) et puisque D est le sous-arbre de l'arbre de dérivation TAG défini pour une TT-MCTAG : il existe un ensemble d'ensembles LPA_{old} et il existe une partition $a_1(\gamma), \ldots, a_k(\gamma)$ des arguments $a(\gamma)$ avec $a_i(\gamma) \cap \{\beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \ldots, \beta_k\} = \emptyset$ telle que

- $LPA = (LPA_{old} \cup a(\gamma)) \setminus \{\gamma\}$,
- s'il n'existe pas de $p_i = \epsilon$ pour $i, 1 \leq i \leq k$, alors $LPA_{old} = \emptyset$,
- pour toute position $p_i \neq \epsilon$ ($1 \leq i \leq k$), $LPA_i = (a_i(\gamma) \cup a(\beta_i)) \setminus \{\beta_i\}$,
- pour $p_i = \epsilon$, $LPA_i = (LPA_{old} \cup a_i(\gamma) \cup a(\beta_i)) \setminus \{\gamma, \beta_i\}$.

si et seulement si

$D_1, \ldots, D_l$ sont des arbres de dérivation TT-RCG dans G',
et dans G', il existe les clauses suivantes :

a) Supposons que $q_1, \ldots, q_m$ sont les sites d'adjonction possibles dans γ. Alors, selon la construction de G', il existe une clause

$$\langle \gamma, LPA_{old} \rangle(\sigma_\gamma) \rightarrow \langle adj, \gamma, q_1, LPA_{q_1} \rangle(L_{q_1}, R_{q_1}) \ldots \langle adj, \gamma, q_m, LPA_{q_m} \rangle(L_{q_m}, R_{q_m}) \langle sub, \gamma, p_{k+1} \rangle(X_{p_{k+1}}) \ldots \langle sub, \gamma, p_l \rangle(X_{p_l})$$

telle que

- $LPA_q = \epsilon$ pour toute position $q \in \{q_1, \ldots, q_m\} \setminus \{p_1, \ldots, p_k\}$.
- $LPA_{p_i} = a_i(\gamma)$ pour toute position $p_i \in \{p_1, \ldots, p_k\}, p_i \neq \epsilon$.
- $LPA_{p_i} = (LPA_{old} \setminus \{\gamma\}) \cup a_i(\gamma)$ pour $p_i \in \{p_1, \ldots, p_k\}, p_i = \epsilon$.

b) pour toute position $p_i \in \{p_1, \ldots, p_k\}$, il existe une clause
$\langle adj, \gamma, p_i, LPA_{p_i} \rangle(L, R) \rightarrow \langle \beta_i, LPA'_{p_i} \rangle(L, R)$
telle que $LPA'_{p_i} = LPA_{p_i} \setminus \{\beta_i\}$.

c) pour toute position $q \in \{q_1, \ldots, q_m\} \setminus \{p_1, \ldots, p_k\}$, il existe une clause
$\langle adj, \gamma, q, \emptyset \rangle(\epsilon, \epsilon) \rightarrow \epsilon$.

d) pour toute position $p_i \in \{p_{k+1}, \ldots, p_l\}$, il existe une clause
$\langle sub, \gamma, p_i \rangle(X) \rightarrow \langle \alpha_i, \emptyset \rangle(X)$.

e) (puisque σ_γ est la chaîne de décoration) il existe des instanciations de ces clauses produisant, pour un membre de gauche $X = \langle \gamma, LPA \rangle(w)$ (ou $X = \langle \gamma, LPA \rangle(w_1, w_2)$ respectivement) :

$$X \Rightarrow \langle adj, \gamma, q_1, LPA_{q_1} \rangle(w^1_{q_1}, w^2_{q_1}) \ldots \langle adj, \gamma, q_m, LPA_{q_m} \rangle(w^1_{q_m}, w^2_{q_m}) \langle sub, \gamma, p_{k+1} \rangle(w_{k+1}) \ldots \langle sub, \gamma, p_l \rangle(w_l)$$
$$\stackrel{*}{\Rightarrow} \langle \beta_1, LPA'_{p_1} \rangle(w^1_{p_1}, w^2_{p_1}) \ldots \langle \beta_k, LPA'_{p_k} \rangle(w^1_{p_k}, w^2_{p_k}) \langle \alpha_{k+1}, \emptyset \rangle(w_{k+1}) \ldots \langle \alpha_l, \emptyset \rangle(w_l)$$

avec $w^1_q = w^2_q = \epsilon$ pour toute position $q \in \{q_1, \ldots, q_m\} \setminus \{p_1, \ldots, p_k\}$ et $w^1_{p_i} = w^1_i, w^2_{p_i} = w^2_i$ pour toute position $p_i \in \{p_1, \ldots, p_k\}$.

si et seulement si D est un arbre de dérivation TT-RCG dans G'.

□