Université de Montréal

Le traitement des locutions en génération automatique de texte multilingue

Par

Michaelle Dubé

Département de linguistique et de traduction, Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de maîtrise ès arts (M.A.)

en linguistique

août 2021

Université de Montréal

Département de linguistique et de traduction, Faculté des arts et des sciences

---

Ce mémoire intitulé

Le traitement des locutions en génération automatique de texte multilingue

Présenté par

Michaelle Dubé

A été évaluée par un jury composé des personnes suivantes

Patrick Drouin
Président-rapporteur

François Lareau
Directeur de recherche

Alain Polguère
Membre du jury

Résumé

La locution est peu étudiée en génération automatique de texte (GAT). Syntaxiquement, elle forme un syntagme, alors que sémantiquement, elle ne constitue qu'une seule unité. Le présent mémoire propose un traitement des locutions en GAT multilingue qui permet d'isoler les constituants de la locution tout en conservant le sens global de celle-ci. Pour ce faire, nous avons élaboré une solution flexible à base de patrons universels d'arbres de dépendances syntaxiques vers lesquels pointent des patrons de locutions propres au français (Pausé, 2017). Notre traitement a été effectué dans le réalisateur de texte profond multilingue GenDR à l'aide des données du *Réseau lexical du français* (RL-fr). Ce travail a abouti à la création de 36 règles de lexicalisation par patron (indépendantes de la langue) et à un dictionnaire lexical pour les locutions du français. Notre implémentation couvre 2 846 locutions du RL-fr (soit 97,5 %), avec une précision de 97,7 %.

Le mémoire se divise en cinq chapitres, qui décrivent : 1) l'architecture classique en GAT et le traitement des locutions par différents systèmes symboliques ; 2) l'architecture de GenDR, (principalement sa grammaire, ses dictionnaires, son interface sémantique-syntaxe et ses stratégies de lexicalisations) ; 3) la place des locutions dans la phraséologie selon la théorie Sens-Texte, ainsi que le RL-fr et ses patrons syntaxiques linéarisés ; 4) notre implémentation de la lexicalisation par patron des locutions dans GenDR, et 5) notre évaluation de la couverture de la précision de notre implémentation.

Mots-clés : locution, expression polylexicale, génération automatique de texte, lexicalisation, théorie Sens-Texte, réalisation linguistique.

Abstract

Idioms are rarely studied in natural language generation (NLG). Syntactically, they form a phrase, while semantically, they correspond to a single unit. In this master's thesis, we propose a treatment of idioms in multilingual NLG that enables us to isolate their constituents while preserving their global meaning. To do so, we developed a flexible solution based on universal templates of syntactic dependency trees, onto which we map French-specific idiom patterns (Pausé, 2017). Our work was implemented in Generic Deep Realizer (GenDR) using data from the *Réseau lexical du français* (RL-fr). This resulted in the creation of 36 template-based lexicalization rules (independent of language) and of a lexical dictionary for French idioms. Our implementation covers 2846 idioms of the RL-fr (i.e., 97.5%), with an accuracy of 97.7%.

We divided our analysis into five chapters, which describe: 1) the classical NLG architecture and the handling of idioms by different symbolic systems; 2) the architecture of GenDR (mainly its grammar, its dictionaries, its semantic-syntactic interface, and its lexicalization strategies); 3) the place of idioms in phraseology according to Meaning-Text Theory (*théorie Sens-Texte*), the RL-fr and its linearized syntactic patterns; 4) our implementation of the template lexicalization of idioms in GenDR; and 5) our evaluation of the coverage and the precision of our implementation.

Key words: idiom, multiword expressions, multilingual natural language generation, lexicalization, Meaning-Text theory, linguistic realization.

Table des matières

Résumé 3

Abstract 4

Table des matières 5

Liste des tableaux 8

Liste des figures 9

Liste des sigles et abréviations 12

Remerciements 14

Introduction 15

Chapitre 1 Génération automatique de texte 17

1.1 Architecture classique 17

1.2 Réalisation linguistique 19

1.2.1 Réalisateurs de surface 20

1.2.1.1 SimpleNLG 20

1.2.1.2 JSrealB 21

1.2.1.3 RealPro 22

1.2.2 Réalisateurs profonds 24

1.2.2.1 KPML 24

1.2.2.2 FUF/SURGE 26

1.2.2.3 FLAUBERT/G-TAG 28

1.2.2.4 MARQUIS 32

1.2.2.5 FORGe 34

1.3 Synthèse 35

Chapitre 2 GenDR 36
2.1 Architecture 36
2.1.1 Dictionnaires 37
2.1.2 Grammaires 39
2.1.3 Graphes 40
2.2 Interface sémantique-syntaxe 43
2.2.1 Théorie Sens-Texte 43
2.2.2 Arborisation 45
2.2.3 Lexicalisation 46
Chapitre 3 Locutions : définition, corpus et classification 49
3.1 Définition 49
3.1.1 Phraséologie dans le cadre de la théorie Sens-Texte 49
3.1.2 Locutions 52
3.2 Données lexicales 53
3.2.1 Réseau lexical du français 53
3.2.2 Patrons linguistiques : classification des locutions du français 54
Chapitre 4 Implémentation 61
4.1 Grammaire 61
4.1.1 Patrons génériques 61
4.1.1.1 Calcul des patrons génériques 62
4.1.1.2 Mise en correspondance des patrons linguistiques et génériques 67
4.1.2 Règles de transduction 69
4.1.3 Génération automatique des règles de transduction 71
4.2 Dictionnaire 77

4.2.1 Structure du dictionnaire 77
4.2.2 Génération automatique du dictionnaire 80
Chapitre 5 Évaluation 89
5.1 Couverture 89
5.1.1 Méthodologie 89
5.1.2 Résultats 89
5.1.3 Discussion 92
5.2 Précision 95
5.2.1 Méthodologie 95
5.2.2 Résultats 98
5.2.3 Discussion 98
Conclusion 102
Références bibliographiques 104
Annexe 1 : patrons génériques 113
Annexe 2 : protocole de l'évaluation finale 118

Liste des tableaux

Tableau 1 – Niveaux d'un modèle Sens-Texte 44

Tableau 2 – Classification sémantique des locutions 52

Tableau 3 – Exemples de patrons différenciés par l'ajout de marqueurs fonctionnels 56

Tableau 4 – Exemples de patrons différenciés par l'ajout de positions actancielles 59

Tableau 5 – Exemples de patrons syntaxiques linéarisés 60

Tableau 6 – Patrons génériques `3_01`, `3_02` et leurs partitions 63

Tableau 7 – Nombre d'arbres à *n* nœuds différents 64

Tableau 8 – Nombre d'arbres à 7 nœuds 65

Tableau 9 – Exemple de correspondance entre les patrons linguistiques et génériques 67

Tableau 10 – Exemple de fréquence des patrons génériques 76

Tableau 11 – Extrait d'un fichier XML de Spiderlex 81

Tableau 12 – Extrait d'un fichier CSV de Spiderlex 81

Tableau 13 – Extrait de la table de données `idiom_patterns.tsv` 82

Tableau 14 – Extrait de la table de données combinées `idioms` 83

Tableau 15 – Patrons linguistiques les plus fréquents du RL-fr 90

Tableau 16 – Couverture des locutions dans GenDR selon le nombre de nœuds 90

Tableau 17 – Nombre de locutions du RL-fr et pourcentage de notre couverture en fonction de la partie du discours 91

Tableau 18 – Fréquence des patrons génériques dans le RL-fr 92

Tableau 19 – Nombre d'arbres possibles à *n* nœuds et les locutions du RL-fr qu'ils couvrent 92

Tableau 20 – Exemple de locutions non traitées 95

Tableau 21 – Répartition des locutions évaluées par patron générique 96

Tableau 22 – Résultats de l'évaluation manuelle 98

Liste des figures

Figure 1 – Modules et tâches de GAT 18
Figure 2 – Architecture classique de la GAT 19
Figure 3 – Trois façons d'ajouter un syntagme prépositionnel dans SimpleNLG 21
Figure 4 – Exemple du traitement de la négation dans JSrealB 22
Figure 5 – Architecture de RealPro 23
Figure 6 – Exemple de structure syntaxique dans RealPro 24
Figure 7 – Système en réseau multilingue 25
Figure 8 – Exemple d'entrée SPL dans KPML 25
Figure 9 – Exemple de génération multilingue par patron 26
Figure 10 – Exemple de structure d'entrée et de sortie dans SURGE 27
Figure 11 – Architecture de G-TAG 28
Figure 12 – Arbres élémentaires TAG 29
Figure 13 – Opérations de substitution et d'adjonction 29
Figure 14 – Arbre dérivé et arbre de dérivation 29
Figure 15 – Exemple d'arbre de g-dérivation 30
Figure 16 – Exemple d'arbre g-dérivé 30
Figure 17 – Exemple de représentation conceptuelle dans FLAUBERT 31
Figure 18 – Étapes de la réalisation linguistique dans MARQUIS 33
Figure 19 – Exemple d'entrée prédicat-arguments dans FORGe 34
Figure 20 – Exemple d'entrée du dictionnaire sémantique pour 'courtiser' 38
Figure 21 – Exemple d'entrée du dictionnaire lexical pour COURTISER 38
Figure 22 – Organisation des règles dans GenDR 40
Figure 23 – Exemple de graphes sémantiques en mode graphique et textuel dans GenDR 41
Figure 24 – Exemple de graphes syntaxiques profonds en mode graphique et textuel 41
Figure 25 – Exemple de graphes de surface en mode graphique et textuel 42
Figure 26 – Structure fonctionnelle d'un modèle Sens-Texte 43
Figure 27 – Typologie générale des phrasèmes lexicaux 50

Figure 28 – Structure syntaxique de surface de *X pédale dans le yaourt*. 57
Figure 29 – Structure syntaxique de surface de *X va aux fraises.* 57
Figure 30 – Structure syntaxique de surface de *X bat de l'aile.* 58
Figure 31 – Patrons génériques possibles pour un arbre à quatre nœuds.....62
Figure 32 – Exemple de patrons génériques non linéaires identiques.....62
Figure 33 – Trois partitions de l'entier 3.....63
Figure 34 – Exemple démontrant la non-linéarité des patrons génériques.....66
Figure 35 – Correspondance des patrons pour la locution ⌜COUPER LES CHEVEUX EN QUATRE⌝.....68
Figure 36 – Règle de lexicalisation `lex_idiom_2`.....69
Figure 37 – Entrée de dictionnaire pour la locution ⌜COUPER LES CHEVEUX EN QUATRE⌝.....70
Figure 38 – Extrait du fichier `idioms.rls` pour la règle `lex_idiom_2` 71
Figure 39 – Fonction `rule()` extraite de `partition.py`.....72
Figure 40 – Fonction `rightside()` extraite de `partition.py`.....73
Figure 41 – Fonction `partition()` extraite de `partition.py`.....74
Figure 42 – Fonction `flatten_tuple()` extraite de `partition.py`.....74
Figure 43 – Fonction `flatten_list()` extraite de `partition.py`.....74
Figure 44 – Fonction `trees()` extraite de `partition.py`.....75
Figure 45 – Exemple de correspondance entre les tuples et les arbres pour le patron `4_03`..75
Figure 46 – Exemples d'arbre profond et d'arbre plat 76
Figure 47 – Exemple d'héritage dans le dictionnaire lexical.....78
Figure 48 – Entrée de dictionnaire pour le patron linguistique `V Art NC Prép Num`.....79
Figure 49 – Entrée de dictionnaire pour la locution ⌜COUPER LES CHEVEUX EN QUATRE⌝.....80
Figure 50 – Script pour écrire les entrées des patrons linguistiques.....84
Figure 51 – Exemple de correspondance entre le script et le dictionnaire lexical.....84
Figure 52 – Exemple d'entrée pour un patron linguistique.....85
Figure 53 – Extrait du dictionnaire lexical.....85
Figure 54 – Fonction `standard_name()` extraite de `idioms.py`.....86
Figure 55 – Rappel de l'entrée de dictionnaire pour ⌜COUPER LES CHEVEUX EN QUATRE⌝.....86
Figure 56 – Fonction `reorder()` extraite de `idioms.py`.....87

Figure 57 – Fonction `to_list()` extraite de `idioms.py` 87
Figure 58 – Fonction `normalize_amalgams()` extraite de `idioms.py` 87
Figure 59 – Arborisation de la locution ⌜METTRE LES PETITS PLATS DANS LES GRANDS⌝ 93
Figure 60 – Exemple de patrons récursifs pour ⌜METTRE LES PETITS PLATS DANS LES GRANDS⌝ 94
Figure 61 – Exemple de RSyntP d'une locution dans GenDR 96
Figure 62 – Exemple de RSyntP d'une locution dans GenDR 97
Figure 63 – Exemple de RSyntS d'une locution dans GenDR 97
Figure 64 – RSyntS proposée pour le patron linéarisé V Art LocN 100

Liste des sigles et abréviations

ATILF : Analyse et traitement informatique de la langue française

DAG : graphe orienté acyclique (*directed acyclic graph*)

DPOS : partie du discours profonde (*deep part of speech*)

GAT : génération automatique de texte (*natural language generation*)

GenDR : *Generic Deep Realizer*

GP : patron de régime (*government pattern*)

OLST : Observatoire de linguistique Sens-Texte

RL-fr : Réseau lexical du français

RSém : représentation sémantique

RSyntP : représentation syntaxique profonde

RSyntS : représentation syntaxique de surface

SFG : grammaire fonctionnelle systémique (*systemic functional grammar*)

SPOS : partie du discours de surface (*surface part of speech*)

TAL : traitement automatique de la langue (*natural language processing*)

TST : théorie Sens-Texte (traduit en anglais par *Meaning-Text Theory*)

Merci de m'avoir stimulée intellectuellement
comme personne ne l'avait fait auparavant

Remerciements

Tout d'abord, je remercie vivement mon directeur de recherche, François Lareau, pour ses lumières, sa confiance et ses encouragements. Je lui suis reconnaissante, tout comme aux autres membres de l'OLST, d'avoir fourni un climat de collaboration chaleureux et agréable. Merci pour les séminaires, les dîners-discussions et l'humour sophistiqué.

J'adresse mes remerciements les plus sincères aux membres de mon jury, Patrick Drouin et Alain Polguère. En particulier, je remercie ce dernier de m'avoir accueillie à l'ATILF dans le cadre d'un stage outremer. Je souligne à ce sujet la contribution financière du CRSH — Développement Savoir. Je tiens aussi à remercier l'apport de Marie-Sophie Pausé, Florie Lambrey, Daniel Galarreta-Piquette et Sandrine Ollinger, dont les travaux ont servi d'assises à mon projet.

Je suis reconnaissante envers les personnes rencontrées grâce au CÉSAR, à VocUM et à Thèsez-vous, qui ont su m'outiller pour la recherche. Je salue au passage l'aide de Mathieu L'Allier et Julien Plante-Hébert. Je remercie également mes camarades de maîtrise, en particulier, Hubert Corriveau, qui a considérablement contribué à la qualité du mémoire.

J'aimerais exprimer ma gratitude à mes proches pour leur soutien tout au long du processus. Tout particulièrement, je remercie Eve-Marie Gendron-Pontbriand pour son temps, ses conseils judicieux et son amitié inconditionnelle. Merci à mes parents de m'avoir toujours incitée à viser plus haut. Merci aussi à Jordane La Salle et Humberto Pérez-Gagnon de m'avoir permis d'avancer.

Enfin, je serai éternellement reconnaissante envers Maxime Colleret pour ses encouragements incessants, sa pensée critique et son optimisme. Comme je connais son affection pour le domaine, je lui dédie ce mémoire.

Introduction

La **locution** a été peu étudiée en **génération automatique de texte** (GAT). Elle a ceci de particulier que, du point de vue de la **sémantique**, elle n'est qu'une seule unité de sens, alors que du point de vue de la **syntaxe**, elle forme un syntagme (un groupe d'unités syntaxiques). Cependant, elle se différencie du syntagme libre par son caractère **non compositionnel**, qui ne permet pas de prédire sa forme par la simple combinaison régulière de signes plus simples.

De plus, la locution n'est pas aussi **figée** qu'on le présume, comme l'a démontré Gross (1996). Certaines locutions permettent la flexion d'un ou de plusieurs de leurs constituants. Par exemple, le verbe FAIRE dans la locution ⌜FAIRE ATTENTION À SES FESSES⌝ peut être fléchi librement, alors que FESSES est obligatoirement au pluriel. Cette locution inclut aussi *ses*, qui peut nécessiter d'être relié à un autre élément de la phrase par coréférence. Nous devons également envisager les locutions incluant un adjectif qui dépend d'un nom. Cela complique le traitement, puisque la flexion n'est pas nécessairement faite sur la tête syntaxique de la locution. Cette situation problématique est exacerbée dans les langues à cas, comme le lituanien, où les adjectifs, en plus de s'accorder en genre et en nombre, s'accordent aussi en cas avec le nom (voir Dubinskaite, 2017). Cela est donc d'autant plus important dans une optique multilingue.

Nous proposons donc un **traitement des locutions** en GAT **multilingue** qui permet d'isoler les constituants de la locution tout en conservant le sens global de celle-ci. Pour ce faire, nous avons élaboré une solution flexible à base de patrons universels d'arbres de dépendances syntaxiques (appelés *patrons génériques*) vers lesquels pointent des patrons de locutions propres au français (appelés *patrons linguistiques*).

Le présent projet s'inscrit dans la lignée de ceux de Lambrey (2016), Dubinskaitė (2017), Galarreta-Piquette (2018), Zhao (2018), He (2020), Corriveau (2021) et Portenseigne (à paraître), qui ont participé à la mise en place et à la bonification de **GenDR**, un réalisateur de texte profond multilingue (Lareau et coll., 2018) en cours de développement à l'Observatoire de linguistique

Sens-Texte (OLST)[1]. Il a comme principal objectif d'implémenter la **lexicalisation par patron** dans GenDR, d'abord pour les locutions du français, tout en offrant un cadre pour d'autres langues. Il a aussi comme objectif secondaire de vérifier la précision des descriptions lexicographiques proposées par le ***Réseau lexical du français*** (RL-fr)[2] (Polguère, 2009 ; 2014), qui est également tributaire de la **théorie Sens-Texte** (TST ; Mel'čuk, 1997 ; Žolkovsky et Mel'čuk, 1967).

Le présent mémoire est divisé en cinq chapitres : les trois premiers établissent les bases théoriques sur lesquelles s'appuient les deux derniers. Nous présentons d'abord la branche du TAL dans lequel s'inscrit ce mémoire, soit la GAT (chapitre 1). Nous faisons un bref survol de différents systèmes de réalisation de texte, puis nous présentons GenDR, le réalisateur de texte dans lequel nous avons effectué notre implémentation (chapitre 2). Cela nous amène à présenter notre cadre théorique, la TST, sur lequel se base notre lexicalisation par patron. Puis, nous définissons la locution et présentons notre ressource lexicale, le RL-fr (chapitre 3). Ces trois chapitres jettent les bases nécessaires à notre implémentation de patrons génériques par la création d'une grammaire et d'un dictionnaire (chapitre 4). Nous concluons par l'évaluation de ces derniers en mesurant la couverture et la précision de notre implémentation (chapitre 5).

[1] http://olst.ling.umontreal.ca/
[2] https://lexical-systems.atilf.fr/

Chapitre 1 Génération automatique de texte

La GAT est une branche du **traitement automatique de la langue** (TAL) qui se concentre sur l'automatisation de tâches afin d'alléger le travail humain nécessaire à la rédaction de textes (Reiter et Dale, 1997). Elle est ainsi centrée sur la production de textes les plus naturels possibles à partir de contenu généralement non linguistique. L'entrée peut être des données brutes, des représentations abstraites ou même du texte.

La GAT peut être utile pour tout besoin communicationnel qu'il soit. Il n'est donc pas étonnant que ses **applications** soient variées et personnalisées : documents, rapports, résumés, explications, instructions, descriptions et autres. Par exemple, des systèmes ont été créés pour générer des bulletins personnalisés sur la qualité de l'air (Wanner et coll., 2010) et pour produire rapidement des rapports météorologiques précis (Sripada, 2014). Ces systèmes ont même été créés à des fins pédagogiques, par exemple pour aider de jeunes enfants à rédiger des poèmes (Hamalainen 2018). La GAT est aussi utilisée pour la couverture médiatique de compétitions sportives en langue aborigène australienne (Lareau et coll., 2011). Elle a ainsi ouvert la porte à ce qu'on peut appeler le robot-journalisme. De plus, des systèmes ont été conçus pour tester des théories et leurs modélisations. Cela permet notamment de bonifier les travaux en linguistique théorique. C'est le cas notamment de GenDR, dont il sera question au prochain chapitre. Par ailleurs, la GAT se veut économique comme elle permet de réduire le temps de production de textes et donc les coûts associés. Le développement de la GAT multilingue permet également de s'affranchir des barrières linguistiques.

Le présent chapitre commence par une présentation de l'architecture classique de la GAT. Il se concentre ensuite sur la dernière étape de celle-ci, soit la réalisation linguistique. Il se termine par un survol de différents réalisateurs de texte de surface et profonds.

1.1 Architecture classique

Dans la présente section, nous dressons les grandes lignes de l'architecture classique de la GAT décrite par Reiter et Dale (1997, 2000). Ces auteurs proposent une organisation en trois modules :

la planification du document, la microplanification et la réalisation de surface. La figure 1 présente les tâches que comporte chaque module selon une classification forme-contenu.

Module	Tâches liées au contenu	Tâches liées à la forme
Planification du document	- Détermination du contenu	- Structuration du document
Microplanification	- Lexicalisation - Génération d'expressions référentielles	- Agrégation
Réalisation	- Réalisation linguistique	- Réalisation de structure

Figure 1 – Modules et tâches de GAT (Reiter et Dale, 2000, p. 49)

Selon Danlos (1983), ainsi que Gatt et Krahmer (2018), le processus de communication peut être scindé en deux questions : la question préliminaire « quoi dire ? » et celle ultérieure « comment le dire ? ». La réponse à la première question se trouve dans la **sélection du contenu**, la **structuration du document** et l'**agrégation**. Respectivement, ces tâches demandent de : 1) déterminer quelle est l'information à communiquer en fonction de l'objectif ; 2) choisir comment organiser ces morceaux d'informations dans un ordre logique (généralement représenté par une structure arborescente) ; et si nécessaire 3) déterminer comment regrouper les informations à l'intérieur d'une phrase et d'un paragraphe.

Puis, la réponse à la deuxième question est de l'ordre de la **lexicalisation**, de la **génération d'expressions référentielles**, de la **réalisation linguistique** et de la **réalisation structurelle**. Ces dernières tâches viennent raffiner le message à transmettre et demandent respectivement de : 4) choisir les unités lexicales pour représenter le sens désiré du message ; 5) déterminer les expressions utilisées pour référer aux entités selon l'historique du discours ; 6) appliquer les règles syntaxiques, morphologiques et orthographiques pour produire le texte ; et 7) s'assurer que le format du texte respecte les balises de présentation. La figure 2, extraite du mémoire de Galarreta-Piquette (2018), résume ce processus séquentiel.

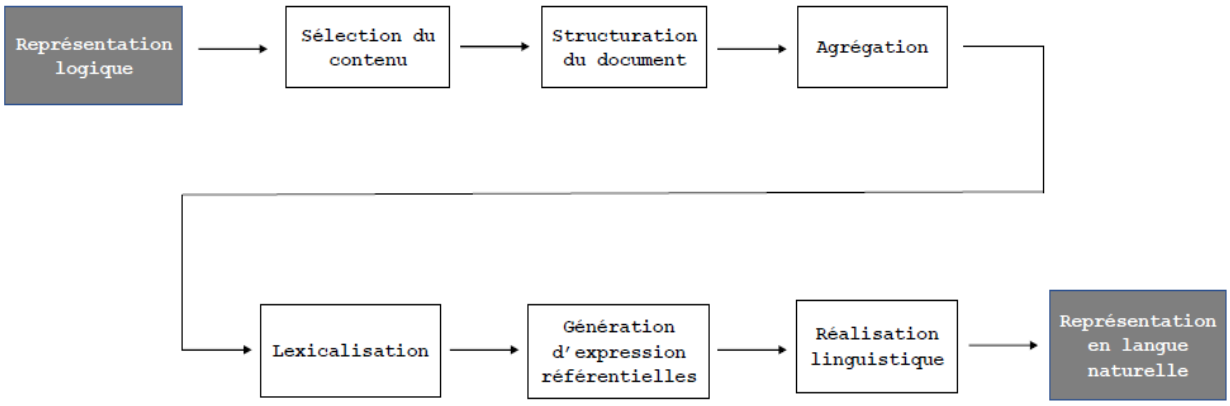


Figure 2 – Architecture classique de la GAT (Galarreta-Piquette, 2018, p. 7 ; Reiter et Dale, 1997)

Dans le cadre de notre projet de recherche, nous nous concentrons sur la réalisation linguistique. La prochaine section présente donc un tour d'horizon de ce qu'elle représente.

1.2 Réalisation linguistique

Bien que les **modèles statistiques** et **neuronaux** soient dominants en TAL, ceux-ci se concentrent le plus souvent sur la prédiction du mot suivant (Gatt et Krahmer, 2018). Ainsi, la place qu'ils réservent aux locutions, tout comme aux expressions figées en général, est restreinte : ils les considèrent comme de simples séquences probables. Les expressions figées sont extraites de leurs bases de données ou corpus et sont ensuite reprises intégralement dans les textes générés. Pourtant, l'organisation interne des locutions est fortement intéressante d'un point de vue linguistique. Sur ce point, les **modèles symboliques** présentent certains avantages pour la GAT. En effet, pour de nombreuses applications concrètes, la marge d'erreur est très restreinte et le produit de la génération se doit d'être le plus précis possible. Dans cette optique, les réalisateurs à base de règles sont les mieux adaptés, en raison de leur précision généralement plus grande. Du point de vue de la linguistique, ils offrent également une plateforme utile pour la validation des modèles (Danlos, 1987). Nous nous concentrerons donc sur les réalisateurs symboliques.

Dans les deux prochaines sections, nous faisons un survol comparatif de quelques réalisateurs de texte et de leur potentiel quant au traitement des locutions. Nous nous concentrons sur leur aspect multilingue, leur cadre théorique, leur flexibilité et leur traitement de la phraséologie.

Nous présentons d'abord trois réalisateurs qui se concentrent sur la tâche de réalisation telle que décrite par Reiter et Dale (1997, 2000), c'est-à-dire des réalisateurs de surface. Puis, nous présentons quelques réalisateurs qui prennent en charge au minimum l'interface sémantique-syntaxe, c'est-à-dire des réalisateurs profonds.

1.2.1 Réalisateurs de surface

Dans la présente section, nous traçons le portrait de trois réalisateurs de surface nommés SimpleNLG, JSrealB et RealPro. Ceux-ci se concentrent sur la tâche de réalisation où les règles de grammaire sont appliquées pour produire du texte conforme aux règles syntaxiques, morphologiques et orthographiques (Reiter et Dale, 1997).

1.2.1.1 SimpleNLG

SimpleNLG (Gatt et Reiter, 2009) est un réalisateur de surface pour l'anglais écrit en Java. Comme son nom l'indique, il vise la simplicité, car il se destine aux usagers non experts. C'est un réalisateur à base de règles qui prend en entrée des arbres de constituants, les linéarise et produit du texte. Il fonctionne grâce à une librairie Java et des structures syntaxiques lexicalisées encodées en format XML.

Plus précisément, la réalisation s'opère comme suit. D'abord, il faut apparier les constituants de base aux lexèmes correspondants. Puis, à l'aide d'opérations prédéfinies par l'interface de programmation d'application (`API` en anglais), on attribue des traits propres aux lexèmes. Par la suite et toujours grâce à l'API, on combine les lexèmes en constituants de plus en plus larges. Finalement, à l'obtention d'une phrase complète, celle-ci est linéarisée. C'est dans cette dernière étape que les unités lexicales sont ordonnées et que les règles morphosyntaxiques des traits spécifiés (dont la flexion) sont appliquées.

Ce réalisateur ne traite pas les locutions en tant que telles. La réalisation des locutions se fait donc comme celle de syntagmes. La figure 3 présente un exemple des différentes façons de traiter un syntagme. Alors que la première option simplifiée permet de réaliser une locution figée, les deux autres options permettent une réalisation plus flexible de la locution. Notamment, isoler certaines unités permet au réalisateur d'effectuer la flexion.

SPhraseSpec p = nlgFactory. createClause ("Mary", "chase", "the monkey") ;	
La façon simple :	**p.addComplement ("in the park") ;**
La façon semi-sophistiquée :	**PPPhraseSpec pp =** **nlgFactory. createPrepositionPhrase ("in", "the park") ;** **p.addComplement (pp) ;**
La façon sophistiquée :	**NPPhrase. Spec place =** **nlgFactory.createNounPhrase("park");** **place. setDeterminer ("the") ;** **PPPhraseSpec pp = nlgFactory.createPrepositionPhrase();** **pp.addComplement(place);** **pp.setPreposition("in");** **p.addComplement(pp);**

String output = realiser.realiseSentence(p); // Realiser created earlier
System.out.println(output);	

`Mary chases the monkey in the park.`	

Figure 3 – Trois façons d'ajouter un syntagme prépositionnel dans SimpleNLG (documentation[3] ; notre traduction)

Néanmoins, ce réalisateur offre une large couverture langagière. La version bilingue français-anglais **SimpleNLG-EnFr** (Vaudry et Lapalme, 2013) couvre pratiquement toute la grammaire du *Français fondamental (1er degré* ; Gougenheim et coll., 1967) et dispose d'un lexique de 3 871 entrées couvrant l'*échelle orthographique Dubois Buyse* (Ters et coll., 1988). Il a également été adapté dans de nombreuses langues : allemand (Bollmann, 2011 ; Braun et coll., 2019), français (Vaudry et Lapalme, 2013), portugais brésilien (de Oliveira et Sripada, 2014), italien (Mazzei et coll., 2016), espagnol (Ramos-Soto et coll., 2017), mandarin (Chen et coll., 2018), galicien (Cascallar-Fuentes et coll., 2018), néerlandais (de Jong et Theune, 2018) et tibétain (Kuanzhuo et coll., 2020).

1.2.1.2 JSrealB

JSrealB, pour *JavaScript Realizer Bilingual* (Lapalme, 2020 ; Molins, 2014 ; Molins et Lapalme, 2015), est un réalisateur de surface bilingue pour l'anglais et le français écrit en JavaScript. Ce réalisateur à base de règles est largement inspiré de SimpleNLG (Gatt et Reiter, 2009) et est

[3] https://github.com/simplenlg/simplenlg/wiki/Section-XI-%E2%80%93-Prepositional-phrases (23/06/2021)

destiné aux programmeurs web. Il prend en entrée des arbres de constituants (sous forme de structures syntaxiques linéarisées) et génère des expressions ou des phrases complètes. Ce texte peut par la suite être formaté en HTML si une utilisation à l'intérieur d'une page web est désirée. JSrealB peut également être utilisé comme module `node.js` prenant l'entrée d'un système externe.

Ce réalisateur ne traite pas les locutions en particulier, mais il est possible de les générer en spécifiant la structure syntaxique correspondante. Toutefois, JSrealB offre le traitement de la négation, qui est exprimée notamment par la locution ⌜NE… PAS⌝ en français. La négation (tout comme la passivation et la pronominalisation) est réalisée à l'aide d'une étiquette `type` ajoutée à la phrase active. Cette étiquette vient ajouter et réorganiser les unités syntaxiques nécessaires selon la structure attendue dans la langue sélectionnée. La figure 4 en présente un exemple.

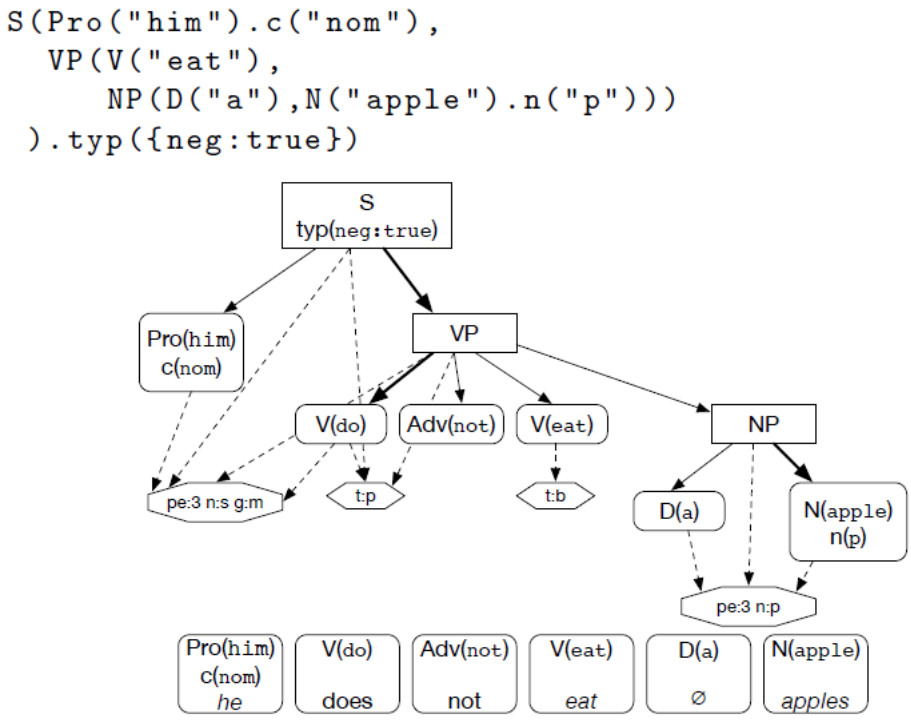


Figure 4 – Exemple du traitement de la négation dans JSrealB (Lapalme, 2020, p. 10)

1.2.1.3 RealPro

RealPro (Lavoie et Rambow, 1997) est un réalisateur de surface à base de règles suivant le formalisme de la TST. Il prend en entrée des arbres syntaxiques profonds. Ces arbres sont non

linéaires et axés sur la relation de dépendance entre les nœuds. Ils donnent donc plus de flexibilité à ce réalisateur que s'il utilisait des arbres de constituants. La figure 5 présente son architecture.

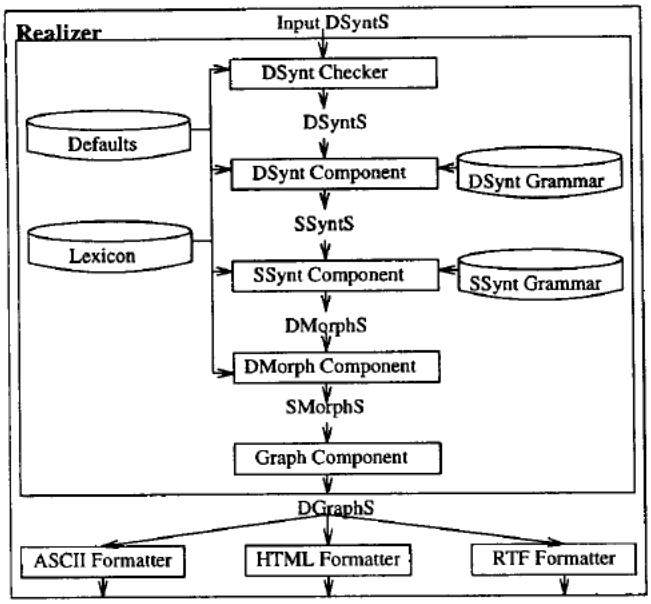


Figure 5 – Architecture de RealPro (Lavoie et Rambow, 1997, p. 3)

RealPro est implémenté en C++ et il peut être utilisé comme un serveur indépendant grâce à l'interface de programmation d'applications de C++ ou de Java. Il supporte différents formats : HTML, texte brut (ASCII) et RTF (pour *Rich Text Format*).

Le réalisateur peut traiter plusieurs phénomènes syntaxiques, dont la coordination, la topicalisation, les formes verbales (temps composés, l'aspect, le passif, la négation, les questions), ainsi que les différents pronoms (personnels, possessifs, relatifs). Cependant, il ne traite pas très bien la syntaxe dite « sophistiquée », comme les relations de dépendances de longue distance dans les phrases clivées, ainsi que les propositions subordonnées ou les relatives. La figure 6 présente un exemple de structure syntaxique dans RealPro.

```
// -------------------------------------------------------------
// John does not love Mary.
// -------------------------------------------------------------

DSYNTS:

love [ class:verb tense:pres polarity:neg ]
( I John [ class:proper_noun ]
  II Mary [ class:proper_noun ]
)

END:
```

Figure 6 – Exemple de structure syntaxique dans RealPro (manuel d'utilisation[4], p. 24)

Pour ce qui est du traitement des locutions, cela n'est pas abordé par ce réalisateur. Les locutions peuvent être traitées comme un seul nœud appartenant à la partie du discours analogue (`nom` au lieu de `locution nominale`, par exemple) ou comme plusieurs nœuds afin de permettre aux règles syntaxiques de s'appliquer. La responsabilité de créer la structure adéquate repose donc uniquement sur les épaules de l'usager.

1.2.2 Réalisateurs profonds

La présente section traite des réalisateurs profonds KPML, FUF, FLAUBERT, MARQUIS et FORGe. Ces réalisateurs prennent en entrée une représentation plus profonde (par exemple sémantique) que les réalisateurs de surface. La présentation de chacun d'entre eux est accompagnée de celle de sa grammaire et de sa théorie linguistique sous-jacente.

1.2.2.1 KPML

KPML (Bateman, 1997 ; Bateman et coll., 2005) est un réalisateur profond multilingue provenant de l'union de **KOMET** et de **PENMAN** (Bateman et coll., 1991 ; Mann, 1983 ; Mann et Matthiessen, 1982). Ce réalisateur est basé sur la **grammaire fonctionnelle systémique** (SFG pour *Systemic Functional Grammar* ; Matthiessen et Halliday, 1997). Selon cette perspective fonctionnaliste, tout choix linguistique est motivé par l'accomplissement d'une fonction. Son architecture prend donc la forme d'un réseau orienté où chaque embranchement représente un choix grammatical. La figure 7 présente l'organisation du système sous forme de réseau.

[4] http://cogentex.com/papers/realpro-manual.pdf (23/06/2021)

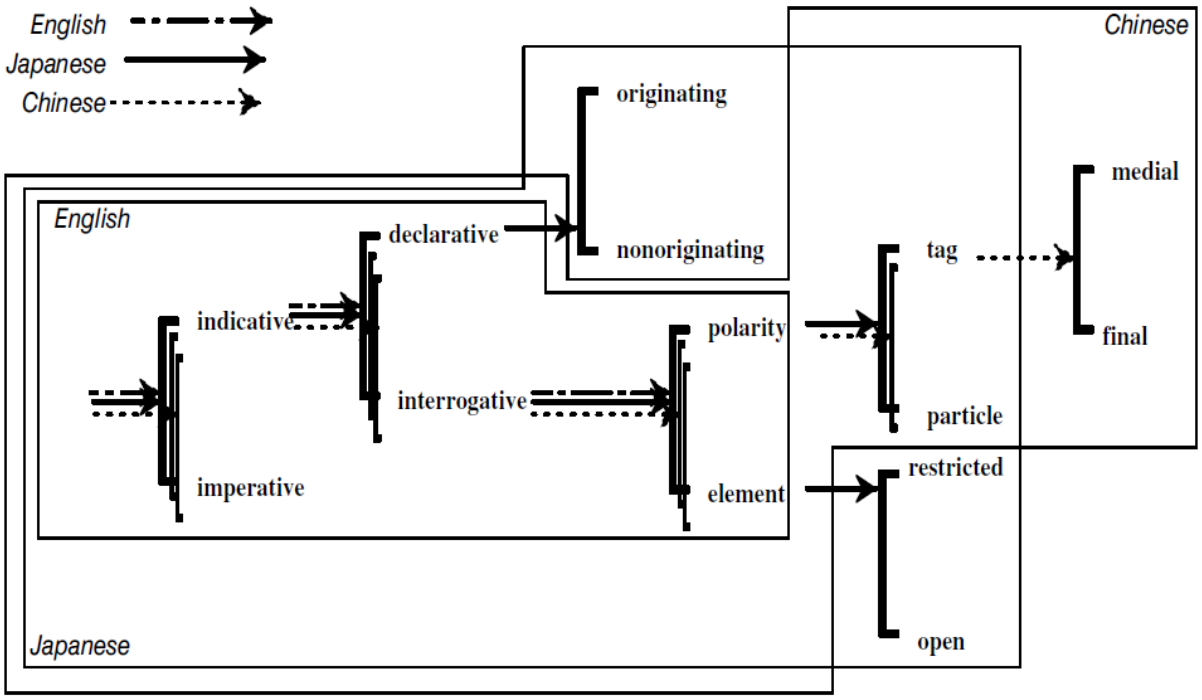


Figure 7 – Système en réseau multilingue (Bateman, 1997, p. 12)

Le réalisateur part d'une entrée en format SPL (de l'anglais *Sentence Planning Language;* Kasper, 1989), constituée d'une matrice d'attributs et de valeurs. Cette entrée profonde peut relever de la sémantique ou de la syntaxe. La figure 8 en présente un exemple.

```
((p0 / :english directed-action & :lex destroy
                                & :actor e0 & :actee b0
       :japanese nondirected-action & :lex collapse
                                    & :actor b0 & :cause e0
       :tense past)
 (e0 / object :lex earthquake)
 (b0 / object :lex building))
```

Figure 8 – Exemple d'entrée SPL dans KPML (Bateman, 1997, p. 15)

Cet exemple offre également un aperçu du traitement multilingue de KPML. Celui-ci offre une large couverture de l'anglais, de l'allemand et du néerlandais, ainsi qu'une base pour le français, le japonais et les combinaisons bilingues de toutes ses langues. La génération en anglais est faite grâce à la grammaire **NIGEL** (Mann et Matthiessen, 1983).

KPML est également doté d'un module de génération multilingue par patron. Celui-ci est activé par un mot-clé SPL et permet de récupérer des expressions auparavant générées ou des

expressions figées (par le style, la convention ou la législation). La figure 9 présente un exemple où cette génération est utilisée en combinaison avec le conditionnement multilingue et les annotations.

```
(e / template
:pattern :english ("Host " n " is unreachable")
:german ("Host " n " ist nicht erreichbar")
:actor (n / host ...
:annotate "http://www...."))
```

Figure 9 – Exemple de génération multilingue par patron (Bateman, 1997, p. 16)

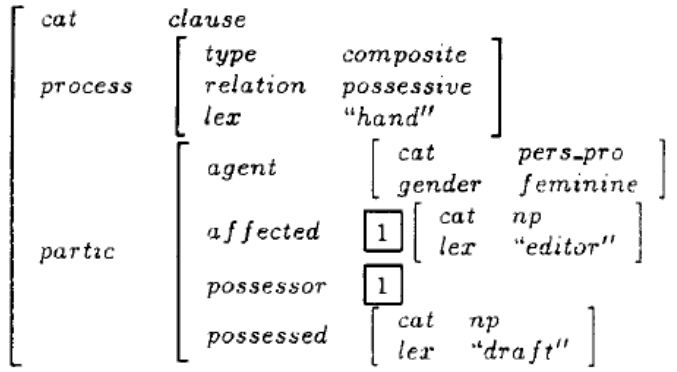
Bien que cela puisse être économique, la création des patrons et les choix qui en découlent reviennent tout de même à l'utilisateur.

1.2.2.2 FUF/SURGE

FUF, dont le nom réfère au Formalisme d'Unification Fonctionnel, est un réalisateur de texte basé sur la **grammaire d'unification fonctionnelle** (FUG). Il est utilisé avec une grammaire de l'anglais nommée **SURGE** (pour *Systemic Unification Realization Grammar of English* ; Elhadad et Robin, 1996, 1997).

Sa structure d'entrée est un arbre thématique non linéarisé composé de traits encodés en FUF. Ces traits sont appelés des descriptions fonctionnelles et seules les unités sémantiquement pleines sont représentées. Ce formalisme ne nécessite donc pas la création d'un dictionnaire, comme les entrées lexicales et leurs propriétés sont insérées à même les descriptions fonctionnelles. La figure 10 en présente un exemple.

Input Specification (I_1):



Output Sentence (S_1): "She hands the draft to the editor"

Figure 10 – Exemple de structure d'entrée et de sortie dans SURGE (Elhadad et Robin, 1996, p. 2)

Les descriptions fonctionnelles sont également utilisées pour décrire la grammaire interne du système SURGE. Celle-ci est responsable de l'application des règles morphosyntaxiques et de la linéarisation. Elle produit ainsi une phrase exprimant le sens désiré qui respecte les contraintes de la grammaire.

SURGE permet de générer des paraphrases et d'alterner entre elles, tout en prévenant la surgénération de phrases agrammaticales. Cependant, elle ne précise pas de règles concernant les locutions et leur traitement.

1.2.2.3 FLAUBERT/G-TAG

FLAUBERT (Meunier et Danlos, 1998) est un réalisateur de texte bilingue (français et anglais) implémenté grâce au formalisme **G-TAG** (Danlos, 1998, 2000 ; Danlos et coll., 2014). La figure 11 en présente l'architecture.

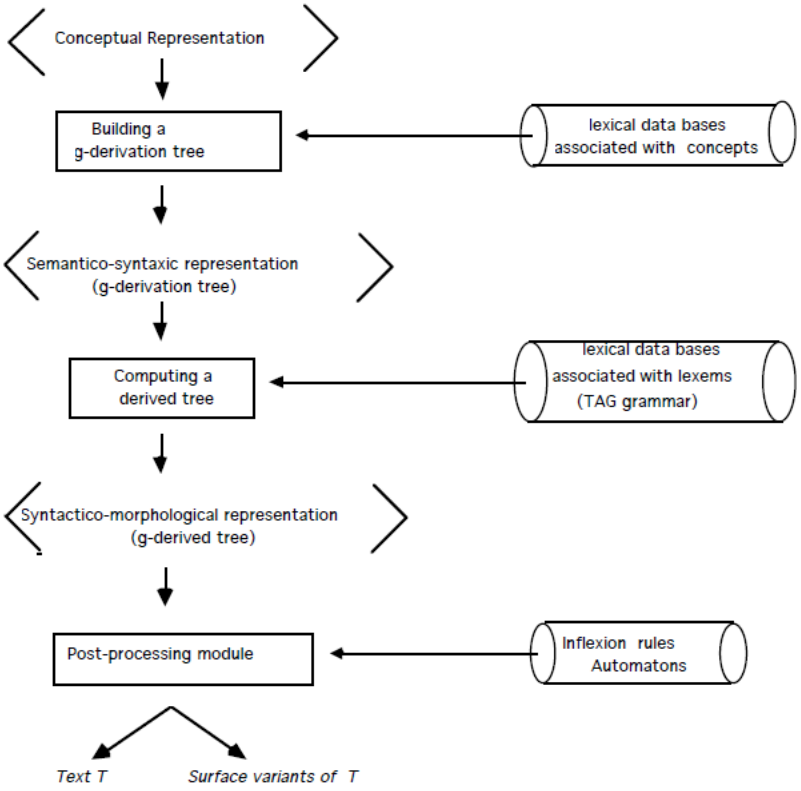


Figure 11 – Architecture de G-TAG (Danlos, 2000, p. 3)

FLAUBERT est inspiré du formalisme de la grammaire lexicalisée d'arbres adjoints (TAG, pour *Tree-Adjoining Grammar*), qui représentent la phrase par une combinaison d'arbres élémentaires. Ceux-ci sont présentés à la figure 12. Nous reprenons ici l'éloquente vulgarisation de G-TAG faite par Steinlin (2003) à l'aide de la phrase « une cerise est dégustée par René ». À ces arbres s'ajoutent des opérations de substitution (flèches a et b) et d'adjonction (flèche c), illustrées à la figure 13.

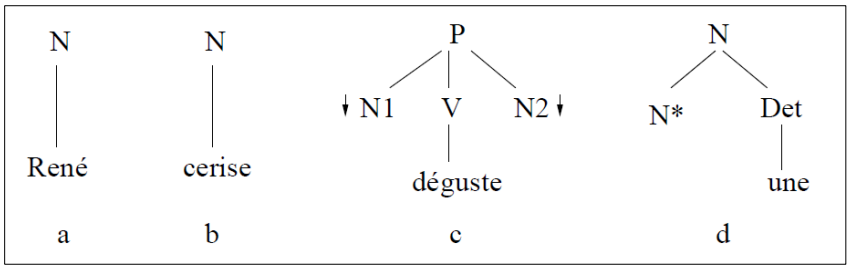


Figure 12 – Arbres élémentaires TAG (Steinlin, 2003, p. 27)

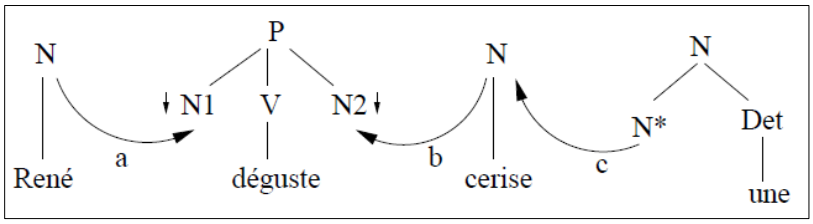


Figure 13 – Opérations de substitution et d'adjonction (Steinlin, 2003, p. 28)

Ces opérations sont décrites grâce à des *arbres de dérivation* (b) qui mènent à la création des *arbres dérivés* (a), chacun illustré dans la figure 14.

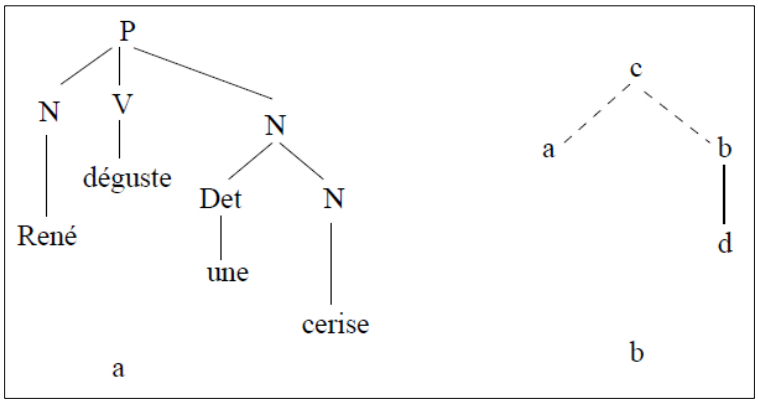


Figure 14 – Arbre dérivé et arbre de dérivation (Steinlin, 2003, p. 28)

G-TAG s'appuie sur ces arbres pour créer des *arbres de g-dérivation*, une représentation sémantico-syntaxique, et des *arbres g-dérivés*, une représentation syntactico-morphologique. La figure 15 et 16 en présente un exemple.

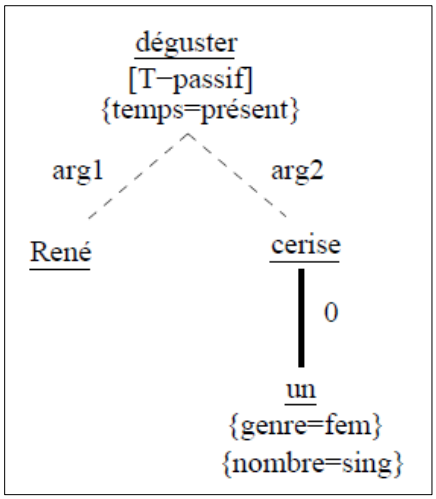


Figure 15 – Exemple d'arbre de g-dérivation (Steinlin, 2003, p. 29)

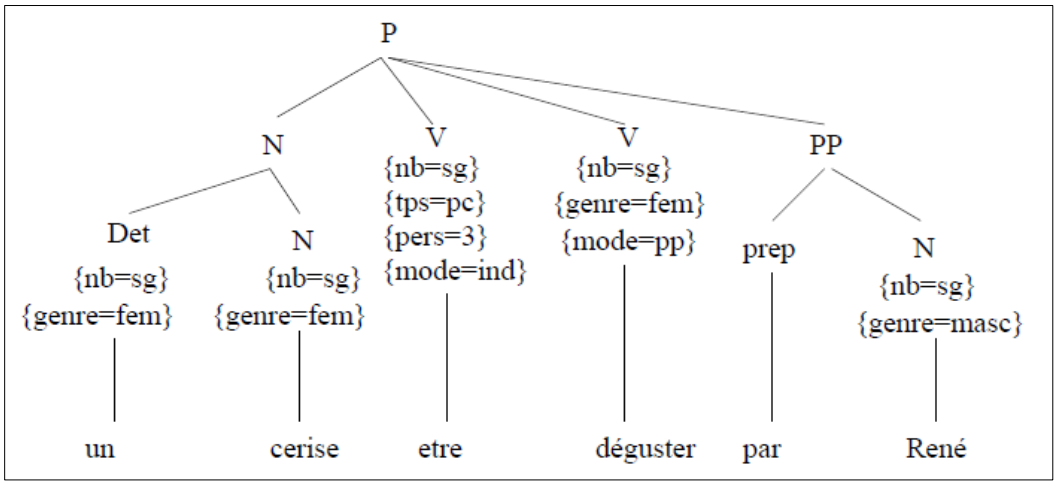


Figure 16 – Exemple d'arbre g-dérivé (Steinlin, 2003, p. 30)

Les bases de données de ces formalismes sont lexicalisées ; ils dépendent donc de l'interface conceptuelle-sémantique. FLAUBERT prend ainsi en entrée une représentation conceptuelle (voir figure 17), conçu par l'utilisateur à l'aide d'un menu déroulant.

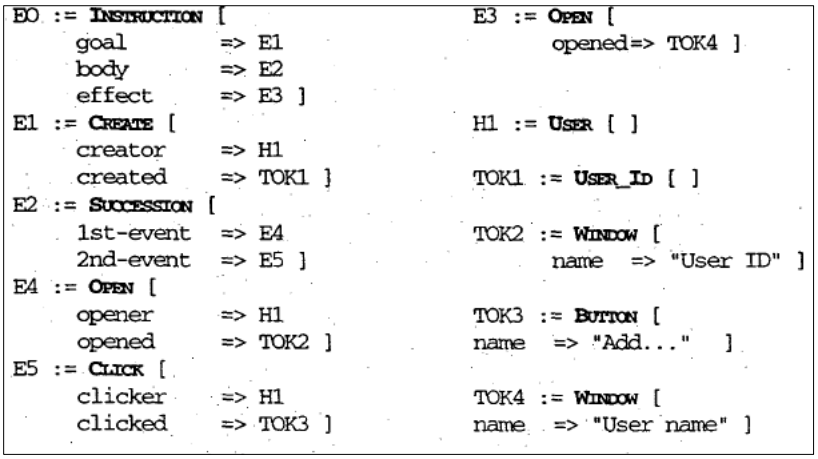


Figure 17 – Exemple de représentation conceptuelle dans FLAUBERT

(Meunier et Danlos, 1998, p. 286)

Puis, à partir de l'arbre g-dérivé et de l'arbre de g-dérivation, FLAUBERT produit les phrases suivantes :

En français : `Pour créer un identificateur d'utilisateur, ouvrez la fenêtre "User ID" avant de cliquer sur le bouton "Add... ". La fenêtre "User name" s'ouvre.`

En anglais : `In order to create an user ID, open the "User ID" window. Afterwards, click on the "Add..." button. The "User name" window is opened.`

À notre connaissance, il n'existe pas de réalisateur basé sur G-TAG qui offre un traitement particulier des locutions.

1.2.2.4 MARQUIS

MARQUIS (pour *Multimodal Air QUality Information Service for General Public* ; Lareau et Wanner, 2007 ; Wanner, Bohnet, et coll., 2007 ; Wanner et coll., 2010 ; Wanner, Nicklaß, et coll., 2007 ; Wanner et Lareau, 2009) est un générateur de texte complet, qui produit notamment des bulletins météorologiques multilingues sur la qualité de l'air de différents pays d'Europe. Pour ce domaine d'application, il couvre le français, l'anglais, l'espagnol, le catalan, l'allemand, le portugais, le polonais et le finnois.

Le module de réalisation de texte de MARQUIS est organisé selon la **TST** et se concentre sur la lexicalisation des collocations et des expressions les plus répandues dans la langue. La réalisation part d'une entrée sous forme de réseau conceptuel pour générer du texte. La figure 18 présente toutes les étapes de cette réalisation accompagnées de leur représentation pour une phrase donnée. Le passage entre les différents niveaux de représentations est assuré grâce à la grammaire et à l'aide de dictionnaires conceptuels, sémantiques et lexicaux.

Bien que ce réalisateur s'intéresse à la phraséologie, il ne porte pas d'attention particulière aux locutions. Le réalisateur offre un traitement simple des locutions, où un seul élément en syntaxe profonde est représenté par plusieurs éléments en syntaxe de surface. Ce genre de règle est notamment utilisé pour produire des verbes conjugués aux temps composés. Hormis cela, il traite généralement ces « termes complexes » comme des unités lexicales simples, c'est-à-dire comme un seul nœud syntaxique (Wanner et coll., 2010, p. 39).

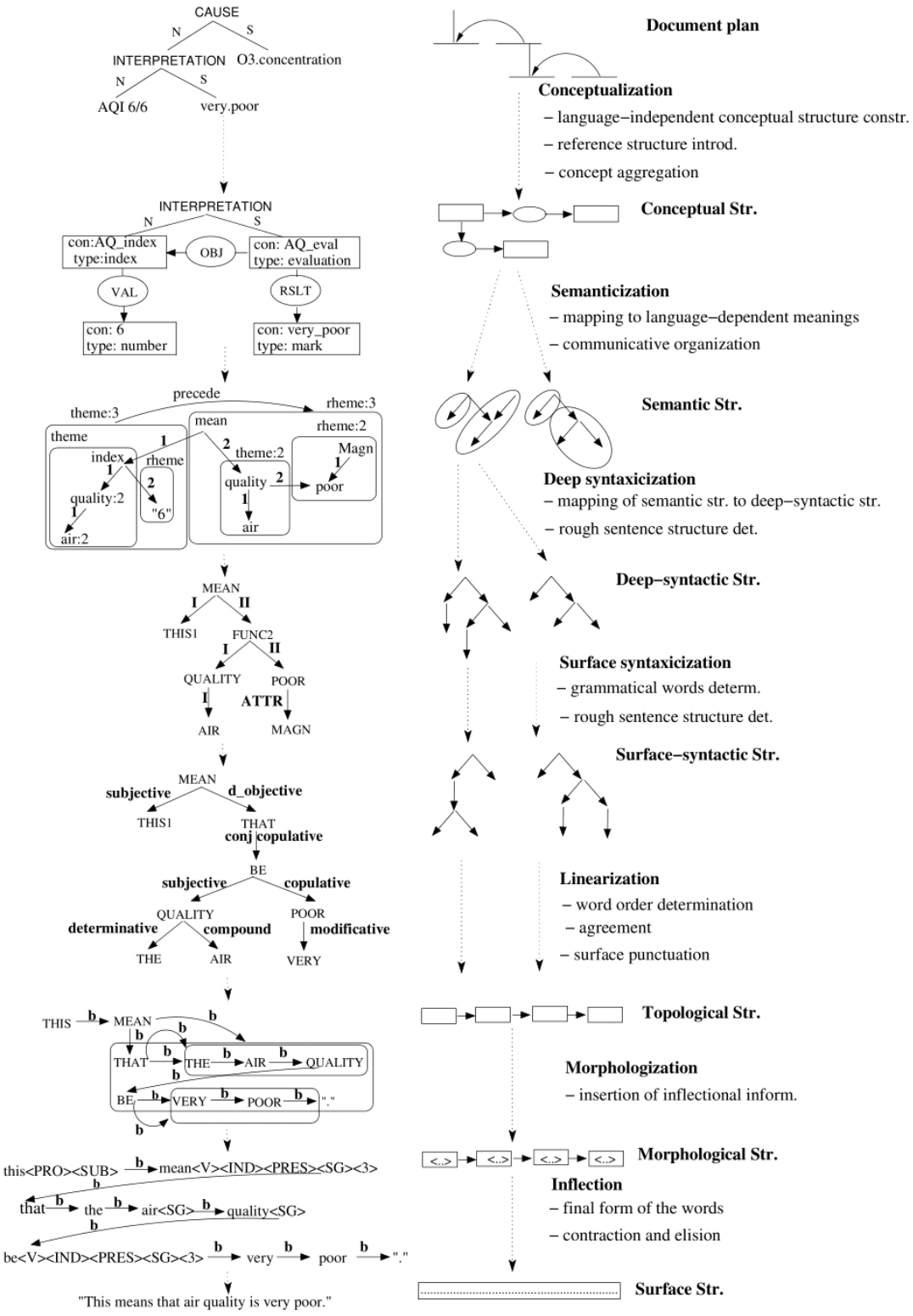


Figure 18 – Étapes de la réalisation linguistique dans MARQUIS (Wanner et coll., 2010, p. 36)

1.2.2.5 FORGe

FORGe (Mille et coll., 2017, 2019 ; Mille et Wanner, 2017) est un réalisateur profond qui s'appuie sur la transduction de graphes. Il suit également le formalisme de la **TST**. Il prend en entrée une représentation sémantique sous forme de graphe acyclique orienté (DAG). Sa grammaire assure ensuite le passage entre les différents niveaux de représentation subséquents ; puis, il linéarise le graphe final et produit du texte. La figure 19 présente un exemple de structure sémantique d'entrée pour ce réalisateur.

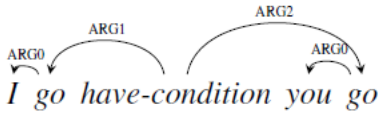


Figure 19 – Exemple d'entrée prédicat-arguments dans FORGe (Mille et coll., 2017, p. 921)

L'architecture modulaire de FORGe permet une implémentation facile de nouvelles langues. Son traitement multilingue s'étend actuellement à l'anglais, au français, à l'espagnol, à l'allemand et au polonais (Mille et Wanner, 2017). Il n'offre cependant aucun traitement particulier des locutions.

1.3 Synthèse

En conclusion, nous avons présenté quelques systèmes de réalisation de texte et leur prise en charge de la locution. Alors que les réalisateurs de surface ont l'avantage d'être faciles à développer et à utiliser, les réalisateurs profonds, pour leur part, requièrent une plus grande expertise linguistique ; cela augmente considérablement le temps et les coûts de production.

Nos recherches ont confirmé que le problème du traitement des locutions est souvent ignoré par ceux-ci. Tous les systèmes que nous avons étudiés imposent le fardeau du choix de la structure syntaxique des locutions aux utilisateurs. Plusieurs ne semblent pas apporter de traitement spécial aux expressions figées comme c'est aussi le cas pour les modèles statistiques et neuronaux.

Le prochain chapitre est consacré à la description de GenDR, le réalisateur profond dans lequel nous implémentons notre traitement des locutions. Nous y énonçons les raisons qui ont motivé le choix de ce réalisateur, soit sa théorie linguistique sous-jacente, son importante couverture phraséologique, sa concentration sur l'interface sémantique-syntaxe, ainsi que sa grammaire modulaire qui facilite l'intégration multilingue.

Chapitre 2 GenDR

GenDR (pour *Generic Deep Realizer*) est un **réalisateur de texte profond multilingue** (Lareau et coll., 2018) né du désir de modéliser divers phénomènes linguistiques présents dans différentes langues. Sa création a commencé par la génération des collocations, dans le projet **GÉCO** (Lambrey, 2016 ; Lambrey et Lareau, 2015 ; 2016), et il a depuis pris de l'expansion pour traiter davantage de phénomènes linguistiques, notamment les patrons de régimes (Galarreta-Piquette, 2018 ; He, 2021) et les propositions relatives (Portenseigne, à paraître). GenDR est qualifié de profond parce qu'il modélise l'interface sémantique-syntaxe. Il se concentre ainsi sur les opérations d'arborisation et de lexicalisation ayant lieu dans celui-ci. Il ne génère donc pas du texte en soi, mais produit plutôt diverses structures syntaxiques de surface grâce à une arborisation basée sur la saillance communicative. Comme il part d'une représentation profonde, notamment sémantique, il peut produire de nombreuses paraphrases.

Le cœur de GenDR reprend une bonne partie des règles de son précurseur **MARQUIS** (présenté à la section 1.2.2.3 ; Wanner et coll., 2010). Tout comme GenDR, celui-ci utilise la plateforme **MATE** (pour *Meaning-Text Development Environment* ; Bohnet et coll., 2000 ; Bohnet et Wanner, 2010) implémentée en Java, pour assurer la transduction des graphes. Ces trois systèmes sont basés sur la **TST** (Agel et coll., 2003 ; Kahane, 2003 ; Milićević, 2006 ; Žolkovsky et Mel'čuk, 1967) que nous présenterons plus en détail à la section 2.2.1.

Ce chapitre décrit premièrement l'architecture de GenDR, qui repose sur l'utilisation de dictionnaires, de grammaires et de graphes. Puis, nous présentons l'interface sémantique-syntaxe du réalisateur, qui s'appuie principalement sur les processus d'arborisation et de lexicalisation.

2.1 Architecture

GenDR est un réalisateur de texte profond qui opère dans l'interface sémantique-syntaxe (présentée à la section 2.2). Il passe donc de la sémantique à la syntaxe profonde avant d'arriver à la syntaxe de surface. La plateforme MATE permet de naviguer entre ces trois niveaux de

représentations. Concrètement, MATE est un transducteur de graphes conçu pour implémenter la TST dans le cadre de la génération automatique de texte. Il est ainsi possible de réaliser du texte tout en testant les fondements de la TST.

Grâce à cette plateforme, nous pouvons créer et modifier indépendamment les dictionnaires, les grammaires et les graphes. Alors que les dictionnaires encodent les propriétés des unités sémantiques et lexicales, les grammaires modélisent le passage d'un niveau de représentation à un autre. L'éditeur de graphes, quant à lui, permet de visualiser et de construire ces graphes. Voyons maintenant ces fonctions plus en détail.

2.1.1 Dictionnaires

À ce jour, GenDR compte trois types de dictionnaires. Le premier décrit les **sémantèmes** et leurs différentes lexicalisations. Le deuxième répertorie les **entrées lexicales**, ainsi que leurs propriétés. Ces deux dictionnaires sont propres à chaque langue traitée par GenDR. Le troisième, qui ne dépend d'aucune langue en particulier, décrit les **fonctions lexicales** (Lambrey, 2016 ; Lambrey et Lareau, 2015) ; à chaque fonction lexicale est associée une entrée de dictionnaire qui décrit sa sémantique et sa syntaxe.

Dans le cadre du présent mémoire, nous nous intéressons plus particulièrement au dictionnaire lexical. Alors que le **dictionnaire sémantique** est utile pour la mise en correspondance de la représentation sémantique (RSém) et de la représentation syntaxique profonde (RSyntP), le **dictionnaire lexical** permet la mise en correspondance des représentations syntaxiques profonde (RSyntP) et de surface (RSyntS). Les figures 20 et 21 montrent un exemple de cette correspondance pour le sémantème 'courtiser'. Comme on peut le voir, ce sémantème peut se réaliser autant par un simple lexème que par une locution. Le dictionnaire sémantique de GenDR permet donc de réaliser une grande quantité de paraphrases, comme un sémantème peut se réaliser par différents lexèmes et même par différentes parties du discours.

```
courtiser {
  lex = courtiser
  lex = "faire la cour"
  lex = draguer
  lex = cruiser
  lex = "faire du charme"
  lex = "faire du rentre-dedans"
  lex = fleureter
  lex = "conter fleurette"
  lex = galantiser
  lex = "faire le galant"
  lex = "faire du gringue"
}
```

Figure 20 – Exemple d'entrée du dictionnaire sémantique pour 'courtiser'

```
courtiser {
  dpos = V
  spos = verb
  gp = {
     1=I 2=II
     I={dpos=N rel=subj}
     II={dpos=N rel=dobj}
     }
}
```

Figure 21 – Exemple d'entrée du dictionnaire lexical pour COURTISER

Le dictionnaire lexical répertorie donc toutes les entrées lexicales et chacune d'entre elles est accompagnée de ses **informations**. Celles-ci comprennent :

- la partie du discours profonde (*dpos*) ;
- la partie du discours superficielle (*spos*) ;
- la diathèse ;
- le patron de régime (*gp*) ;
- les fonctions lexicales (*lf*).

La diathèse assure la correspondance entre les arguments sémantiques et syntaxiques. Le patron de régime, quant à lui, réfère à la description d'une construction syntaxique sélectionnée par un lexème (Galarreta-Piquette, 2018).

De plus, GenDR utilise un **mécanisme d'héritage** qui permet de factoriser l'information dans des classes d'éléments qui ont des caractéristiques communes. Ainsi, chacune des entrées du dictionnaire dépend d'une classe mère et hérite de ses traits. Par exemple, l'entrée de COURTISER, illustrée à la figure 21, renvoie à la classe des verbes transitifs directs, qui partagent le même

patron de régime. Il est également possible de bloquer cet héritage pour une entrée lexicale donnée.

GenDR propose une **approche multilingue** à la GAT. Il suit le principe du partage des ressources, notamment en créant des dictionnaires qui ne dépendent d'aucune langue précise et en réutilisant les classes d'informations entre ses différentes branches linguistiques. Jusqu'à présent, les travaux menés à l'aide de ce réalisateur se sont concentrés sur l'anglais, le français et le mandarin ; l'incorporation du lituanien et du persan a également été amorcée. Quantitativement, les branches linguistiques du réalisateur répertorient les 1 500 lemmes les plus fréquents du français et de l'anglais, 180 lexèmes du lituanien, 129 lexies du mandarin et 60 lexèmes du persan ; ce nombre restreint de lexèmes est tout de même suffisant pour mettre à l'épreuve le réalisateur.

2.1.2 Grammaires

La grammaire de GenDR comprend deux modules de règles de transduction de graphes. Le **module sémantique** (*1-sem-dsynt.rls* dans les figures ci-dessous) assure la correspondance entre les RSém et les RSyntP. Il est responsable de l'arborisation, qui s'effectue selon un modèle descendant (*top-down*). C'est également dans ce module que s'opère la lexicalisation profonde de l'entrée. Il y a donc une sélection des lexèmes qui représentent les sémantèmes donnés selon les conditions d'application des règles. Le **module syntaxique** (*2-dsynt-ssynt.rls* dans les figures ci-dessous) s'occupe, pour sa part, de la correspondance entre les RSyntP et les RSyntS. Il gère l'arborisation et la lexicalisation de surface en ajoutant les mots fonctionnels et les relations de surface. C'est dans ce module que se situe la contribution du présent mémoire. L'organisation modulaire de GenDR vient grandement faciliter le traitement multilingue, puisque les règles plus générales sont partagées entre les différentes branches linguistiques et que les autres règles servent de base de développement pour implémenter de nouvelles langues.

Les **règles des grammaires** respectent le **format** suivant. L'en-tête comprend l'interface dans laquelle la règle s'exécute, le nom de la règle, ainsi que l'identification du groupe auquel elle appartient. La partie gauche (*leftside*) contient le graphe en entrée. Conséquemment, la partie droite (*rightside*) comporte le graphe créé en sortie. Seuls les nœuds et les arcs directement

impliqués par la règle sont représentés dans les graphes. La section du dessous stipule les conditions d'application qui permettent à la règle de s'activer. Il y a également une seconde partie au bas de la fenêtre où il est possible de noter des commentaires pertinents concernant les règles. La figure 22 illustre l'organisation des règles de GenDR. La règle affichée dans cette figure (`lex_lu`) décrit la lexicalisation simple que nous présentons à la section 2.2.3. Pour une présentation plus détaillée des règles de GenDR, voir Lareau et Lambrey (2016).

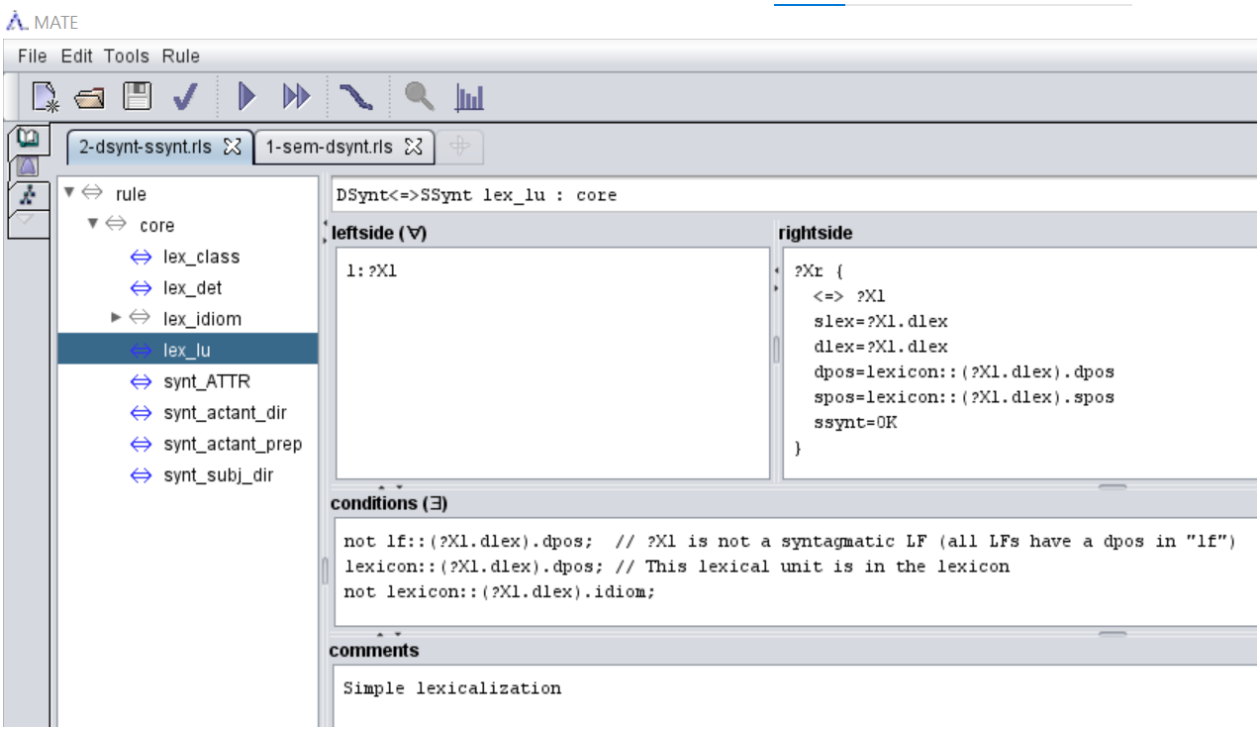


Figure 22 – Organisation des règles dans GenDR

2.1.3 Graphes

L'éditeur de graphes de GenDR permet de visualiser les trois niveaux de représentation. La **RSém** est un graphe où les arguments sont liés à leur prédicat par des relations numérotées selon leur position logique. Le graphe syntaxique en **RSyntP** suit la même logique, avec les arguments numérotés selon leur position actancielle (Meľčuk, 2012). La **RSyntS**, quant à elle, est construite à l'aide d'un graphe qui lie les arguments à l'aide de noms de relations (Mel'čuk, 1988). Les figures 23, 24 et 25 montrent des exemples en mode graphique et textuel pour la phrase *Renaud doit une patate à Ariane*.

```
structure Sem S1 {
  S:1 {
    devoir:1 {
      tense=PRES
      1-> Renaud:1
      2-> patate:1
      3-> Ariane:1
    }
    Renaud:1{class=proper_noun}
    patate:1 {number=SG definiteness=INDEF}
    Ariane:1{class=proper_noun}
    main-> devoir:1
  }
}
```

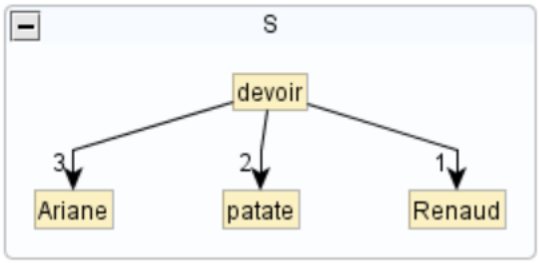
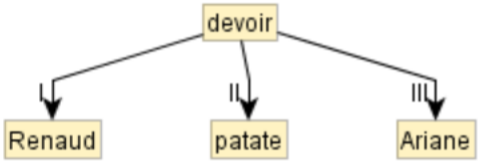


Figure 23 – Exemple de graphes sémantiques en mode graphique et textuel dans GenDR

```
structure DSynt S313_1_0 {
  devoir:1 {
    tense=PRES dsynt=OK finiteness=FIN
    dpos=V clause=main
    I-> Renaud:1{number=SG dsynt=OK dpos=N class=proper_noun
                 definiteness=DEF}
    II-> patate:1{dsynt=OK number=SG dpos=N defsiniteness=INDEF}
    III-> Ariane:1{number=SG dsynt=OK dpos=N class=proper_noun
                   definiteness=DEF}
  }
}
```




```
structure DSynt S313_1_0 {
  devoir:1 {
    tense=PRES dsynt=OK finiteness=FIN dpos=V clause=main
    I-> Renaud:1{number=SG dsynt=OK dpos=N class=proper_noun
                 definiteness=DEF }
    II-> "pomme de terre#1":1 {dsynt=OK number=SG dpos=N
                               definiteness=INDEF }
    III-> Ariane:1 {number=SG dsynt=OK dpos=N class=proper_noun
                    definiteness=DEF }
  }
}
```

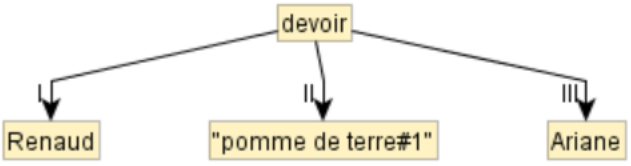
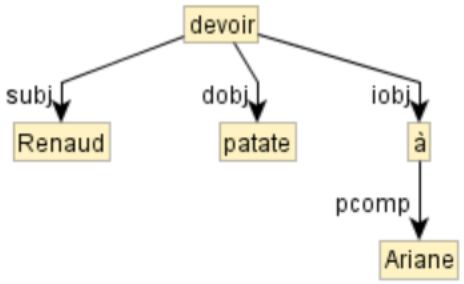


Figure 24 – Exemple de graphes syntaxiques profonds en mode graphique et textuel

```
structure SSynt S144_1_0 {
  devoir:1 {
    tense=PRES spos=verb finiteness=FIN
    dpos=V ssynt=OK
    dlex=devoir
    subj-> Renaud:1 {spos=proper_noun
                     number=SG
                     class=proper_noun
                     dpos=N ssynt=OK
                     definiteness=DEF }
    dobj-> patate:1 {spos=noun number=SG dpos=N ssynt=OK
                     definiteness=INDEF dlex=patate}
    iobj-> à:1 {split=top spos=preposition dpos=Adv
                pcomp-> Ariane:1{split=bottom number=SG
                                 spos=proper_noun dpos=N
                                 class=proper_noun
                                 definiteness=DEF ssynt=OK }
    }
  }
}
```




```
structure SSynt S142_1_0 {
  devoir:1 {
    tense=PRES spos=verb dpos=V
    finiteness=FIN ssynt=OK
    dlex=devoir
    subj-> Renaud:1{number=SG
                    spos=proper_noun
                    class=proper_noun
                    dpos=N
                    definiteness=DEF
                    ssynt=OK }
    dobj-> "pomme":1{number=SG spos=noun dpos=N
                     definiteness=INDEF ssynt=OK
                     dlex=" pomme de terre#1"
                     relx-> "de":1 {ssynt=OK
                                    relx-> "terre":1 {ssynt=OK }
                     }
    }
    iobj-> à:1 {split=top spos=preposition dpos=Adv
                pcomp-> Ariane:1{split=bottom number=SG
                                 spos=proper_noun dpos=N
                                 class=proper_noun
                                 definiteness=DEF ssynt=OK }
    }
  }
}
```

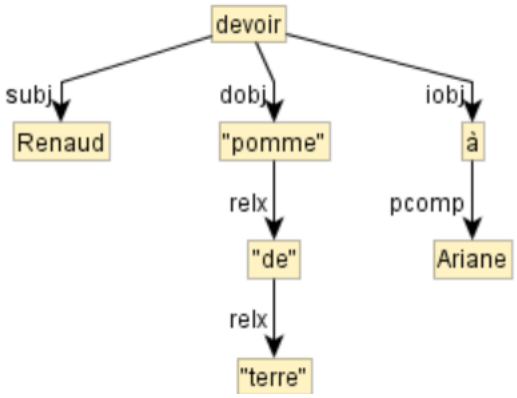


Figure 25 – Exemple de graphes de surface en mode graphique et textuel

MATE permet d'éditer tous les graphes séparément à chacun des niveaux de représentation. Lorsque le graphe de départ est une RSém, cela permet d'avoir une entrée qui n'est pas régie par les règles syntaxiques et qui ainsi se prête mieux à la paraphrase. Grâce à cela, il est possible d'avoir plusieurs résultats en sortie. Par exemple, le sémantème 'patate' (voir figures précédentes) peut être représenté à la fois par le lexème PATATE et la locution ⌜POMME DE TERRE⌝. Ces différentes lexicalisations sont encodées dans le dictionnaire sémantique.

2.2 Interface sémantique-syntaxe

Pour mieux comprendre les notions décrites dans la présente section, il est essentiel dresser le portrait du cadre théorique dans lequel s'inscrit GenDR.

2.2.1 Théorie Sens-Texte

La TST est une théorie linguistique visant la description de la correspondance entre le ***Sens*** et les énoncés, appelés ***Textes*** (Mel'čuk, 1997 ; Žolkovsky et Mel'čuk, 1967). Cette théorie est par nature computationnelle comme elle s'appuie sur des **modèles formels** qui se prêtent bien à la GAT. La figure 26, reprise de Polguère (1998), illustre la structure fonctionnelle d'un modèle Sens-Texte.

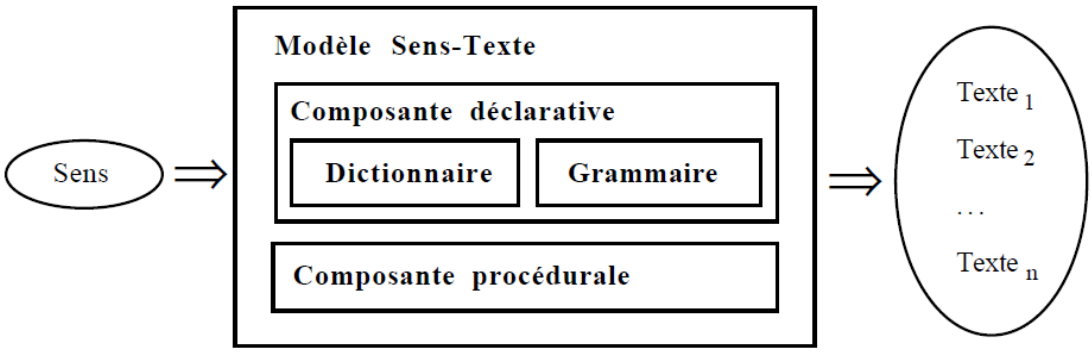


Figure 26 – Structure fonctionnelle d'un modèle Sens-Texte (Polguère, 1998, p. 4)

Ce modèle comporte six **modules de correspondance** Sens-Texte qui regroupent des règles linguistiques pour assurer le passage entre des **niveaux de représentation** adjacents. Ceux-ci couvrent les branches principales de la linguistique, soit la sémantique, la syntaxe, la morphologie, la phonologie et la phonétique. Les niveaux subséquents de syntaxe, morphologie

et phonétique sont scindés en représentations profondes ou de surface. On compte au total sept niveaux. Les différentes représentations sont illustrées dans le tableau 1, accompagnées de leurs formalismes respectifs.

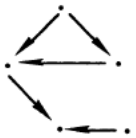
Représentations	Formalismes
Repr. sémantique (RSém)	 **Réseau sémantique** : les nœuds représentent des sens et les arcs des relations prédicat-argument.
Repr. syntaxique profonde (RSyntP)	 **Arbre de dépendance non linéairement ordonné** : les nœuds représentent des lexies pleines et les arcs des dépendances syntaxiques profondes (universelles).
Repr. syntaxique de surface (RSyntS)	 **Arbre de dépendance non linéairement ordonné** : les nœuds représentent des lexies (pleines ou vides) et les arcs es dépendances syntaxiques de surface (liées à la langue en question).
Repr. morphologique profonde (RMorphP)	**Chaîne de lexies marquées morphologiquement**
Repr. morphologique de surface (RMorphS)	**Chaîne de morphèmes**
Repr. phonologique profonde (RPhonP)	 **Chaîne de phonèmes**
Repr. phonologique de surface (RPhonS)	 **Chaîne de phones**

Tableau 1 – Niveaux d'un modèle Sens-Texte (Polguère, 1998, p. 6 ; notre adaptation)

Le niveau le plus profond est celui de la **représentation sémantique** (RSém). Les sémantèmes y sont liés grâce aux relations prédicat-argument. Les niveaux subséquents sont la **représentation syntaxique profonde** (RSyntP) et la **représentation syntaxique de surface** (RSyntS). Ces deux niveaux s'appuient sur des arbres de dépendance qui, contrairement aux arbres de constituants, ne sont pas ordonnés linéairement. La RSyntP ne comprend que les lexies sémantiquement pleines alors que la RSyntS comporte aussi les mots-outils. C'est en RSyntS que les relations syntaxiques propres à une langue sont définies. Puis, à partir de la **représentation morphologique**

profonde (RMorphP), les graphes sont linéarisés sous forme de chaînes. La RMorphP contient des lexies et de traits grammaticaux, alors que la **représentation morphologique de surface** (RMorphS) contient des morphèmes. Ceux-ci font place aux phonèmes en **représentation phonologique profonde** (RPhonP) et aux phones en **représentation phonologique de surface** (RPhonS).

La TST postule six modules pour assurer la correspondance entre chacun des niveaux de représentation. Dans le cadre de notre travail, nous nous intéressons aux deux modules de l'interface sémantique-syntaxe et plus précisément au module qui assure la correspondance entre la RSyntP et la RSyntS, où la lexicalisation de surface des locutions a lieu. La prochaine section présente les deux processus en jeu dans cette interface, soit l'arborisation et la lexicalisation.

2.2.2 Arborisation

L'arborisation est le processus par lequel, à partir d'un graphe sémantique acyclique (DAG), on construit un arbre syntaxique profond. En sémantique, les prédicats sont liés à leurs arguments grâce à des arcs numérotés. La structure syntaxique, quant à elle, rassemble l'ensemble des liens de dépendances fonctionnelles entre les unités lexicales d'un énoncé (Mel'čuk, 1988 ; Tesnière, 1959).

L'**arborisation profonde** demande d'appliquer des règles de correspondance sémantique. Comme mentionné précédemment, dans GenDR, ces règles suivent un algorithme *top-down* piloté par la structure communicative. Cette méthode procédurale est expliquée par Lareau et coll. (2018) et basée sur l'algorithme de Polguère (1990). Dans le cadre du présent mémoire, nous nous intéressons tout particulièrement à l'interface entre les deux niveaux de représentation syntaxique (profond et de surface), où l'arborisation a déjà eu lieu. Nous ne présentons donc sommairement que l'arborisation.

Le premier élément à être créé est le plus saillant : la racine. Une fois celle-ci créée, elle est lexicalisée, puis l'information concernant le patron de régime et la diathèse est récupérée. Ce processus est par la suite répété récursivement pour chacun des arguments de la racine, jusqu'à l'atteinte d'un graphe complet. L'arborisation est donc faite de concert avec la lexicalisation dans

GenDR (Galarreta-Piquette, 2018 ; Lareau et coll., 2018) : un nœud est créé, il est lexicalisé et ses règles actancielles sont appliquées ; puis, on passe au prochain nœud s'il y a lieu. Cela complète l'arborisation profonde. Par la suite, l'**arborisation de surface** et la lexicalisation de surface apposent une étiquette sur ces relations numérotées selon les liens de dépendances existants entre les nœuds. Par exemple, le premier argument d'un verbe porte le nom de relation *sujet*. Puis, les prépositions et conjonctions régies sont ajoutées selon le régime des lexies qui les gouvernent. Pour un exemple étapes par étapes de la construction de l'arbre syntaxique profond, voir Galarreta-Piquette (2018) et Lareau et coll. (2018).

2.2.3 Lexicalisation

La lexicalisation dans GenDR implique trois niveaux de représentation : la sémantique, la syntaxe profonde et la syntaxe de surface. La première étape de la lexicalisation fait appel aux deux premiers niveaux de représentation. Cette **lexicalisation profonde** (`dlex`) consiste à choisir la ou les unités lexicales[5] qui expriment le sémantème donné. Les unités sémantiquement pleines et les verbes supports sont alors lexicalisés. La seconde étape est celle de la **lexicalisation de surface** (`slex`), qui a lieu entre la RSyntP et la RSyntS. C'est à ce moment que sont introduits les mots fonctionnels et les lexies de surface.

Il existe six types de lexicalisation dans GenDR, qui ont été décrits par Lareau et coll. (2018) et que nous résumons ici. La **lexicalisation simple** part d'un sémantème donné et fouille dans le dictionnaire sémantique pour y récupérer ses expressions lexicales. Cette sélection dépend de la partie du discours profonde imposée au nœud (le plus souvent, par le régime de son gouverneur) et permet d'augmenter le pouvoir de paraphrasage du système. Par la suite, la lexicalisation de surface ne fait que reprendre les informations de la lexicalisation profonde. Pour chacune des langues traitées, le réalisateur précise l'étiquette de surface attribuée à l'étiquette profonde.

Le mécanisme d'héritage de GenDR, présenté précédemment, rend aussi possible la **lexicalisation par classe**. Ces classes comprennent les nombres, les unités, les noms propres, les villes, les

[5] Ce choix peut s'opérer au sein d'un ensemble de synonymes, notamment de simples lexèmes, des locutions ou des collocations, mais il peut également se faire parmi différentes parties du discours. Par exemple, 'causer (x, y)' peut être exprimé par *x cause y* au même titre que *x est la cause de y*, *y a eu lieu à cause de/en raison de x*, *y est dû à x*', etc. (Lareau et coll., 2018).

acronymes, etc. Puisque ces classes contiennent un grand nombre d'éléments ayant un comportement prévisible, cela ne vaut pas la peine de créer une entrée pour chacun d'entre eux. Le nœud ayant un trait `class` va plutôt chercher sa partie du discours et ses traits grammaticaux dans l'entrée de la classe en question. Ces informations sont également transmises lors de la lexicalisation de surface subséquente.

La **lexicalisation grammaticale**, quant à elle, est seulement présente dans le deuxième module de règles. Elle permet d'insérer les mots-outils et les lexies fonctionnelles dans la RSyntS. En effet, la RSyntP est exempte de mots-outils ; les auxiliaires et les déterminants sont plutôt représentés comme des traits grammaticaux sur les noms et les verbes. Une règle a été créée pour chacune de ces entrées, puisqu'il n'en existe que très peu. De plus, cette lexicalisation ajoute les lexies fonctionnelles, dont font partie les prépositions régies et les complémenteurs. Ces lexies sont dites fonctionnelles, car elles sont imposées par le patron de régime d'autres lexies. Elles viennent s'insérer entre un gouverneur et son dépendant. Pour plus d'informations, voir Galarreta-Piquette (2018).

La **lexicalisation liée** est plus complexe. Elle permet la lexicalisation des collocations, qui sont définies par la relation privilégiée existant entre une base et son collocatif. Ces relations sont décrites par des fonctions lexicales, qui sont regroupées selon les patrons apparents. Les règles de lexicalisation liée profonde agissent comme des patrons où les nœuds et leurs relations se voient attribuer un pantonyme comme étiquette. La lexicalisation de surface vient par la suite récupérer le collocatif dans l'entrée lexicale de la base. Pour en savoir plus sur l'implémentation des règles concernant les fonctions lexicales, consulter Lambrey (2016) ainsi que Lambrey et Lareau (2015)[6].

Bien que les locutions et les collocations appartiennent toutes deux à la phraséologie, elles diffèrent dans leur mécanisme de lexicalisation. Au chapitre 3, nous reviendrons plus en détail sur la place de celles-ci dans la phraséologie. Dans le module sémantique, qui assure la correspondance entre la RSém et la RSyntP, les locutions sont lexicalisées grâce à la lexicalisation

[6] Pour plus amples informations au sujet des collocations en TAL, voir Apresi͡an et coll. (2000), Apresjan et coll. (2007), Fonseca et coll. (2016), Jousse (2010), Kahane et Polguère (2001), Lambrey (2016), Mel'čuk (1995, 1996, 1998, 2007, 2014), Polguère (2007), Wanner (1996), ainsi que Žolkovsky et Mel'čuk (1967).

simple. Puis, tout comme dans le cas de la lexicalisation grammaticale, de nouveaux nœuds sont insérés dans la RSyntS. Cependant, cette nouvelle lexicalisation ne vient pas qu'insérer un nœud, mais bien un patron lexico-syntaxique, semblablement à ce que fait la lexicalisation liée. Malgré ces similitudes, les locutions demandent néanmoins leur propre type de lexicalisation. Cela constitue d'ailleurs le cœur de la problématique à laquelle le présent mémoire tente de répondre.

La **lexicalisation par patron** est un type de lexicalisation de surface. Elle est activée lorsque l'entrée en RSyntP est un nœud portant le trait `idiom`. Cette lexicalisation exige un patron générique décrivant la RSyntS de la locution, ainsi qu'une liste de ses lexèmes et des relations entre ceux-ci. Chaque patron générique est doté de sa propre règle de lexicalisation. Celle-ci est un arbre à trous rempli grâce à l'information trouvée dans les entrées du dictionnaire lexical. La section 4.1.2 donne une description détaillée des règles de transduction associées à l'arborisation et à la lexicalisation des locutions.

La **lexicalisation de secours** s'active lorsqu'aucune des cinq lexicalisations précédentes ne s'applique à un nœud. S'il y a une entrée dans le dictionnaire lexical, il est présumé que l'entrée du dictionnaire sémantique a simplement été omise et la lexicalisation simple s'applique. Dans le cas contraire, la lexicalisation de secours essaie de deviner la partie du discours à laquelle appartient le lexème selon la langue traitée. S'il existe une contrainte associée au nœud, elle lui attribue la partie du discours désirée, et sinon la partie du discours la plus probable. Plus précisément, elle la traite comme un nom, car cette catégorie est nettement plus fréquente que les autres. Qui plus est, le programme signale chacune des informations manquantes ; il est donc possible de filtrer ces nœuds non identifiés ou de les post-traiter au besoin.

En conclusion, la modélisation de l'interface sémantique-syntaxe en trois étapes permet au réalisateur de faire un meilleur traitement des processus tels que la lexicalisation. Avant d'aborder le cœur de notre implémentation, nous consacrons le prochain chapitre à la définition de la locution au sein de la phraséologie et à la présentation de notre corpus.

Chapitre 3 Locutions : définition, corpus et classification

Dans les chapitres précédents, nous avons décrit la branche du TAL dans laquelle s'inscrit notre étude, soit la GAT, ainsi que le réalisateur de texte dans lequel nous avons implémenté notre traitement. Le présent chapitre est consacré à la description du phénomène linguistique qui nous intéresse : la locution. Pour ce faire, nous faisons un tour d'horizon de la **phraséologie** dans le cadre de la TST pour distinguer les différents types de **phrasèmes**. Nous définissons ensuite la locution et énonçons ses principales caractéristiques. Une fois celle-ci bien circonscrite, nous présentons notre corpus, le RL-fr et ses **patrons syntaxiques linéarisés**.

3.1 Définition

Afin de définir notre objet d'étude, la locution, nous passons d'abord en revue les différents phrasèmes pour ensuite la distinguer de ceux-ci.

3.1.1 Phraséologie dans le cadre de la théorie Sens-Texte

La **phraséologie** est l'ensemble des expressions non libres d'une langue ; ces unités sont appelées des **phrasèmes**. Mel'čuk (1997, 2004) définit un phrasème comme une association contrainte d'au moins deux unités. Ces unités peuvent être de nature lexicale ou morphologique ; elles forment alors des phrasèmes syntagmatiques ou des phrasèmes morphologiques, respectivement. Les **phrasèmes morphologiques** (aussi appelés quasi-morphes) sont plutôt marginaux en français et ne font pas partie de notre étude ; nous nous intéressons uniquement aux phrasèmes syntagmatiques. La figure 27 en présente la typologie selon Mel'čuk (2015, p. 68 ; notre traduction).

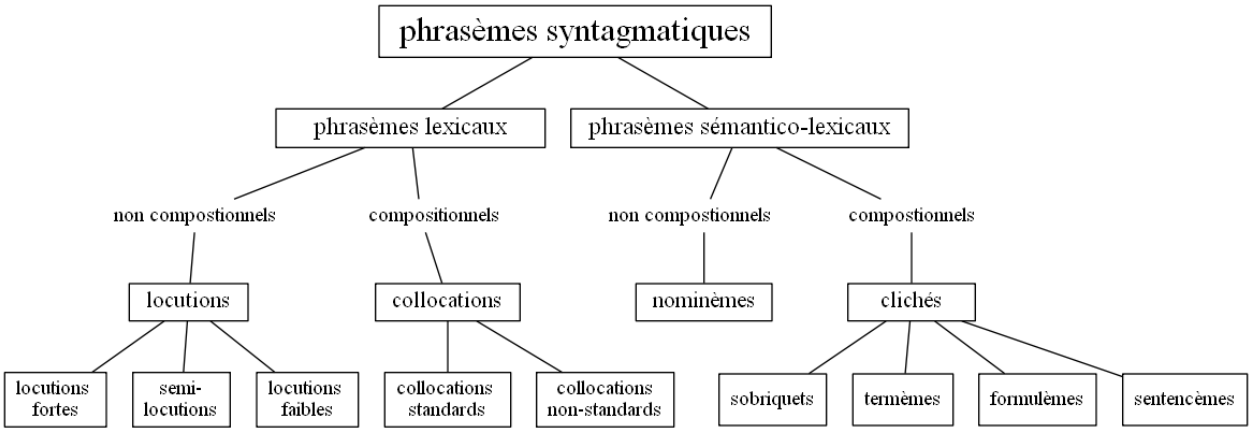


Figure 27 – Typologie générale des phrasèmes lexicaux (Mel'čuk, 2015, 2015, p. 68)

Les **phrasèmes syntagmatiques** se divisent d'abord en deux grands groupes en fonction de la nature de leurs contraintes phraséologiques de sélection. D'une part, le sens des **phrasèmes lexicaux** est assemblé librement, mais le choix des lexèmes pour l'exprimer est contraint. D'autre part le sens des **phrasèmes sémantico-lexicaux** n'est pas construit ; il est plutôt sélectionné comme un tout en fonction du message conceptuel à transmettre (Mel'čuk, 2013). La compositionnalité de ces deux groupes de phrasèmes permet de les distinguer davantage. Rappelons qu'un segment est sémantiquement compositionnel si son sens global est équivalent à la combinaison du sens de toutes ses composantes (Polguère, 2015). Si un autre sens que celui de ses composantes se dégage du segment, alors celui-ci n'est pas compositionnel.

Dans un premier temps, les phrasèmes sémantico-lexicaux peuvent être **non compositionnels** (des nominèmes) ou **compositionnels** (des clichés linguistiques). . Les **nominèmes** comprennent les noms propres composés comme *la Grande Ourse* (Iordanskaja et Mel'čuk, 2017). Les **clichés linguistiques** se distinguent selon deux axes perpendiculaires : l'axe *spécifique-générique* et l'axe *concret-abstrait*. Un **sobriquet** est utilisé pour référer à une entité concrète et spécifique ; c'est ce qu'on appelle communément un surnom, par exemple VILLE LUMIÈRE pour désigner Paris. Le **termème**, quant à lui, a également comme référent une entité concrète, mais celle-ci est générique ; par exemple, la MALADIE D'ALZHEIMER qui fait référence à un type de démence. Pour ce qui est du **formulème**, son référent est abstrait et spécifique ; il est contraint par la pragmatique d'une langue. Par exemple, en français « Quel âge avez-vous/as-tu ? » est la manière prototypique de demander l'âge d'une personne. Enfin, le **sentencème** fait référence à une classe

de situations, soit un référent abstrait et générique. Par exemple, l'expression « *Jamais deux sans trois* » réfère au fait que les choses apparaissent généralement sous forme de triade. Pour de plus amples détails, consulter Mel'čuk (2015).

Dans un deuxième temps, les **phrasèmes lexicaux** peuvent aussi être **non compositionnels** (des locutions) ou **compositionnels** (des collocations). La **collocation** est définie comme « un syntagme AB (ou BA) qui est tel que, pour le construire, le Locuteur sélectionne A librement d'après son sens 'A', alors qu'il sélectionne B pour exprimer auprès de A un sens 's' en fonction de contraintes imposées par A [= la combinatoire restreinte de A]. » (Polguère, 2016, p. 64). Cela signifie que la base est sélectionnée librement et qu'elle conditionne le choix du collocatif. Par exemple, *grièvement blessé* est une collocation puisque la base, *blessé*, est sélectionnée librement et le collocatif, *grièvement*, est choisi en fonction de cette base. Nous donnerons une description détaillée des locutions dans la prochaine section.

Pour compléter le portrait des phrasèmes, il faut également mentionner deux cas particuliers d'expressions figées non libres, soit les pragmatèmes et les proverbes. Les **pragmatèmes** sont des expressions lexicales contraintes par la situation dans laquelle elles sont employées. Les pragmatèmes sont généralement des clichés linguistiques. Cependant ils peut aussi s'agir de collocations (*peinture fraîche*, sur un panneau), de locutions (⌜À QUI DE DROIT⌝, dans l'en-tête d'une lettre officielle) ou de lexèmes (la commande militaire REPOS !), selon leurs contraintes (Mel'čuk, 2013).

Comme les locutions, les **proverbes** sont sémantiquement non compositionnels. Par contre, ils sont souvent moralisateurs et visent à exprimer une vérité vérifiée par l'expérience (sentence). Par exemple, *Pierre qui roule n'amasse pas mousse* signifie 'à force de changer de situation ou de voyager, on n'amasse pas de richesse' (Pausé, 2017). Ils se différencient des locutions phrastiques, qui sont plutôt situationnelles et dépendent du contexte. Par exemple, ⌜LES CAROTTES SONT CUITES.⌝ signifie 'Il est trop tard, on ne peut plus revenir en arrière.' (Pausé, 2017). Les proverbes s'apparentent aux dictons, mais ces derniers n'ont pas de contenu moralisateur. Ils expriment plutôt un lien de cause à effet (par exemple, entre la température et le mois de l'année : *En avril, ne te découvre pas d'un fil…*) et dépendent généralement du contexte.

3.1.2 Locutions

Comme le montre la section précédente, les **locutions** sont par définition des syntagmes non compositionnels : leur sens global ne peut être reconstitué par l'addition des sens de leurs constituants. Il existe trois types de locutions, qui se distinguent par leur degré de non-compositionnalité ; ils sont définis dans le tableau 2 :

Classification sémantique	Définition	Exemple
locution forte	le sens global de la locution ne contient le sens d'aucun de ses constituants	⌜PRENDRE LA MOUCHE⌝ 'se fâcher brusquement'
semi-locution	le sens global de la locution contient le sens d'un de ses constituants, qui n'est pas le pivot sémantique de la locution	⌜GARDER LE LIT⌝ 'rester allongé sur son lit du fait d'une maladie'
locution faible	le sens global de la locution contient le sens de tous ses constituants, mais aucun n'est le pivot sémantique de la locution	⌜COQ AU VIN⌝ 'plat constitué de viande de coq cuisinée au vin'

Tableau 2 – Classification sémantique des locutions

Cette classification sémantique des locutions s'appuie sur la notion de pivot sémantique. Mel'čuk (2013) stipule que, dans une définition analytique, le sens 's' d'une lexie est constitué d'au moins deux composantes : 's' = 's_1' ⊕ 's_2'. Le **pivot sémantique** est défini comme le 's_1' du sens 's' d'une lexie, si et seulement si, 's_2' est un prédicat dont 's_1' est l'argument.

Ces trois types de locutions sont traités de la même manière dans le cadre de notre implémentation, car celle-ci n'est pas basée sur une classification sémantique, mais bien sur une classification grammaticale des locutions. Dans la prochaine section, nous présentons notre corpus, qui est structuré en fonction de cette classification des locutions.

3.2 Données lexicales

3.2.1 Réseau lexical du français

Le RL-fr[7] (Polguère, 2009 ; 2014) est une **ressource lexicographique informatisée** en cours de développement au laboratoire d'Analyse et traitement informatique de la langue française (ATILF). Il s'agit du volet français d'un projet plus large, qui couvre aussi l'anglais (RL-en) et le russe (RL-ru). Ce projet est supervisé par Alain Polguère et fait suite aux travaux du *Dictionnaire explicatif et combinatoire* (Mel'čuk et coll., 1984 ; 1988 ; 1992 ; 1999), du *Lexique actif du français* (Mel'čuk et Polguère 2007) et du DiCo[8].

Le RL-fr vise à fournir une représentation du lexique utilisable en TAL, aussi bien pour générer du texte que pour l'interpréter ou le traduire. On peut consulter le RL-fr grâce à l'instrument de navigation Spiderlex (Gader et coll., 2014 ; Ollinger et coll., 2020 ; Ollinger et Polguère, 2020), qui permet une exploration dynamique du réseau.

Chaque lexie du réseau possède son propre **article lexicographique**, dans lequel figure une description qui comprend :

- des caractéristiques grammaticales ;
- des informations morphologiques (Gader et coll. 2014, notamment des mots-formes exprimant les formes fléchies des lexèmes) ;
- des informations sémantiques (identification des actants pour les lexies sémantiquement prédicatives, étiquette sémantique et définition lexicographique) ;
- des informations syntaxiques (patron de régime) ;
- des exemples ;
- des relations avec d'autres nœuds du réseau.

[7] https://lexical-systems.atilf.fr/
[8] http://olst.ling.umontreal.ca/dicouebe/index.php

Les **caractéristiques grammaticales** comprennent à leur tour :

- les marques d'usage :
    - langagier : rare, vieilli, argotique, anglicisme, etc. ;
    - stylistique : enfantin, familier, soutenu, etc. ;
    - rhétorique : affectueux, ironique, péjoratif, etc. ;
- la partie du discours ;
- la structure lexico-syntaxique ;
- le genre ;
- la flexion ;
- les autres caractéristiques fonctionnelles.

Les relations entre les nœuds du réseau comprennent des liens paradigmatiques et syntagmatiques. Ceux-ci sont modélisés au moyen de fonctions lexicales (Polguère, 2014b ; Mel'čuk 2007 ; Mel'čuk et Polguère, 2021), de la copolysémie, de définitions lexicographiques et de structures lexico-syntaxiques (Pausé 2014, 2017). Une structure lexico-syntaxique est une association entre un patron syntaxique linéarisé et des unités lexicales. La section suivante est consacrée à la description de ces patrons syntaxiques linéarisés.

3.2.2 Patrons linguistiques : classification des locutions du français

Les **patrons syntaxiques linéarisés** sont le fruit des travaux de Pausé (2014, 2017) sur les locutions. Celles-ci proviennent, d'une part, de bases de données textuelles telles que *Frantext*[9], *Est Républicain*, *Europress*[10] et, d'autre part, de recherches sur la toile faites à l'aide de FrWac[11] (Baroni et coll. 2009), de *Google Books* et d'autres outils (Pausé, 2017, p. 11).

Un patron syntaxique linéarisé est une suite de parties du discours qui représentent chaque constituant d'une locution. Par exemple, la locution ⌜ENTENDRE UNE MOUCHE VOLER⌝ s'est vu attribuer le patron linguistique `V Art NC V`. Les locutions peuvent être regroupées en fonction de leur patron syntaxique linéarisé ; on peut ainsi effectuer une classification grammaticale des locutions. Afin de simplifier la lecture, les parties du discours ont été étiquetées par leur abréviation dans la création des patrons. Les **parties du discours** répertoriées dans le RL-fr sont les suivantes :

[9] http://www.frantext.fr
[10] http://www.europresse.com/fr
[11] http://nl.ijs.si/noske/wacs.cgi/first_form

- verbe (`V`) ;
- verbe pronominal (`Vpro`) ;
- nom commun (`NC`) ;
- nom propre (`Npr`) ;
- numéral (`Num`) ;
- pronom impersonnel (`ProImpers`) ;
- pronom personnel (`ProPers`) ;
- pronom interrogatif (`ProInter`) ;
- pronom relatif (`ProRel`) ;
- adjectif (`Adj`) ;
- adverbe (`Adv`) ;
- interjection (`Interj`) ;
- article (`Art`) ;
- préposition (`Prép`) ;
- préposition partitive (`PrépPart`) ;
- conjonction (`Conj`) ;
- adjectif déterminatif (`AdjDét`) ;
- pronom déterminatif (`ProDét`).

Notons que les deux dernières parties du discours sont propres au RL-fr. Elles désignent un type de déterminant qui est soit un adjectif (`AdjDét`), soit un pronom (`ProDét`). De plus, le RL-fr utilise les parties du discours suivantes pour décrire les **unités lexicales complexes** :

- locution verbale (`LocV`) ;
- locution nominale (`LocN`) ;
- locution adverbiale (`LocAdv`) ;
- locution prépositionnelle (`LocPrép`) ;
- phrasème non connexe (`PhNC`) ;
- adverbe de négation (`AdvNég`).

Cette liste ne comprend que les locutions imbriquées répertoriées jusqu'à présent. Les **phrasèmes non connexes** ont été intégrés aux RL-fr à la suite des travaux de Pausé (2017). Il s'agit de phrasèmes comme *de le* [X] dans ⌜BROYER DU NOIR⌝, dont le patron `V PhNC NC` rend possible les reprises anaphoriques (comme dans le cas de « quand je broie du noir, j'en broie. »[12]). Cette nouvelle catégorie de phrasèmes comprend en principe les adverbes de négation ; ceux-ci se

[12] Exemple repris de Pausé (2017), qui l'a elle-même tiré de la toile (https://forum.hardware.fr/).

voient malgré tout attribuer leur propre partie du discours (AdvNég) afin de faciliter les recherches au sein de la ressource.

Le RL-fr fait également la distinction entre les verbes pronominaux et les formes pronominales. La banque de dépannage linguistique (BDL) de l'Office québécois de la langue française (OQLF) propose les termes « verbes essentiellement pronominaux » et « verbes occasionnellement pronominaux » [13]. Comme les constituants des **verbes** *essentiellement* **pronominaux** sont indissociables, ceux-ci sont décrits à l'aide de la partie du discours Vpro. Cependant, les verbes *occasionnellement* pronominaux (les **formes pronominales**) sont décomposables. Le clitique dans la forme pronominale peut servir à exprimer un sens réfléchi ('Maxime se sourit dans le miroir' = 'Maxime sourit à lui-même dans le miroir') ou un sens réciproque ('Maxime et Eve-Marie se sourient' = 'Maxime sourit à Eve-Marie et Eve-Marie sourit à Maxime'). Il peut aussi équivaloir à un passif ('Les appartements dans Rosemont se louent très cher' = 'Les appartements dans Rosemont sont loués très cher'). Ces verbes sont alors décrits par l'entremise de leurs constituants, soit un clitique (Clit) et un lexème verbal (V).

Par ailleurs, les parties du discours Clit et ProImpers se distinguent sur la base de la pronominalisation d'un complément. ProImpers est utilisé pour décrire les pronoms qui ont un référent identifiable, alors que Clit est utilisé pour les pronoms sans référent identifiable.

Afin de préciser leurs patrons, l'équipe du RL-fr a décidé d'ajouter des **marqueurs de fonctions syntaxiques**. Ceux-ci permettent de différencier des suites de parties du discours identiques, mais formant des arbres syntaxiques différents. Le tableau 3 en présente quelques exemples illustrés par les figures 28, 29 et 30.

Locutions	Parties du discours	Patron linguistique
⌜PÉDALER DANS LE YAOURT⌝	V Prép Art NC	V Prép. Circ Art NC
⌜ALLER AUX FRAISES⌝		V Prép. Loc Art NC
⌜BATTRE DE L'AILE⌝		V Prép. Obl Art NC

Tableau 3 – Exemples de patrons différenciés par l'ajout de marqueurs fonctionnels (Pausé, 2017)

[13] http://bdl.oqlf.gouv.qc.ca/bdl/ (consulté le 13/08/2021).

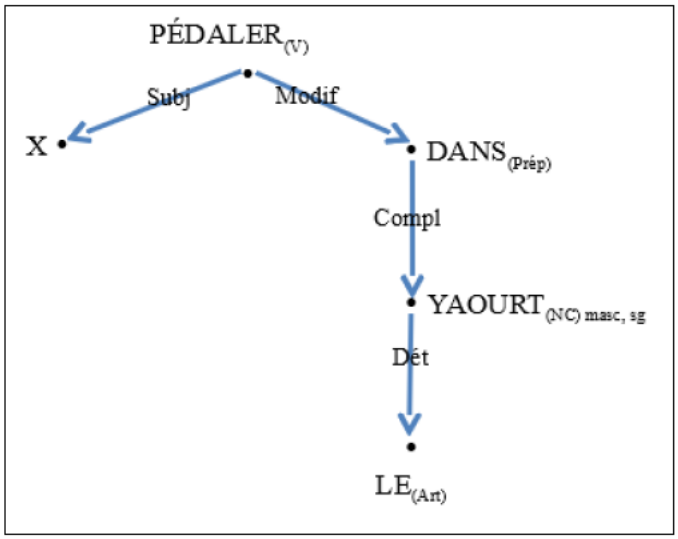


Figure 28 – Structure syntaxique de surface de *X pédale dans le yaourt*. (Pausé, 2017, p. 138)

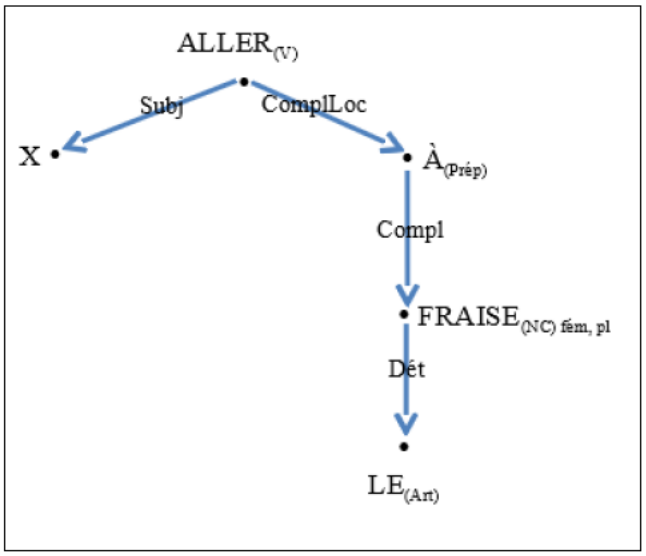


Figure 29 – Structure syntaxique de surface de *X va aux fraises.* (Pausé, 2017, p. 138)

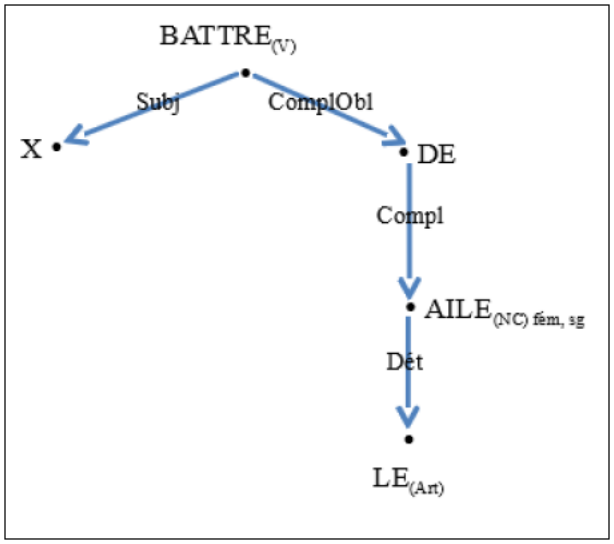


Figure 30 – Structure syntaxique de surface de *X bat de l'aile.* (Pausé, 2017, p. 139)

Voici la liste des **marqueurs fonctionnels** utilisés[14] :

- accusatif (`.Acc`) ;
- actanciel (`.Act`) ;
- apposition (`.Appos`) ;
- attribut de l'objet (`.AttrObj`) ;
- attribut du prédicat (`.AttrPréd`) ;
- attribut du sujet (`.AttrSuj`) ;
- circonstant (`.Circ`) ;
- datif (`.Dat`) ;
- génitif (`.Gén`) ;
- locatif (`.Loc`) ;
- oblique (`.Obl`).

De plus, comme une locution contrôlant des **positions actancielles** n'a pas la même RSyntS qu'une locution qui n'en contrôle pas, celles-ci ont été ajoutées aux patrons. Par exemple, la locution ⌜CASSER LA CROÛTE⌝ s'utilise simplement avec un sujet « *X casse la croûte*. », alors que la locution ⌜CLOUER LE BEC⌝ nécessite l'ajout d'au moins un complément « *X cloue le bec à Y avec Z* » (Pausé, 2017). Les patrons de certaines locutions verbales, prépositionnelles et phrastiques ont ainsi été précisés à l'aide de parenthèses. Le tableau 4 présente quelques exemples de cette désambigüisation. Par exemple, le patron de la locution ⌜LEVER LA MAIN⌝ est précisé grâce à l'ajout

[14] Pausé (2017) précise que ces marqueurs sont à retravailler, puisqu'ils ont été introduits les uns à la suite des autres.

d'une position actancielle (`$2`)[15] qui doit être remplie par une préposition (`Prép`) et celle-ci est liée à la locution par une relation oblique `(.Obl)`. Seuls les patrons ambigus sont dotés de précision par rapport à leurs actants.

Locutions	Parties du discours	Patron linguistique
⌜CASSER LA CROÛTE⌝	V Art NC	V Art NC
⌜CLOUER LE BEC⌝		V Art NC (Prép. Dat_$2)
⌜LEVER LA MAIN⌝		V Art NC (Prép. Obl_$2)
⌜EN DANSEUSE⌝	Prép NC	Prép NC
⌜EN ROUTE⌝		Prép NC (Prép. Obl_$2)
⌜À DOS⌝		Prép NC (Prép. Gén_$2)

Tableau 4 – Exemples de patrons différenciés par l'ajout de positions actancielles (Pausé, 2017)

Pausé (2017) attribue une partie du discours à ces patrons en se basant sur **trois critères** : la sémantique (par exemple, tous les verbes sont des prédicats sémantiques), la syntaxe (par exemple, en français, un nom peut généralement occuper la place de sujet ou de complément d'un verbe) et la morphologie (par exemple, en français, prototypiquement, un nom varie en nombre et a un genre fixe). Cela lui permet d'établir une **classification grammaticale des locutions**, qui se répartissent selon les 12 parties du discours suivantes :

- locution verbale ;
- locution nominale ;
- locution nominale nom propre ;
- locution numérale ;
- locution pronominale ;
- locution adjectivale ;
- locution adverbiale ;
- locution interjective ;
- locution propositionnelle ;
- locution phrastique ;
- locution prépositionnelle ;
- locution conjonctive.

[15] Les positions actancielles X, Y, Z sont représentés par les symboles $1, $2 et $3 dans les patrons et sont séparés des marqueurs fonctionnels par des tirets bas (_).

Cette typologie se fonde en grande partie sur la position que la locution occupe dans le texte, c'est-à-dire sa capacité à se combiner à d'autres lexies en tant que dépendant syntaxique (sa **valence passive**). Cependant, les locutions nominales nom propre[16], numérales, interjectives et propositionnelles dépendent également du signifiant du syntagme qu'elles forment (Iordanskaja et Mel'čuk 2017).

Le tableau 5 présente des exemples de patrons syntaxiques linéarisés pour chaque partie du discours qui y est décrite.

Patrons syntaxiques linéarisés	Exemples	Partie du discours
V Art NC	⌜LEVER LE MASQUE⌝	locution verbale
	⌜REMONTER LA PENTE⌝	
NC Prép NC	⌜SIGNE DE VIE⌝	locution nominale
	⌜JEU DE MOTS⌝	
Num Prép Num	⌜UN À UN⌝	locution numérale
ProImpers Conj ProImpers	⌜ICI ET LÀ⌝	locution pronominale
Adv Adj	⌜POLITIQUEMENT CORRECT⌝	locution adjectivale
	⌜TOUT CRU⌝	
Adv Conj Adv	⌜BEL ET BIEN⌝	locution adverbiale
	⌜PLUS OU MOINS⌝	
Interj Prép Art NC !	⌜MORT AUX VACHES !⌝	locution interjective
ProRel Clit V	⌜QUI SE RESPECTE⌝	locution propositionnelle
	⌜Ce n'est pas la mer à boire.⌝	locution phrastique
Prép NC	⌜PAR AILLEURS⌝	locution prépositionnelle
	⌜SANS DOUTE⌝	
Conj Art NC	⌜COMME L'ÉCLAIR⌝	locution conjonctive
	⌜COMME DES MOUCHES⌝	

Tableau 5 – Exemples de patrons syntaxiques linéarisés

Dans le cadre du présent mémoire, nous référons aux patrons syntaxiques linéarisés en tant que ***patrons linguistiques*** afin de les distinguer de nos *patrons génériques*. Ces derniers seront présentés dans le chapitre suivant, qui porte sur notre implémentation.

[16] Pausé (2017, p.52) ajoute à ce sujet : « Cette appellation, certes discutable, a vocation à dénommer les locutions dont la tête de syntagme est un nom propre ».

Chapitre 4 Implémentation

Le présent chapitre décrit notre implémentation de la **lexicalisation par patron** dans GenDR (voir chapitre 2). Cette implémentation consiste en la création d'une grammaire et d'un dictionnaire. Nous commençons par présenter la grammaire, qui est constituée de règles de transduction. Celles-ci découlent de la création de patrons génériques et de leur mise en correspondance avec les patrons linguistiques présentés au chapitre 3. Ces patrons génériques prennent la forme d'arbres à trous, qui sont complétés à l'aide d'information lexicale propre à la langue traitée. Nous présentons, par la suite, la création du dictionnaire lexical qui contient cette information.

4.1 Grammaire

L'écriture de la grammaire nécessite la création de nombreuses **règles de transduction**. Celles-ci doivent assurer la correspondance entre les RSyntP et les RSyntS des locutions. Dans le cadre de la TST, la locution ne forme qu'une seule unité en syntaxe profonde, alors qu'elle correspond à plusieurs unités en syntaxe de surface. Nous proposons donc des patrons génériques qui décrivent la RSyntS des locutions.

Dans la présente section, nous commençons par décrire les patrons génériques ; nous expliquons comment ceux-ci sont calculés et mis en correspondance avec les patrons linguistiques. Puis, nous enchaînons par la présentation des règles de transduction et de leur génération automatique.

4.1.1 Patrons génériques

Les patrons génériques sont basés sur la structure syntaxique de chaque locution en **RSyntS**. Ils constituent donc toutes les configurations possibles pour une **locution à *n* lexèmes** de surface. Autrement dit, nous calculons toutes les représentations syntaxiques de surface possibles pour un **arbre à *n* nœuds**.

Prenons un exemple concret. La figure 31 représente tous les patrons génériques théoriquement possibles pour une locution à quatre nœuds (voir l'Annexe 1 pour la liste complète des patrons de deux à six nœuds).

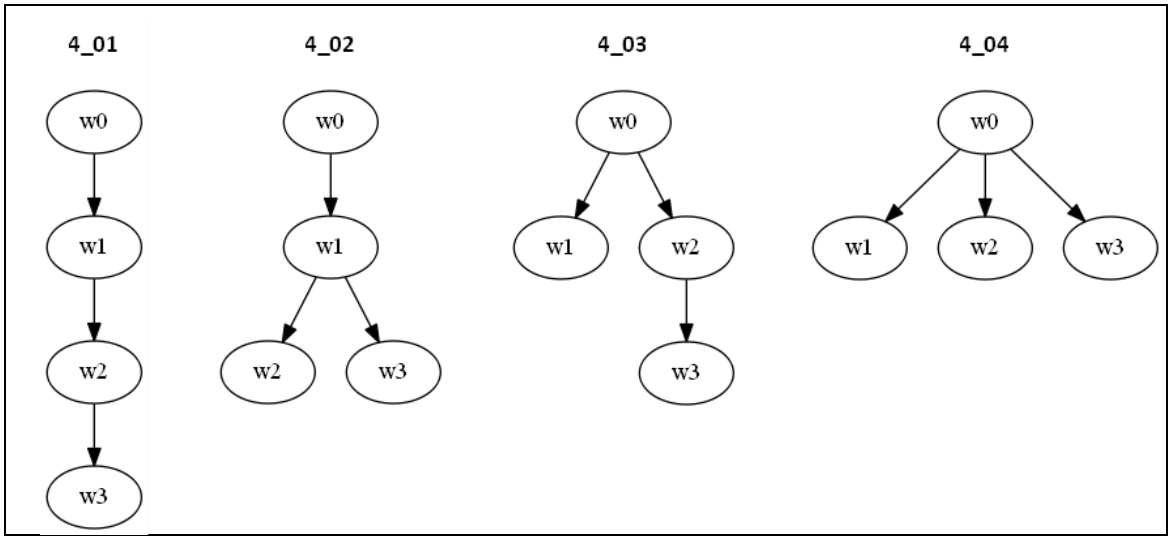


Figure 31 – Patrons génériques possibles pour un arbre à quatre nœuds

Chacun de ces arbres est unique et **non linéaire** ; l'ordre d'apparition des lexèmes n'importe pas. Nous nous intéressons strictement à la relation de gouvernance entre les nœuds de l'arbre. Ainsi, les deux patrons illustrés à la figure 32 sont équivalents.

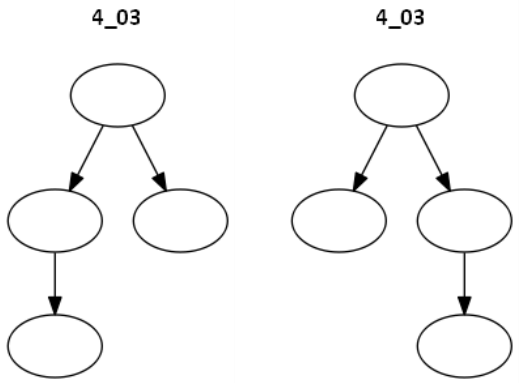


Figure 32 – Exemple de patrons génériques non linéaires identiques

Dans les deux prochaines sous-sections, nous abordons d'abord la question du calcul des patrons génériques, puis nous expliquons comment ces patrons sont complétés à l'aide d'information lexicale.

4.1.1.1 Calcul des patrons génériques

Afin de déterminer le nombre de patrons génériques possibles, nous nous sommes appuyée sur la **partition des entiers**. La théorie des nombres décrit une **partition** comme une décomposition

de l'entier en une somme d'entiers strictement positifs[17] (nommés *parties*) (Andrews, 1998). La figure 33 en est un exemple.

3

2 + 1

1 + 1 + 1

Figure 33 – Trois partitions de l'entier 3

Nous avons trouvé que les différentes partitions d'un entier correspondent aux différentes organisations possibles des dépendants de la racine. Ainsi, la partition des entiers nous permet d'établir le nombre d'arbres possibles pour un *n* donné.

Prenons l'exemple d'un arbre à trois nœuds. Cet arbre est constitué d'une racine et de deux dépendants. Deux partitions sont possibles. Dans le premier cas, la racine gouverne un dépendant ayant lui-même un dépendant (donc un sous-arbre à deux nœuds) ; dans le second cas, la racine gouverne directement deux nœuds (donc deux sous-arbres à un seul nœud). Le tableau 6 permet de visualiser les partitions. La racine est notée `r` et elle correspond au nœud `w0` de l'arbre. Les dépendants de l'arbre sont regroupés selon les partitions de l'entier 2, c'est-à-dire `2` ou `1 + 1`.

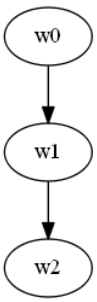
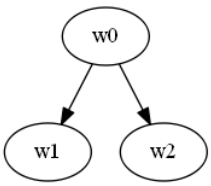
Parallèle avec partitions	`r+2`	`r+1+1`
Patrons génériques	**3_01**  	**3_02** 

Tableau 6 – Patrons génériques `3_01`, `3_02` et leurs partitions

[17] Le nombre de partitions pour un entier donné est une séquence connue dans le monde mathématique : 1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42, 56, 77, 101, 135, 176, 231… (https://oeis.org/A000041).

Selon la même logique, un arbre à quatre nœuds voit ses dépendants organisés ainsi :

a) une racine reliée à un sous-arbre de trois nœuds (`r+3`)
b) une racine reliée à deux sous-arbres ayant un et deux nœuds, respectivement (`r+2+1`)
c) une racine reliée à trois sous-arbres d'un seul nœud (`r+1+1+1`)

Grâce à la partition de l'entier 3 (voir figure 33), nous pouvons déduire que quatre patrons génériques différents sont possibles. D'une part, il y a un patron pour chacun des sous-arbres décrits en b) et c) et, d'autre part, deux patrons pour le sous-arbre en a). En effet, ce dernier contient lui-même deux configurations de sous-arbres (voir tableau 6). Cela signifie que chaque partie de partition est associée à un nombre d'arbres possible. Grâce à cette récursivité, nous arrivons à énumérer systématiquement tous les patrons génériques possibles pour un arbre à *n* nœuds. Le tableau 7 offre une récapitulation des notions mentionnées ci-dessus.

Nombre de nœuds	**Nombre de dépendants**	**Partitions des dépendants**[18]	**Nombre d'arbres par partition**	**Nombre d'arbres total**
2	1	1	1	1
3	2	2	1	2
		1 + 1	1	
4	3	3	2	4
		2 + 1	1	
		1 + 1 + 1	1	
5	4	4	4	9
		3 + 1	2	
		2 + 2	1	
		2 + 1 + 1	1	
		1 + 1 + 1 + 1	1	
6	5	5	9	20
		4 + 1	4	
		3 + 2	2	
		3 + 1 + 1	2	
		2 + 2 + 1	1	
		2 + 1 + 1 + 1	1	
		1 + 1 + 1 + 1 + 1	1	

Tableau 7 – Nombre d'arbres à *n* nœuds différents

[18] La partition est calculée à partir du nombre total de dépendants de la racine de l'arbre.

Il répertorie le nombre d'arbres possibles par nombre de nœuds, les partitions possibles selon le nombre de dépendants de la racine, ainsi que le nombre d'arbres pour une partition donnée. Chaque ligne de la colonne des partitions des dépendants correspond à une partition. Les partitions sont affichées en ordre décroissant. De plus, chaque ligne de la colonne suivante correspond à la somme du nombre d'arbres pour chaque partition donnée. En répertoriant les informations ainsi, nous pouvons facilement repérer les oublis ou les doublons. Dès qu'un chiffre de la partition ne peut plus se décomposer en un nombre entier positif, nous passons à la partition suivante. Par doublon, nous entendons deux partitions équivalentes présentées dans un ordre différent tel que 2 +1 et 1 +2 (voir figure 32).

Afin de valider nos calculs, nous avons vérifié la partition d'entiers plus grands. Prenons à titre d'exemple un arbre à sept nœuds, les informations pertinentes à sa création sont répertoriées dans le tableau 8.

Nombre de nœuds	**Nombre de dépendants**	**Partitions des dépendants**[19]	**Nombre d'arbres par partition**	**Nombre d'arbres total**
7	6	6	20	48
		5 + 1	9	
		4 + 2	4	
		4 + 1 + 1	4	
		3 + 3	3	
		3 + 2 + 1	2	
		3 + 1 + 1 + 1	2	
		2 + 2 + 2	1	
		2 + 2 + 1 + 1	1	
		2 + 1 + 1 + 1 + 1	1	
		1 + 1 + 1 + 1 + 1 + 1	1	

Tableau 8 – Nombre d'arbres à 7 nœuds

Portons une attention particulière à la partition 3 + 3, où une même *partie* revient plus d'une fois au sein de la même partition. Nous avons vu précédemment qu'un *n* de 3 permet de créer deux arbres distincts. Si nous remplaçons les *parties* (3) de la partition par le nombre d'arbres qu'elles représentent, cela donne 2 et 2. Jusqu'à présent, une simple multiplication nous permettait d'obtenir le nombre d'arbres total pour un *n* donné. Cependant, ici, un total de quatre arbres

[19] Idem.

serait erroné. Pour le démontrer, nous pouvons considérer les deux arbres possibles comme des variables, nommons-les `A` et `B`. Ces variables peuvent se combiner ainsi : `AA`, `AB`, `BA` et `BB`. Comme nos patrons sont non linéaires, l'ordre n'est pas important. Cela signifie que la combinaison `AB` est équivalente à `BA` ; il s'agit donc de doublons. C'est ce qu'illustre la figure 34.

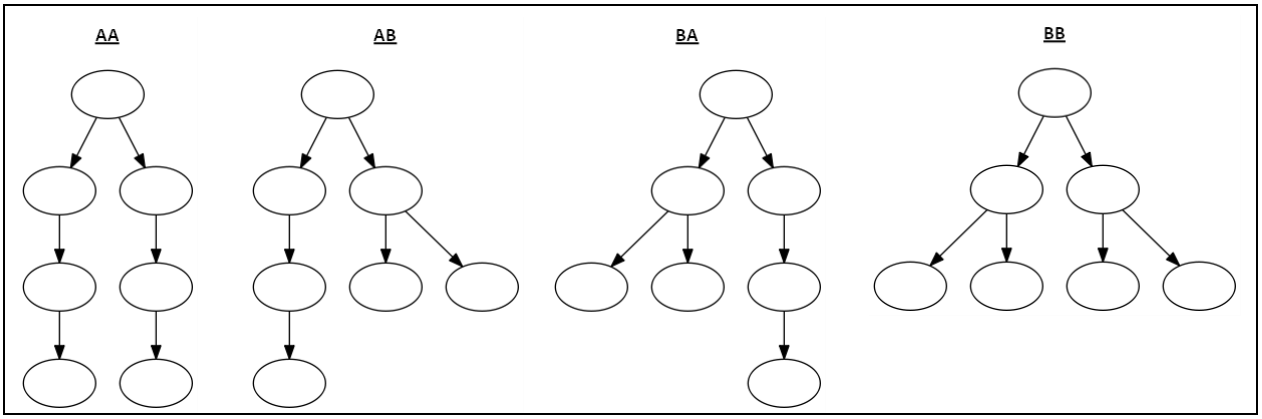


Figure 34 – Exemple démontrant la non-linéarité des patrons génériques

À la lumière de ce qui précède, nous avons établi une **équation** pour effectuer le calcul du nombre d'arbres différents (T, pour *trees*) en fonction du nombre de nœuds (N) dans la RSyntS :

$$T(N) = \sum_{p \in P(N-1)} \prod_{n,k \in p} \frac{(T(n)+k-1)!}{(T(n)-1)!k!}$$

Le **calcul** fonctionne de façon récursive et est limité aux nombres entiers positifs. Il est basé sur le nombre d'occurrences (`k`) d'un nombre entier (`n`) dans chaque partition (`p`). L'équation comporte le coefficient binomial `t (n, k)`, qui correspond à la dernière section de l'équation. Ce coefficient fait appel aux différentes combinaisons et permet d'éviter les doublons.

$$\mathrm{t}(n,k) = ((T(n)+k-1)!/(T(n)-1)!\,k!$$

On calcule le produit des coefficients `t` pour chaque `n` qui apparait un nombre `k` de fois dans la partition `p`. Puis, on fait la somme de ces produits pour chaque `p` de l'ensemble `P` des partitions selon le nombre de dépendants de la racine (`N-1`).

Après cette présentation du calcul des patrons génériques, voyons maintenant comment ceux-ci peuvent être utilisés avec diverses langues.

4.1.1.2 Mise en correspondance des patrons linguistiques et génériques

Comme les patrons génériques sont essentiellement des **arbres à trous**, il faut les remplir à l'aide d'informations lexicales et grammaticales propres à chaque langue. Pour ce faire, nous associons à chacun des patrons linguistiques identifiés pour une langue donnée un de nos patrons génériques. Dans le cadre du présent projet, nous nous sommes limitée aux patrons des locutions du RL-fr (voir section 3.2.2). La même méthode peut cependant être utilisée avec d'autres ressources lexicales ou avec des patrons linguistiques détectés dans un corpus préalablement étiqueté.

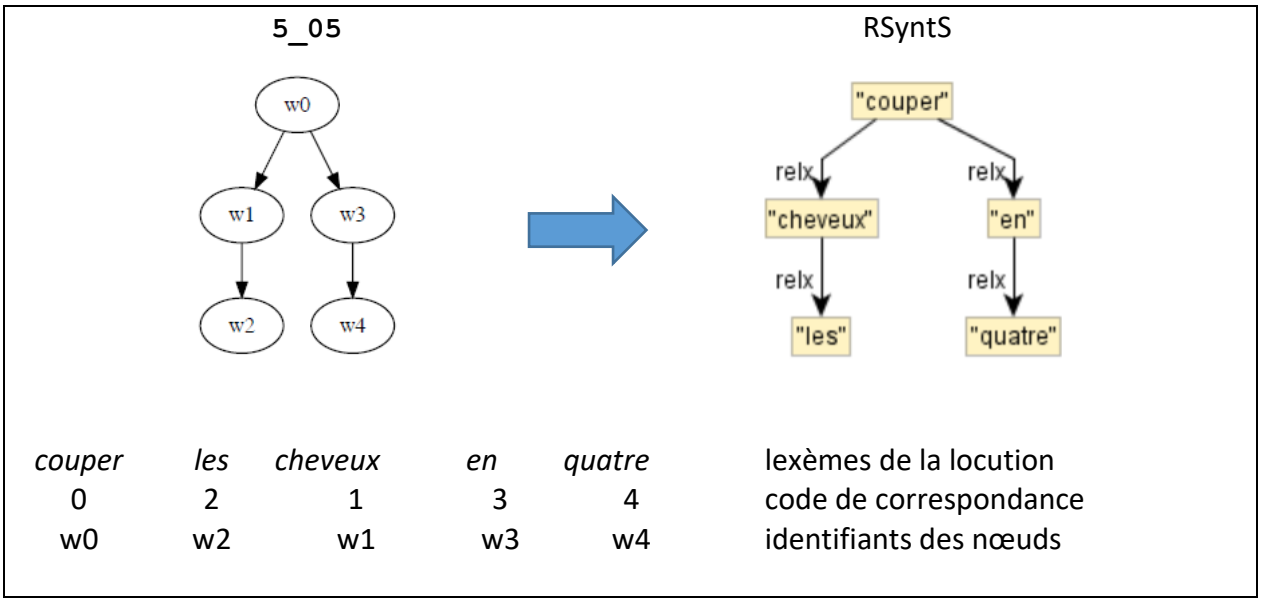
La **mise en correspondance des patrons** nécessite de sélectionner le patron générique associé à un patron de locution, c'est-à-dire de créer sa RSyntS. Cette étape demande donc une bonne connaissance du formalisme utilisé. Bien que l'automatisation de cette mise en correspondance soit envisageable, il est plus fiable de l'effectuer manuellement. Le tableau 9 présente un aperçu organisé de la mise en correspondance des patrons linguistiques et génériques.

Patron linguistique	**Patron gén.**	**Corr.**	**Exemple**
Prép V	2	01	à vomir
V Prép NC	3_01	012	changer de couleur
Art NC Adj	3_02	102	les jambes molles
Prép NC Prép. Loc NC	4_01	0123	de bouche à oreille
V Art NC Adj	4_02	0213	enfoncer une porte ouverte
Prép Art Adj NC	4_02	0231	dans de beaux draps
V Art NC Prép Num	**5_05**	**02134**	**couper les cheveux en quatre**
V Art NC Prép. Loc Art NC (Prép_$2)	6_10	021354	mettre la puce à l'oreille
Clit V Prép Adj Art NC	6_13	102453	en voir de toutes les couleurs

Tableau 9 – Exemple de correspondance entre les patrons linguistiques et génériques

À titre explicatif, prenons la locution ⌜COUPER LES CHEVEUX EN QUATRE⌝ ; elle comprend cinq nœuds. Nous lui attribuons le patron générique `5_05` et le **code de correspondance** *02134*. La figure 35 ci-dessous illustre la mise en correspondance entre les lexèmes de cette locution et les nœuds de l'arbre (la RSyntS). Le code de correspondance indique à notre réalisateur quel lexème attribuer à chacun des nœuds de l'arbre lors de la lexicalisation. Chacun des nœuds est identifié par une variable (*w0*, *w1*, *w2*, etc.), qui réfère à une position précise dans l'arbre. Cette dernière information est cruciale dans l'écriture des règles et des dictionnaires. Nous référons également

aux lexèmes de la locution selon leur ordre d'apparition dans la forme de citation, de gauche à droite. En somme, le code de correspondance prend cette numérotation ordinale et la réorganise selon la position des lexèmes dans l'arbre. Par exemple, dans ce cas-ci, *02134* signifie que le nœud *w0* est « couper », le nœud *w1* est « cheveux », le nœud *w2* est « les » et ainsi de suite. Le code *02134* est donc équivalent à la série *w0 w2 w1 w3 w4*.



couper	*les*	*cheveux*	*en*	*quatre*	lexèmes de la locution
0	2	1	3	4	code de correspondance
w0	w2	w1	w3	w4	identifiants des nœuds

Figure 35 – Correspondance des patrons pour la locution ⌜COUPER LES CHEVEUX EN QUATRE⌝[2021]

En suivant cette procédure, nous avons déterminé les **codes de correspondance** pour chaque paire de patrons. Dans le cadre du présent mémoire, nous nous sommes limitée aux locutions de deux à six nœuds. Cela permet de gérer un nombre de patrons génériques restreint, mais suffisant pour décrire toutes les locutions du français ou d'autres langues. Nous avons prise cette décision en considérant plusieurs facteurs que nous détaillons au chapitre 5 (voir section 5.1.3). La section suivante traite des règles de transduction ; nous verrons comment celles-ci encodent les patrons génériques pour créer les RSyntS appropriées.

[20] La convention de Python veut que le premier élément soit indiqué par un « 0 ».
[21] Les relations de dépendance ne sont pas précisées dans le cadre de notre traitement ; elles sont donc toutes nommées « relx » (voir section 4.2.1).

4.1.2 Règles de transduction

Comme nous l'avons mentionné précédemment, les règles de transduction assurent le passage entre les différents niveaux de représentations de la TST (voir section 2.2.1 pour plus de détails). Notre traitement des locutions fait uniquement appel aux deux **niveaux de représentations syntaxiques**. Cela nécessite la création de règles qui font correspondre un nœud unique en RSyntP à plusieurs nœuds en RSyntS Cela produit donc les représentations des locutions en syntaxe de surface.

De manière générale, la structure de ces règles comporte quatre parties distinctes, illustrées à la figure 36.

Figure 36 – Règle de lexicalisation `lex_idiom_2`

La **partie du haut** énonce le titre de la règle. On y précise le module dans lequel la règle s'insère (`DSynt⇔SSynt`), le nom de la règle (`lex_idiom_2`) et le groupe de règles auquel elle appartient (`lex_idiom`). Les règles appartenant à ce groupe servent principalement à décomposer un nœud en fonction de l'organisation arborescente du patron générique lui étant associé.

La **partie gauche** (`leftside`) présente l'entrée. Dans le cas de `lex_idiom_2`, il s'agit d'un nœud (`?X1`) en RSyntP. La **partie droite** (`rightside`), quant à elle, présente le produit. Ainsi, la racine (`?W0`) est d'abord identifiée, puis toute l'information concernant ce nœud se retrouve entre accolades. La règle sélectionnée comprend deux nœuds ; il y a conséquemment deux paires d'accolades[22]. Dans la première paire, il est d'abord spécifié que le nouveau nœud correspond à celui de la partie gauche. Puis, la règle décrit le chemin à emprunter pour récupérer sa lexicalisation profonde (`dlex`) et superficielle (`slex`), ainsi que sa partie du discours profonde (`dpos`) et superficielle (`spos`). Par exemple, pour récupérer la `slex`, la règle va chercher la `dlex` du nœud `?X1` dans le dictionnaire lexical et récupérer la valeur associée au `w0` qui se trouve dans le trait `idiom`. Dans la figure 39, par exemple, la valeur du nœud `w0` est `"couper"`. Ensuite, la règle précise qu'une relation est ajoutée entre les nœuds `?W0` et `?W1`. Par la suite, les valeurs du deuxième nœud `?W1` sont recherchées dans le dictionnaire selon la même logique.

```
"couper les cheveux en quatre" : "V Art NC Prép Num" {
idiom = {
    w0 = "couper"
    w1 = "cheveux"
    w2 = "les"
    w3 = "en"
    w4 = "quatre"
  }
}
```

Figure 37 – Entrée de dictionnaire pour la locution ⌜COUPER LES CHEVEUX EN QUATRE⌝

Finalement, la **partie du bas** contient les conditions qui doivent être remplies pour que la règle s'applique. Dans l'exemple, la règle demande comme condition d'application que l'entrée de dictionnaire contienne le trait `idiom` de type `idiom_2`. Dans la prochaine section, nous détaillons comment ces règles ont été générées automatiquement grâce à un script.

[22] Nous utilisons l'indentation pour refléter la structure du patron générique dans le code.

4.1.3 Génération automatique des règles de transduction

Les règles de transduction ont été générées à l'aide d'un script Python. Dans la présente section, nous présentons les fonctions principales du script *partition.py*, qui sert à générer les règles de transduction pour la **lexicalisation par patron**. Comme le nom du script l'indique, la génération de ces règles est basée sur la partition des entiers (voir section 4.1.1.1).

Tout d'abord, le script se base sur un intervalle de nombres entiers pour produire toutes les règles de transduction associées à cet intervalle. Celles-ci sont ensuite encodées dans le fichier *idioms.rls*. La figure 38 en présente un extrait pour la règle `lex_idiom_2`, présentée précédemment à la figure 36.

```
/* (()) */
DSynt<=>SSynt lex_idiom_2 : lex_idiom
[
  leftside = [
l :?Xl
  ]
  conditions = [
lexicon::(?Xl.dlex). idiom.type=idiom_2;
  ]
  rightside = [
?W0{
  <=>? Xl
  dlex=?Xl.dlex
  slex=lexicon::(?Xl.dlex). idiom.w0
  dpos=lexicon::(?Xl.dlex). dpos
  spos=lexicon::(?Xl.dlex).spos
  ssynt=OK
  lexicon ::(?Xl.dlex).idiom.r1-> ?W1 {
    slex=lexicon::(?Xl.dlex).idiom.w1
    dpos=lexicon::(lexicon::(?Xl.dlex).idiom.w1). dpos
    spos=lexicon::(lexicon::(?Xl.dlex).idiom.w1).spos
    ssynt=OK
  }
}

  ]
  correspondence = [

  ]
]
…
```

Figure 38 – Extrait du fichier `idioms.rls` pour la règle `lex_idiom_2`

L'écriture de ce fichier est possible grâce à l'utilisation de plusieurs **fonctions**, qui dépendent les unes des autres. Pour commencer, la fonction principale de ce script, nommée `rule()`, crée le squelette de la règle de transduction (voir figure 39). Elle prend en entrée différentes variables et retourne le squelette du patron générique associé à chacune d'entre elles.

```
def rule (tree,n, i) :
    r= """/* {c} */
DSynt<=>SSynt lex_idiom_{n}_{iz} : lex_idiom
[
  leftside = [
l :?Xl
  ]
  conditions = [
lexicon::(?Xl.dlex).idiom.type=idiom_{n}_{iz};
  ]
  rightside = [
{rs}
  ]
  correspondence = [

  ]
]\n""". format (c=re.sub (r',\)', ')', str (tree)), n=n,
iz=str(i).zfill(2), rs=rightside(tree)[0])
    return r
```

Figure 39 – Fonction `rule()` extraite de `partition.py`

On note que cette fonction dépend deux fonctions sous-jacentes. La première, nommée `rightside()`, produit la partie droite de la règle en fonction du patron générique de l'arbre qui lui est associé. Elle est représentée à la figure 40.

```
def rightside(tree, i=0, indent=0):
    mate = '?W' + str(i) + '{\n'
    m = i
    if m == 0:
        mate += '  <=> ?Xl\n'
        mate += '  dlex=?Xl.dlex\n'
        mate += '  slex=lexicon::(?Xl.dlex). idiom.w0\n'
        mate += '  dpos=lexicon::(?Xl.dlex). dpos\n'
        mate += '  spos=lexicon::(?Xl.dlex).spos\n'
        mate += '  ssynt=OK\n'
    else:
        mate += indent*'  ' +

'  slex=lexicon::(?Xl.dlex).idiom.w'+str(m)+'\n'
        mate += indent*'  ' +
                '  dpos=lexicon::(lexicon::(?Xl.dlex).idiom.w'+
                str(m)+').dpos\n'
        mate += indent*'  ' +
                '  spos=lexicon::(lexicon::(?Xl.dlex).idiom.w'+
                str(m)+').spos\n'
        mate += indent*'  ' + '  ssynt=OK\n'
    i += 1
    for dep in tree:
        mate += indent*'  ' + '  lexicon::(?Xl.dlex).idiom.r'+
                str(m+1)+'-> '
        dep_mate, dep_i = rightside(dep, m+1, indent+1)
        mate += dep_mate
        m = max(m, dep_i)
    mate += indent*'  ' + '}\n'
    return mate, m
```

Figure 40 – Fonction `rightside()` extraite de `partition.py`

La deuxième, nommée `tree()`, permet d'énumérer tous les arbres pour un *n* donné. Comme nous l'avons mentionné précédemment, le calcul des patrons génériques se base sur la récursion (voir section 4.1.1.1). Celle-ci s'étend ainsi à leur génération automatique, qui demande la création de nombreuses boucles. La fonction `tree()` dépend elle-même de plusieurs fonctions sous-jacentes nommées `partition()`, `flatten_list()` et `flatten_tuple()`.

La fonction `partition()` prend en entrée un nombre entier `n` et produit la partition de $n-1$. Le résultat est organisé sous forme de tuples pour faciliter l'illustration des différentes partitions. Un tuple est une séquence ordonnée de longueur fixe, présentée entre parenthèses.

```
def partition(n):
    if n < 1:
        return []
    else:
        answer = set()
        answer.add((n, ))
        for x in range(1, n):
            for y in partition(n – x):
                answer.add(tuple(sorted((x, ) + y)))
        return sorted(answer)
```

Figure 41 – Fonction `partition()` extraite de `partition.py`

Entrée : `4`

Sortie: `[(1,1,1,1), (1,1,2), (1,3), (2,2), (4)]`

Ensuite, la fonction `flatten_tuple()`, présentée à la figure 42, aplatit les tuples en développant les imbrications. Cela fait disparaître les doublons dans les partitions. Par exemple, cette fonction permet de développer la partition `ABC` en la liste `AB`, `AC` et `BC`.

```
def flatten_tuple(compressed):
    return list({tuple(sorted (p)) for p in
itertools.product(*compressed)})
```

Figure 42 – Fonction `flatten_tuple()` extraite de `partition.py`

Puis, la fonction `flatten_list()` écrase la liste en faisant disparaître les imbrications (voir figure 43).

```
def flatten_list(compressed):
    flat = []
    for item in compressed:
        if type(item) == list:
            flat = flat + flatten_list(item)
        else:
            flat.append(item)
    return flat
```

Figure 43 – Fonction `flatten_list()` extraite de `partition.py`

Enfin, la fonction `tree()` prend en entrée un nombre entier et produit une liste aplatie de tuples, comme le montre la figure 44. Les quatre fonctions décrites précédemment permettent donc de présenter l'architecture des patrons génériques sous forme de tuples imbriqués (voir sortie).

```
def trees(n):
    if n < 1:
        return []
    elif n == 1:
        return [()]
    return sorted(flatten_list([flatten_tuple(tuple(trees(pp))
for pp in p)  for p in partition(n – 1)]), reverse=True)
```

Figure 44 – Fonction `trees()` extraite de `partition.py`

Entrée : 4

Sortie : `[(((()),),),), (((), ()),), ((), ((),)), ((), (), ())]`

La sortie ci-dessus correspond à une séquence d'arbres, soit les patrons génériques **`4_01`** `(((()),),),)` ; **`4_02`** `(((), ()),)` ; **`4_03`** `((), ((),))` ; et **`4_04`** `((), (), ())`. La figure 45 présente un exemple de correspondance entre les tuples et les arbres pour le patron générique `4_02`. Chaque paire de parenthèses représente un nœud de l'arbre. Les dépendants d'un nœud sont insérés à l'intérieur de la paire de parenthèses. Ainsi, plus un nœud est imbriqué, plus il y a de nœuds qui le gouvernent.

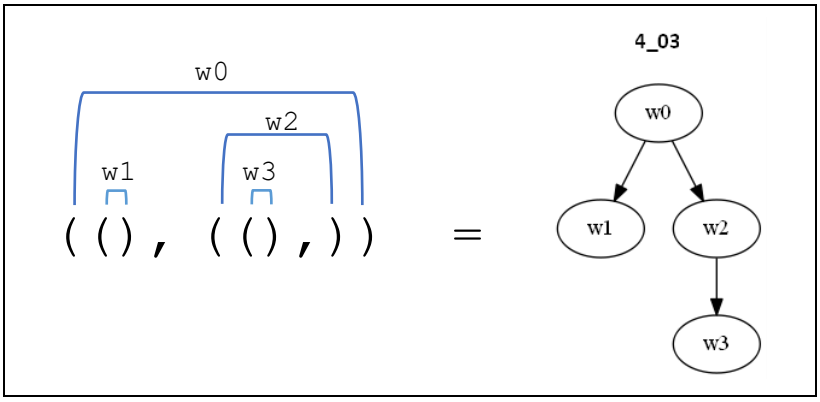


Figure 45 – Exemple de correspondance entre les tuples et les arbres pour le patron `4_03`

Nous avons généré cette séquence pour que les patrons les plus fréquents apparaissent en premier. En effet, en analysant les patrons génériques, nous avons remarqué que les arbres profonds étaient plus fréquents que les arbres plats (voir figure 46). Nous les appelons ainsi en raison de leur aspect visuel : les arbres profonds sont composés d'une chaîne de dépendants, alors que les arbres plats ont plusieurs dépendants directement reliés à la racine. Le tableau 10 compare la fréquence des différents types de patrons génériques dans les locutions du RL-fr.

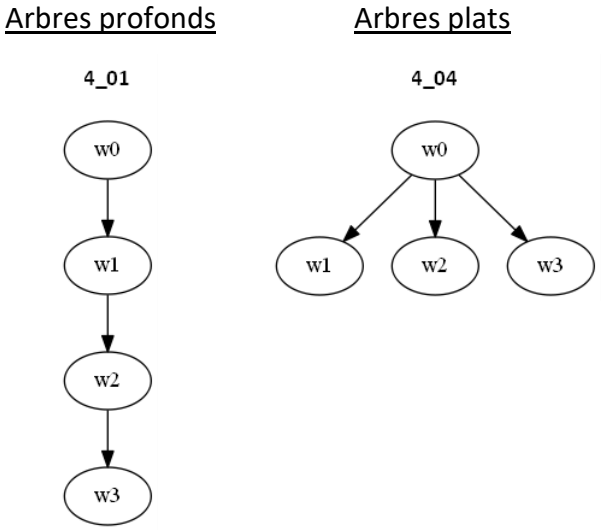


Figure 46 – Exemples d'arbre profond et d'arbre plat

Patrons à arbre profond		Patrons à arbre plat	
3_01 :	1141	**3_02 :**	44
4_01 :	325	**4_04 :**	7
5_01 :	20	**5_09 :**	1

Tableau 10 – Exemple de fréquence des patrons génériques[23]

En conclusion, les règles de transduction de graphes permettent au réalisateur de générer automatiquement les RSyntS[24] des locutions à partir de leur RSyntP. La prochaine section présente le dictionnaire lexical qui contient les informations nécessaires à la lexicalisation des RSyntS.

[23] Données exportées du RL-fr en date du 18/06/2019.

[24] Les relations de dépendance ne sont pas actuellement encodées dans le RL-fr, ce qui explique leur omission dans nos RSyntS. Cependant, notre traitement peut facilement être adapté afin d'intégrer ces informations lorsqu'elles seront disponibles.

4.2 Dictionnaire

Dans GenDR, chaque dictionnaire lexical décrit le vocabulaire d'une langue donnée. Pour le présent projet, nous nous sommes limitée au français. Ainsi, nos données proviennent du RL-fr et plus précisément de la thèse de Pausé (2017). Celle-ci propose des patrons linguistiques qui permettent de classifier les locutions du français (voir section 3.2.2).

Dans la présente section, nous présentons d'abord la structure du dictionnaire lexical et les principales fonctions qui ont permis de le générer automatiquement.

4.2.1 Structure du dictionnaire

Notre dictionnaire est organisé en trois parties. La première regroupe les valeurs par défaut pour des classes de lexies, la deuxième répertorie les entrées des patrons linguistiques et la dernière décrit les locutions du français.

Les données de la première partie sont présentées sous forme hiérarchique, du plus générique au plus spécifique. Le dictionnaire commence par une présentation des **informations métalinguistiques**. Puis, il répertorie les valeurs par défaut des attributs. Pour chacune des parties du discours, les précisions nécessaires sont ajoutées (voir le manuel de GÉCO [Lareau et Lambrey, 2016] pour plus d'informations à ce sujet).

Comme nous l'avons mentionné à la section 2.1.1, GenDR utilise un mécanisme d'héritage pour factoriser l'information en **classes** (par exemple, prédicats, verbes, verbes transitifs directs, etc.). Cela signifie que chaque entrée du dictionnaire dépend d'une classe et hérite de ses traits. À titre d'exemplifier, nous reprenons la locution ⌜COUPER LES CHEVEUX EN QUATRE⌝ de la figure 47. La locution hérite des traits de son patron générique, qui lui-même hérite des traits de sa partie du discours (verbe) ; ce verbe hérite à son tour des traits par défaut de tous les prédicats. D'ailleurs, tous les prédicats ont une diathèse triviale par défaut ; par exemple, le premier actant d'un verbe donne une relation subjectale.

```
predicate {
  gp = { 1 = I
         2 = II
         3 = III
         4 = IV
         5 = V
         6 = VI }
}

verb: predicate {
  dpos = V
  spos = verb
  auxiliary = avoir_aux
  gp = {
     I = {
        dpos = N
        rel = subj
     }
  }
}

"V Art NC Prép Num": verb {
  …
}

"couper les cheveux en quatre": "V Art NC Prép Num" {
  …
}
```

Figure 47 – Exemple d'héritage dans le dictionnaire lexical

Dans la deuxième partie du dictionnaire, les **patrons linguistiques** sont organisés selon une nomenclature alphabétique. Chaque entrée est constituée de façon analogue. La figure 48 montre un exemple d'entrée de dictionnaire pour le patron linguistique `V Art NC Prép Num`. La première ligne de chaque entrée correspond à un exemple de locution pour le patron linguistique en question. La deuxième ligne énonce d'abord le patron linguistique avant de préciser la partie du discours à laquelle il appartient. On indique ensuite à l'aide d'accolades que cette entrée porte le trait `idiom` (pour locution). Puis, on précise à quel type de patron générique est associé ce patron linguistique (au patron générique `5_05` dans l'exemple). L'entrée de dictionnaire se termine par une liste des relations qui existent entre les éléments de l'arbre.

```
// ex : couper les cheveux en quatre
"V Art NC Prép Num" : "locution verbale" {
  idiom = {
    type = idiom_5_05
    r1 = relx
    r2 = relx
    r3 = relx
    r4 = relx
  }
}
```

Figure 48 – Entrée de dictionnaire pour le patron linguistique `V Art NC Prép Num`

Pour le moment, les **relations de dépendance** à l'intérieur des locutions ne sont pas précisées dans le dictionnaire lexical : elles portent toutes l'étiquette `relx`. Le choix des noms de relations dépasse le cadre de notre travail, puisque ceux-ci sont propres à une langue donnée. Par exemple, en français, il existe des relations subjectale, déterminative, objective directe, adverbiale, etc. (Polguère, 1998). Ces décisions reviennent donc aux lexicographes responsables du RL-fr. Nos patrons se veulent les plus génériques possibles pour convenir au plus grand nombre de langues possible.

La troisième et dernière partie du dictionnaire suit la même logique que la précédente ; elle répertorie les **locutions** du français tirées du RL-fr[25] (Pausé, 2017) en ordre alphabétique. Nous présentons un exemple d'entrée lexicale complet pour la locution ⌜COUPER LES CHEVEUX EN QUATRE⌝ à la figure 49. Cette entrée dépend du patron présenté précédemment. Dans la première ligne de l'entrée, on énonce d'abord la locution, puis on précise le patron linguistique associé à celle-ci. Ensuite, on indique de nouveau que cette entrée porte le trait `idiom`. Finalement, on liste tous les nœuds que comporte cette locution et leurs étiquettes. Ces dernières sont attribuées lors de la mise en correspondance présentée à la section 4.1.1.2.

[25] En date du 18/06/2019.

```
"couper les cheveux en quatre" : "V Art NC Prép Num" {
idiom = {
    w0 = "couper"
    w1 = "cheveux"
    w2 = "les"
    w3 = "en"
    w4 = "quatre"
  }
}
```

Figure 49 – Entrée de dictionnaire pour la locution ⌜COUPER LES CHEVEUX EN QUATRE⌝

Chacune des entrées mentionnées ci-dessus peut être inscrite manuellement dans le dictionnaire. Cependant, il s'agit d'une tâche fastidieuse et coûteuse. Nous avons donc automatisé l'écriture du dictionnaire à l'aide d'un script. Nous le présentons dans la prochaine section, qui porte sur la génération automatique du dictionnaire lexical.

4.2.2 Génération automatique du dictionnaire

Afin de réaliser du texte, il est nécessaire d'avoir des données lexicographiques sur lesquelles s'appuyer. Comme nous l'avons mentionné précédemment, les données de notre projet proviennent du RL-fr. Cette collaboration permet à l'équipe de GenDR de tester les limites du réalisateur et, parallèlement, l'utilisation de GenDR permet de vérifier la précision des définitions lexicographiques proposées par le RL-fr.

La génération automatique du dictionnaire a été réalisée grâce à un second script, *idioms.py*. La présente section traitera d'abord des données d'entrée du script, puis des fonctions principales qu'il utilise pour l'écriture du dictionnaire.

En premier lieu, le script prend en entrée les données du RL-fr, qui sont accessibles à l'aide du navigateur **Spiderlex**[26] (Gader et coll., 2014 ; Ollinger et coll., 2020 ; Ollinger et Polguère, 2020). Les tableaux 11 et 12 présentent des extraits des données que nous avons utilisées. On y retrouve des informations sur les patrons linguistiques et les locutions qu'ils décrivent (voir section 3.2.2 pour une liste complète). Le tableau 11 présente l'encodage des caractéristiques grammaticales des parties du discours ; par exemple, `ls:fr:gc:21` est l'identifiant attribué aux locutions nominales. Pour sa part, le tableau 12 montre entre autres comment sont encodées diverses

[26] https://lexical-systems.atilf.fr/spiderlex/).

caractéristiques grammaticales. Chaque colonne du tableau 12 représente un nœud du réseau, c'est-à-dire une entrée distincte du dictionnaire.

```
- <characteristic id="ls:fr:gc:17" name="CARACTÉRISTIQUES GRAMMATICALES FONDAMENTALES" type="0" status="0">
  - <characteristic id="ls:fr:gc:18" name=" Partie du discours" type="0" status="0">
    - <characteristic id="ls:fr:gc:109" name="V" type="0" status="0">
      - <characteristic id="ls:fr:gc:23" name="verbe" type="2" status="0"/>
      - <characteristic id="ls:fr:gc:24" name="verbe pronominal" type="2" status="0"/>
      - <characteristic id="ls:fr:gc:78" name="locution verbale" type="2" status="0"/>
      - <characteristic id="ls:fr:gc:195" name="syntagme verbal" type="2" status="0"/>
     </characteristic>
    - <characteristic id="ls:fr:gc:111" name="N" type="0" status="0">
      - <characteristic id="ls:fr:gc:20" name=" nom commun" type="2" status="0"/>
      - <characteristic id="ls:fr:gc:21" name="locution nominale" type="2" status="0"/>
      - <characteristic id="ls:fr:gc:92" name="nom propre" type="2" status="0"/>
      - <characteristic id="ls:fr:gc:216" name=" nom propre numéral" type="2" status="0"/>
      - <characteristic id="ls:fr:gc:268" name=" nom propre prénom" type="2" status="0"/>
      - <characteristic id="ls:fr:gc:196" name=" locution nominale nom propre" type="2" status="0"/>
      - <characteristic id="ls:fr:gc:80" name="numéral" type="2" status="0"/>
      - <characteristic id="ls:fr:gc:197" name="locution numérale" type="2" status="0"/>
```

Tableau 11 – Extrait d'un fichier XML de Spiderlex

"node"	"ls:fr:node:38386"	"ls:fr:node:44800"	"ls:fr:node:55384"	"ls:fr:node:45663"
"usagenote"	""	""	""	""
"usagenotevars"	""	""	""	""
"POS"	"ls:fr:gc:20"	"ls:fr:gc:78"	**"ls:fr:gc:21"**	"ls:fr:gc:78"
"phraseolstruc"	""	"AdvNég V ($2)"	"NC Prép NC"	"Vpro Art NC"
"embeddedlex"	""	"((ls:fr:node:48619,), (ls:fr:node:53629,), (,))"	"((ls:fr:node:55386,), (ls:fr:node:43036,), (ls:fr:node:55386,))"	"((ls:fr:node:35647,), (, la), (ls:fr:node:27539,))"
"othercharac"	"(ls:fr:gc:28)"	"(ls:fr:gc:166)"	"(ls:fr:gc:28,ls:fr:gc:167)"	"(ls:fr:gc:166)"
"othercharacvars"	"(())"	"(())"	"((),())"	"(())"

Tableau 12 – Extrait d'un fichier CSV de Spiderlex

Afin d'assurer la mise en correspondance des patrons linguistiques et des patrons génériques, nous avons créé une table de données Pandas[27] dans Python[28] (voir tableau 13 pour un extrait). Celle-ci comprend tous les patrons linguistiques tirés du RL-fr (`pat`), accompagnés d'un exemple de locution (`ex`) ; nous y avons ajouté les informations permettant leur mise en correspondance (`map`) avec nos patrons génériques (`tree_id`).

Pat	ex	tree_id	map	rem
Prép NC	de terrain	2	10	
AdvNég V ($2)	ne pas voler	3_02	120	
NC Prép NC	bande de circulation	3_01	120	
Vpro Art NC	casser la figure	3_01	210	

Tableau 13 – Extrait de la table de données `idiom_patterns.tsv`

À partir de tous ces fichiers, nous avons construit une table de données qui comprend l'information nécessaire pour la description des locutions. Pour débuter, nous avons uniquement sélectionné les entrées dont la partie du discours correspond à une locution ; en d'autres termes, nous avons extrait les entrées qui possèdent l'une des étiquettes suivantes :

- locution adjectivale ;
- locution adverbiale ;
- locution conjonctive ;
- locution interjective ;
- locution nominale ;
- locution numérale ;
- locution phrastique ;
- locution pronominale ;
- locution propositionnelle ;
- locution prépositionnelle ;
- locution verbale.

En deuxième lieu, nous avons créé une table de données regroupant les différents fichiers d'entrée nous étant utiles. À partir de cette table, nous avons créé une nouvelle table de données qui sélectionne uniquement les colonnes pertinentes à notre travail ; elle comporte les données

[27] https://pandas.pydata.org/
[28] https://www.python.org/

du RL-fr et y inclut les nôtres (voir tableau 14 pour un extrait). Voici les champs que nous avons conservés du RL-fr et leurs étiquettes dans notre table :

- l'identifiant des caractéristiques grammaticales (`gc_id`) ;
- la partie du discours (`pos`) ;
- l'identifiant du lexème (`node_id`) ;
- le patron linguistique (`pat`) ;
- l'identification du vocable (`entry_id`) ;
- le numéro de lexie (`lexnum`) ;
- le pronom réfléchi (`se`) ;
- le nom de la locution (`idiom_name`) ;
- l'identifiant de l'acception (`subscript`) ;
- l'identifiant de vocable (`supercript`) ;
- les identifiants donnés aux actants (`actants`).

Voici maintenant les variables que nous avons ajoutées :

- la fréquence du patron linguistique (`freq`) ;
- un exemple de locution pour chaque patron linguistique (`ex`) ;
- l'identifiant du patron générique (`tree_id`) ;
- l'identifiant de correspondance entre les patrons (`map`) ;
- les remarques (rem) ;
- le nom standardisé de la locution (`std_name`).

std_name	à table#1	ne pas manger#2	mot à mot # N	se casser la figure#I
gc_id	ls:fr:gc:22	ls:fr:gc:78	ls:fr:gc:21	ls:fr:gc:78
node_id	ls:fr:node:43361	ls:fr:node:44800	ls:fr:node:55384	ls:fr:node:45663
entry_id	ls:fr:entry:38386	ls:fr:entry:39248	ls:fr:entry:44574	ls:fr:entry:39683
se				se
idiom_name	à table	ne pas manger	mot à mot	casser la figure
subscript			N	
superscript		2		
lexnum	1			I
actants	($ 1=X)	($ 1=X, $ 2=Y)	($ 1=X, $ 2=Y)	($ 1=X)
pos	locution prépositionnelle	locution verbale	locution nominale	locution verbale
pat	Prép NC	AdvNég V ($2)	NC Prép NC	Vpro Art NC
ex	de terrain	ne pas voler	bande de circulation	casser la figure
tree_id	2	3_02	3_01	3_01
map	01	120	012	021
rem				

Tableau 14 – Extrait de la table de données combinées `idioms`

À partir de cette table, nous pouvons procéder à l'**écriture du dictionnaire** *idioms.dict*. Pour ce faire, il faut créer un texte à trous et indiquer au script où il peut aller chercher les informations nécessaires dans la table. Nous avons créé deux boucles pour effectuer l'écriture des entrées ; la première est destinée aux patrons linguistiques et la seconde, aux locutions. La figure 50 présente les commandes utilisées pour l'écriture de chaque patron linguistique. Le gras permet de visualiser le texte troué écrit par le script. La figure 51 représente la correspondance entre ce script et l'entrée de dictionnaire finale. Dans cette figure, le texte en gras représente les trous qui seront complétés à l'aide de l'information qui se trouve dans la colonne (`row`) sélectionnée du tableau. Enfin, la figure 52 présente un exemple d'entrée pour un patron linguistique dans lequel les « trous » sont maintenant remplis et mis en relief par le gras.

```
for i, row in patterns.iterrows():
        mate = ''
        if isinstance(row.rem, str):
            mate += f'\n// {row.rem}'
        mate += f'''
                // ex : {row.ex}
                "{row.pat}" : "{row.pos}" {{
                  idiom = {{
                    type = idiom_{row.tree_id}
                '''
        if not np. isnan(float(row.tree_id)):
            n = int(str(row.tree_id).strip().split('_')[0])
            for i in range(1, n):
                mate += f'    r {i} = relx\n'
        mate += '''  }
                }
                '''
```

Figure 50 – Script pour écrire les entrées des patrons linguistiques

```
// ex : {row.ex}
"{row.pat}" : "{row.pos}" {
  idiom = {
    type = idiom_{row.tree _id}
    r{i}  = relx
  }
}
```

Figure 51 – Exemple de correspondance entre le script et le dictionnaire lexical

```
// ex : de terrain
"Prép NC" : "locution prépositionnelle" {
  idiom = {
    type = idiom_2
    r1 = relx
  }
}
```

Figure 52 – Exemple d'entrée pour un patron linguistique

Le produit du script correspond ainsi aux deux dernières parties du dictionnaire lexical (voir figure 53 pour un extrait), évoquées à la section précédente.

```
…
//==================IDIOM PATTERNS====================
…
// ex : de terrain
"Prép NC" : "locution prépositionnelle" {
  idiom = {
    type = idiom_2
    r1 = relx
  }
}
…
//=====================IDIOMS=========================
…
"par exemple": "Prép NC" {
idiom = {
    w0 = "par"
    w1 = "exemple"
  }
}
…
```

Figure 53 – Extrait du dictionnaire lexical

Dans le dictionnaire lexical, deux unités lexicales ne peuvent posséder la même graphie ; cela est proscrit par notre réalisateur. Nous avons donc créé une fonction pour différencier les locutions de notre dictionnaire. Cette fonction, présentée à la figure 54, porte le nom de `standard_name()`. Son application attribue un nom standardisé à chaque locution de la table de donnée. Cette variable est appelée `std_name` ; elle réunit les (signifiants des) locutions avec leur pronom réfléchi, leur numéro de vocable (*superscript*) et leur numéro d'acception (*subscript*). Ainsi, chaque ligne de la table représente une locution au sens distinct.

```
def standard_name(row):

    std_name = ''
    if isinstance(row.se, str):
        if row.se == 'se':
            std_name += f'{row.se} '
        else:
            std_name += f'{row.se}'
    std_name += f'{row.idiom_name}'
    if not (pd.isna (row.subscript) and pd.isna (row.superscript)
    and pd.isna (row.lexnum)):
        std_name += '#'
        script = False
        if not pd.isna(row.subscript):
            script = True
            std_name += f'{row.subscript}'
        if not pd.isna(row.superscript):
            script = True
            std_name += f'{row.superscript}'
        if not pd.isna(row.lexnum):
            if script:
                std_name += '_'
            std_name += f'{row.lexnum}'
    return std_name
```

Figure 54 – Fonction `standard_name()` extraite de `idioms.py`

L'écriture des entrées pour les locutions a également demandé certains ajustements. Il nous fallait bien encoder dans notre dictionnaire lexical les étiquettes des nœuds utilisées lors de la lexicalisation. Nous faisons référence à la mise en correspondance des patrons décrite à la section 4.1.1.2. Les trois fonctions suivantes ont ainsi permis l'écriture du texte en gras dans la figure 55.

```
"couper les cheveux en quatre" : "V Art NC Prép Num" {
idiom = {
    w0 = "couper"
    w1 = "cheveux"
    w2 = "les"
    w3 = "en"
    w4 = "quatre"
  }
}
```

Figure 55 – Rappel de l'entrée de dictionnaire pour ⌜COUPER LES CHEVEUX EN QUATRE⌝

En premier lieu, la fonction `reorder()` prend en entrée les lexèmes de la locution et les ordonnes en fonction du code de correspondance. Rappelons que celui-ci réorganise les unités selon leur positionnement dans le patron générique. La figure 56 présente cette fonction.

```
def reorder(std_name, mapping):
    return [to_list(std_name.split ('#')[0])[int (i)] for i in
mapping]
```

Figure 56 – Fonction `reorder()` extraite de `idioms.py`

L'exécution de cette fonction appelle une autre fonction, nommée `to_list()`. Celle-ci prend en entrée une locution au nom standardisé et la sépare en une chaîne de lexèmes (voir figure 57).

```
def to_list(std_name):
    std_name = re.sub (r"#.+$", "", std_name)
    std_name = re.sub (r"['‘`´’’‘‘`]", "’", std_name)
    std_name = re.sub (r"’", "’ ", std_name)
    std_name = re.sub (r" ([!\.,;\?:])", "", std_name)
    std_name = normalize_amalgams(std_name)
    std_name = re.sub(r"  +", " ", std_name)
    std_name = re.sub(r" aujourd’ hui", "aujourd’hui", std_name)
    std_name = re.sub(r"Vas-y", "Vas — y", std_name)
    return std_name.split()
```

Figure 57 – Fonction `to_list()` extraite de `idioms.py`

La fonction `to_list()` dépend également d'une autre fonction, nommée `normalize_amalgams()`, qui normalise le texte en enlevant les amalgames (voir figure 58).

```
def normalize_amalgams(text):
    text = re.sub (r" ( |^)au ", r"\1à le ", text)
    text = re.sub(r"( |^)aux ", r"\1à les ", text)
    text = re.sub(r"( |^)du ", r"\1de le ", text)
    text = re.sub(r"( |^)des ", r"\1de les ", text)
    text = re.sub(r"( |^)auquel ", r"\1à lequel ", text)
    text = re.sub(r"( |^)auxquel ", r"\1à lesquels ", text)
    text = re.sub(r"( |^)duquel ", r"\1de lequel ", text)
    text = re.sub(r" ( |^)desquels ", r"\1de lesquels ", text)
    return text
```

Figure 58 – Fonction `normalize_amalgams()` extraite de `idioms.py`

La fonction `normalize_amalgams()` a plusieurs utilités. Dans le cadre de la TST, les amalgames sont exclus des représentations syntaxiques de surface ; ils sont plutôt traités au niveau morphologique (voir section 2.2.1 portant sur les niveaux de représentation de la TST).

De plus, cette fonction fait en sorte que le nombre d'éléments de la locution corresponde au nombre d'éléments du patron linguistique et, de ce fait, au nombre d'éléments du patron générique. Sans cela, le script ne peut procéder à la mise correspondance des patrons. C'est d'ailleurs pour cette raison que nous avons dû porter une attention particulière aux pronoms

réfléchis, aux mots composés, aux amalgames et aux patrons linguistiques contenant des locutions imbriquées.

En conclusion, grâce au script *idioms.py,* il nous a été possible de générer automatiquement des entrées de dictionnaire GenDR à partir des données lexicographiques du RL-fr. Le script *partition.py* nous a ensuite permis de générer automatiquement des règles de transduction de graphes qui s'appuient sur des patrons génériques. Ces patrons génériques, inspirés de la partition des entiers, répertorient tous les arbres possibles à *n* nœuds. Enfin, le produit de ces deux scripts permet à notre réalisateur d'assurer le traitement des locutions et de générer efficacement leur RSyntS. Le prochain chapitre traite de l'évaluation de cette implémentation.

Chapitre 5 Évaluation

Dans le présent chapitre, nous présentons l'évaluation de notre implémentation. Celle-ci se concentre sur la lexicalisation de surface des locutions dans GenDR. L'évaluation est basée sur deux critères. D'abord, nous évaluons la couverture de l'implémentation, c'est-à-dire le pourcentage des locutions du RL-fr que nous pouvons générer. Puis, nous évaluons la précision de l'implémentation, c'est-à-dire la proportion des structures générées qui sont correctement formées.

5.1 Couverture

5.1.1 Méthodologie

La couverture de notre implémentation est mesurée en calculant le nombre de locutions que nous traitons sur le total de locutions répertoriées par le RL-fr. Notre traitement consiste en la création de règles de lexicalisation s'appuyant sur des patrons génériques qui décrivent la RSyntS de locutions. Il repose sur le regroupement des locutions en patrons linguistiques en fonction de leurs caractéristiques grammaticales et de leur combinatoire. Seules les locutions qui sont associées à un patron linguistique ont donc été incluses.

5.1.2 Résultats

Notre corpus est composé de **2 919 locutions** répertoriées par le RL-fr[29]. Ce sont principalement des locutions nominales (48 %), prépositionnelles (22 %) et verbales (22 %). Ces locutions sont classées selon 514 patrons linguistiques (Pausé, 2017). Le tableau 15 énumère les patrons de locutions les plus fréquents du RL-fr. Ceux-ci sont accompagnés de leur partie du discours, leur patron générique et d'un exemple de locution.

[29] Données exportées en date du 18/06/2019.

Patron linguistique	Pat. gén.	Exemple de locution	Nb de loc	%	PDD
NC Prép NC	3_01	⌜ÉPREUVE DE FORCE⌝	409	14 %	LocN
NC Adj	2	⌜IDENTITÉ NATIONALE⌝	377	13 %	LocN
Prép NC	2	⌜DE JUSTESSE⌝	222	8 %	LocPrép
NC Prép. Circ NC	3_01	⌜BULLETIN DE VOTE⌝	138	5 %	LocN
Adj NC	2	⌜PETIT PEUPLE⌝	100	4 %	LocN
Prép Art NC	3_01	⌜DANS LA FOULÉE⌝	98	3 %	LocPrép
V Art NC	3_01	⌜REMONTER LA PENTE⌝	79	3 %	LocV
NC Prép Art NC	4_01	⌜POLITIQUE DE L'AUTRUCHE⌝	79	3 %	LocN
Autre			1417	49 %	
Total			**2919**	**100 %**	

Tableau 15 – Patrons linguistiques les plus fréquents du RL-fr

Notre traitement se limite actuellement aux **locutions de six unités ou moins**. Cela correspond à une couverture de 97,5 % des locutions répertoriées par le RL-fr, soit 2846 locutions. Nous détaillons ce pourcentage de couverture selon le nombre de nœuds dans le tableau 16.

	Nb de nœuds	Nb de locutions	%
Locutions couvertes	2	968	33 %
	3	1185	41 %
	4	429	15 %
	5	166	6 %
	6	98	3 %
Locutions non couvertes	7	39	1 %
	8	22	1 %
	9	7	0,2 %
	10	3	0,1 %
	11	1	0,03 %
	12	—	—
	13	—	—
	14	—	—
	15	1	0,03 %

Tableau 16 – Couverture des locutions dans GenDR selon le nombre de nœuds

Seulement 73 locutions (réparties en 62 patrons linguistiques) ne sont pas couvertes par notre implémentation. Le tableau 17 ci-dessous permet de mettre en lumière notre couverture des locutions en fonction de la partie du discours à laquelle elles appartiennent. Nous y remarquons

une haute couverture de toutes les parties du discours, sauf pour les locutions phrastiques (67 %). Cela s'explique par leur longueur moyenne, qui dépasse généralement celle des autres locutions. Cela dit, sur l'ensemble des locutions non traitées, on ne compte que 14 locutions phrastiques, contre 45 locutions verbales. La majorité des locutions non traitées sont donc des locutions verbales (62 %) et les autres sont des locutions phrastiques (19 %), nominales (7 %), prépositionnelles (7 %), conjonctives (4 %) et propositionnelles (1 %).

Partie du discours	Nb locutions total	Nb locutions traitées	% couverture
locution nominale	1414	1409	99,6 %
locution prépositionnelle	655	650	99 %
locution verbale	646	601	93 %
locution conjonctive	87	84	97 %
locution phrastique	42	28	67 %
locution adjectivale	35	35	100 %
Locution adverbiale	21	21	100 %
locution propositionnelle	8	7	88 %
locution numérale	5	5	100 %
locution interjective	4	4	100 %
locution pronominale	2	2	100 %
Total	**2919**	**2846**	**97,5 %**

Tableau 17 – Nombre de locutions du RL-fr et pourcentage de notre couverture en fonction de la partie du discours

Le tableau 18 répertorie la fréquence de nos patrons génériques par rapport aux données du RL-fr. On y voit par le fait même la répartition des locutions par rapport aux patrons génériques. Sur les 36 patrons, seuls 29 sont représentés par les locutions du RL-fr[30].

[30] Il n'y a pas de locutions répertoriées pour les sept autres patrons génériques.

Patron générique	2	3_01	3_02	4_01	4_02	4_03	4_04	5_01	5_02
Fréquence	968	1141	44	325	44	53	7	20	24
	5_03	5_04	5_05	5_06	5_07	5_08	5_09	6_01	6_02
	42	—	14	55	2	8	1	2	—
	6_03	6_04	6_05	6_06	6_07	6_08	6_09	6_10	6_11
	8	—	1	15	11	2	—	37	2
	6_12	6_13	6_14	6_15	6_16	6_17	6_18	6_19	6_20
	3	6	—	—	1	6	1	3	—

Tableau 18 – Fréquence des patrons génériques dans le RL-fr[31]

5.1.3 Discussion

Nous avons limité notre couverture des locutions aux **arbres de six nœuds ou moins** (donc aux locutions de six lexèmes ou moins). Cette décision a été prise pour assurer le traitement efficace des locutions. Elle s'appuie sur deux facteurs. Premièrement, le **nombre d'arbres** (ou de patrons génériques) possibles croît exponentiellement avec le nombre de nœuds. À cet effet, le tableau 19 répertorie le nombre d'arbres différents à *n* nœuds théoriquement possibles, ainsi que la fréquence d'utilisation de ces patrons pour les locutions du RL-fr. On remarque que le nombre d'arbres possibles devient rapidement plus élevé que le nombre même de locutions à décrire.

Nb de nœuds	Nb d'arbres	Nb de locutions
2	1	968
3	2	1185
4	4	429
5	9	166
6	20	98
7	48	39
8	115	22
9	286	7
10	719	3
11	1842	1

Tableau 19 – Nombre d'arbres possibles à *n* nœuds et les locutions du RL-fr qu'ils couvrent

Deuxièmement, on observe une **récursivité** dans les patrons génériques. Un arbre étant une structure intrinsèquement récursive, un sous-arbre est lui-même un arbre. Ainsi, nous pouvons

[31] Basé sur les données du RL-fr en date du 18/06/2019.

concevoir l'organisation interne des locutions comme des poupées russes, les unes imbriquées dans les autres. Par conséquent, nous pouvons regrouper les mots-formes en groupes qui ne correspondent à rien d'un point de vue lexicologique, mais qui opèrent comme une chaîne de caractères invariable d'un point de vue computationnel. Autrement dit, il est possible d'organiser une longue locution en une somme de constituants plus petits correspondant eux-mêmes à des patrons génériques. Il s'agit d'une solution économique pour le traitement des locutions longues. Par exemple, la locution verbale ⌜METTRE LES PETITS PLATS DANS LES GRANDS⌝ peut être représentée par un arbre à sept nœuds (voir figure 59).

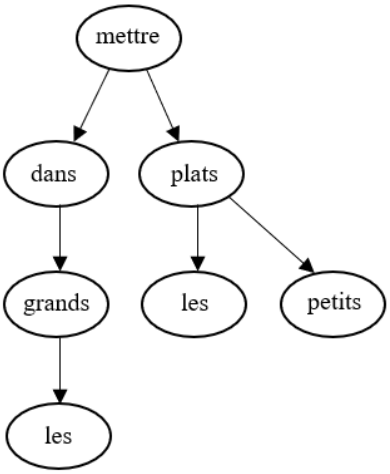


Figure 59 – Arborisation de la locution ⌜METTRE LES PETITS PLATS DANS LES GRANDS⌝

Pour simplifier cet arbre, nous pouvons le reconfigurer en un arbre à trois nœuds où *les petits plats* et *dans les grands* forment deux sous-arbres qui dépendent de METTRE. Le verbe *mettre* est le seul élément de la locution qui soit fléchi, donc le seul qui nécessite d'être isolé des autres éléments. Le haut de la figure 60 illustre la première étape de cette reconfiguration, alors que le bas illustre la deuxième, qui consiste à reconfigurer chacun des sous-arbres selon leur patron générique respectif.

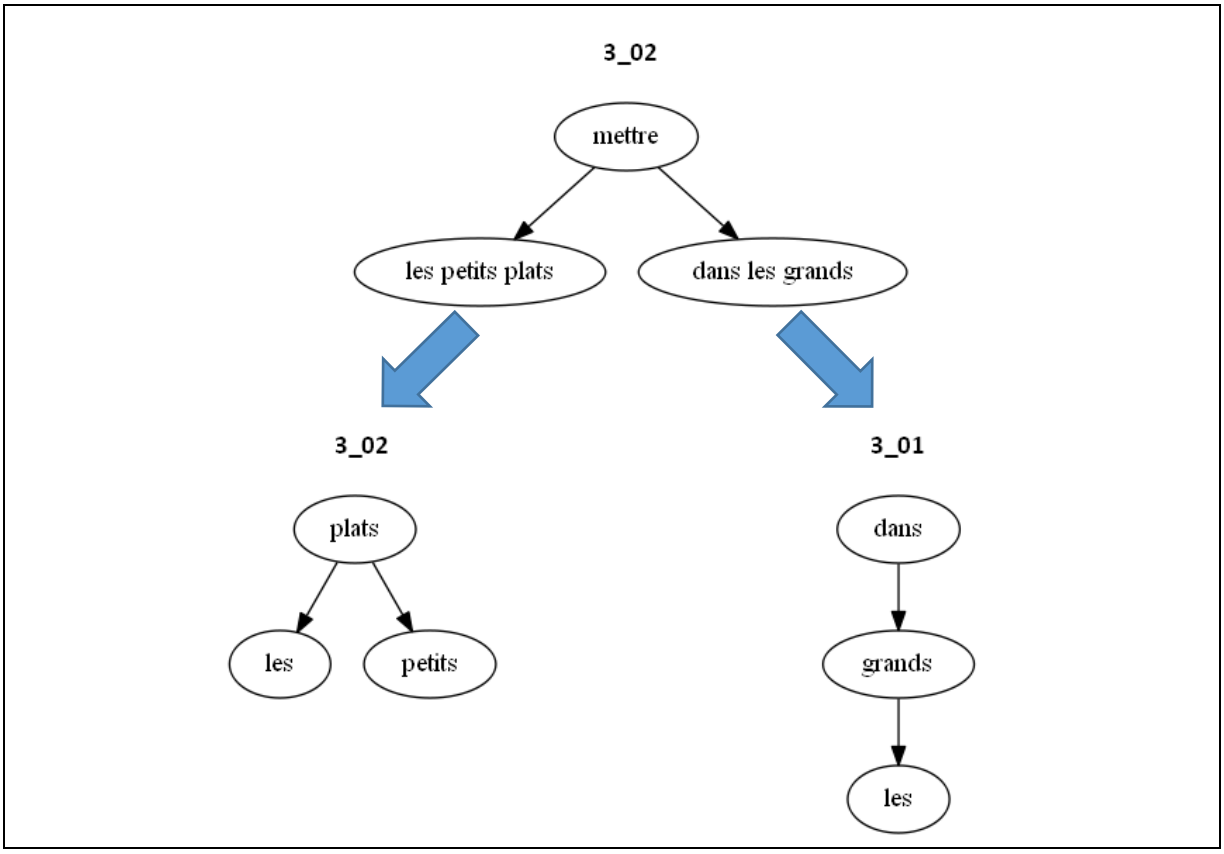


Figure 60 – Exemple de patrons récursifs pour ⌜METTRE LES PETITS PLATS DANS LES GRANDS⌝

Il faut noter que cette reconfiguration n'est pas implémentée pour le moment. Cette solution permettrait de réutiliser nos travaux comme base de conception pour faire le traitement des locutions plus longues. À ce sujet, le tableau 20 présente un exemple de locution non traitée par partie du discours, et comment nous pourrions la découper. Comme nous pouvons le voir, les locutions se prêtent bien à la réorganisation en sous-arbres, expliquée ci-dessus[32].

[32] De nombreuses réorganisations sont possibles et nous ne présentons qu'un exemple parmi tant d'autres. D'un point de vue strictement pratique, les locutions dans lesquelles aucun élément n'est fléchi peuvent très bien être traitées comme un seul nœud (c'était l'approche de MARQUIS, voir section 1.2.2.4).

Partie du discours	Locution	Nb de nœuds
locution nominale	⌜PIQÛRE | DE MOUCHE SUR LA FESSE D'UN ÉLÉPHANT⌝	9
locution prépositionnelle	⌜EN | DEUX COUPS DE CUILLÈRE À POT⌝	7
locution verbale	⌜NE PAS | Y | ALLER | PAR QUATRE CHEMINS⌝	7
locution conjonctive	⌜COMME | LE NEZ AU MILIEU DE LA FIGURE⌝	9
locution phrastique	⌜CE | N' | EST | PAS | LA MORT DU PETIT CHEVAL.⌝	10
locution propositionnelle	⌜SI | LES COCHONS NE LE MANGENT PAS⌝	7

Tableau 20 – Exemple de locutions non traitées[33]

5.2 Précision

5.2.1 Méthodologie

La précision de notre implémentation a été mesurée manuellement. Comme nous l'avons mentionné au chapitre 0, la GAT ne s'intéresse que très peu aux locutions. Il n'existe donc pas de test standardisé pour mesurer la qualité du travail produit.

Nous avons vu au chapitre 2 que GenDR permet de prendre en entrée une structure profonde telle qu'une RSém. Cependant, chaque module de GenDR peut également être utilisé séparément. Notre évaluation ne prend donc en compte que le module qui assure le passage entre la RSyntP et la RSyntS. Cela permet de nous concentrer sur la lexicalisation de surface puisque c'est dans cette dernière qu'a lieu le traitement des locutions (lors du passage de la RSém à la RSyntP, la locution demeure une seule unité).

Nous avons ainsi commencé notre évaluation en générant automatiquement les RSyntP pour chaque locution du dictionnaire lexical. À l'aide d'un script, nous avons récupéré les entrées des locutions et généré des RSyntP en format MATE, où les locutions apparaissent avec leurs actants. La figure 61 est un exemple de structures d'entrée utilisée pour l'évaluation.

[33] Le sommet syntaxique est souligné.

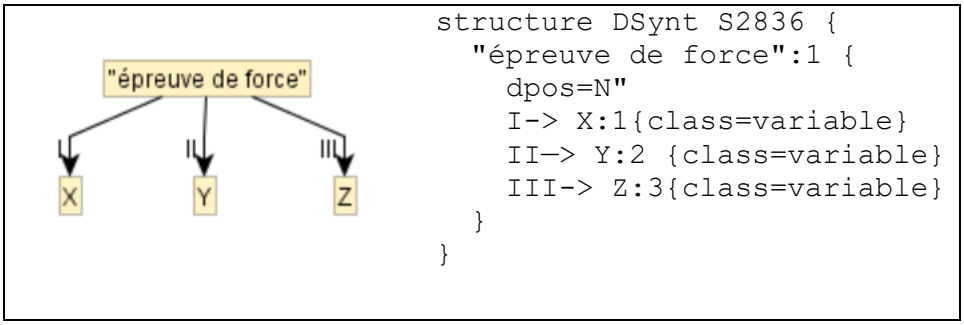


Figure 61 – Exemple de RSyntP d'une locution dans GenDR

Nous avons sélectionné aléatoirement **500 structures** à évaluer grâce à un script de randomisation. Parmi celles-ci, 300 structures ont subi une double évaluation ; les 200 restantes ont été partagées équitablement entre les deux juges (évaluation simple de 100 structures par juge). Cela permettait, d'une part, une évaluation commune d'au moins 10 % des structures et, d'autre part, d'accroître la couverture de notre évaluation à 18 % du corpus, grâce à l'évaluation simple.

Notre échantillonnage a permis de tester 18 des règles de lexicalisation par patron (décrites aux sections 4.1.2 et 4.1.3), soit 50 % des règles (qui représentent chacun un patron générique). Le tableau 21 donne la répartition des locutions évaluées par rapport aux patrons génériques (voir tableau 18 pour la répartition de toutes les locutions du RL-fr).

Patrons génériques	**2**	**3_01**	**3_02**	**4_01**	**4_02**	**4_03**	**4_04**	**5_01**	**5_02**
Fréquence	161	206	4	62	6	9	1	5	7
	5_03	**5_04**	**5_05**	**5_06**	**5_07**	**5_08**	**5_09**	**6_01**	**6_02**
	7	—	2	13	—	2	—	—	—
	6_03	**6_04**	**6_05**	**6_06**	**6_07**	**6_08**	**6_09**	**6_10**	**6_11**
	—	—	1	2	—	1	—	10	—
	6_12	**6_13**	**6_14**	**6_15**	**6_16**	**6_17**	**6_18**	**6_19**	**6_20**
	—	—	—	—	—	1	—	—	—

Tableau 21 – Répartition des locutions évaluées par patron générique

Les personnes évaluant ce travail ont été sélectionnées sur la base de leur formation en linguistique computationnelle. Elles devaient avoir une connaissance approfondie à la fois du système à évaluer et de son cadre théorique, soit GenDR et la TST.

Ces personnes ont observé le **protocole** suivant : pour chacune des structures à évaluer, il fallait ouvrir la RSyntP dans MATE et générer la RSyntS correspondante. Ce processus permettait de tester les règles de lexicalisation par patron de GenDR. L'évaluation a été basée sur la justesse de la RSyntS ; le juge devait vérifier si la structure générée était celle attendue. Une structure était jugée correcte si elle comprenait tous les nœuds de la locution et que ceux-ci dépendaient du bon gouverneur. Si une structure était jugée problématique, il fallait remplir la colonne appropriée dans la grille d'annotation pour préciser la source du problème. Parmi les problèmes potentiels, il y avait la mauvaise application ou le mauvais choix de patron générique ou de code de correspondance pour la mise en correspondance des patrons (voir l'annexe 2 pour le protocole complet).

Les figures 62 et 63 présentent respectivement un exemple de RSyntP et de RSyntS d'une locution dans GenDR. La locution ⌜VICE DE FORME⌝ y est représentée graphiquement à gauche et textuellement à droite.

Figure 62 – Exemple de RSyntP d'une locution dans GenDR

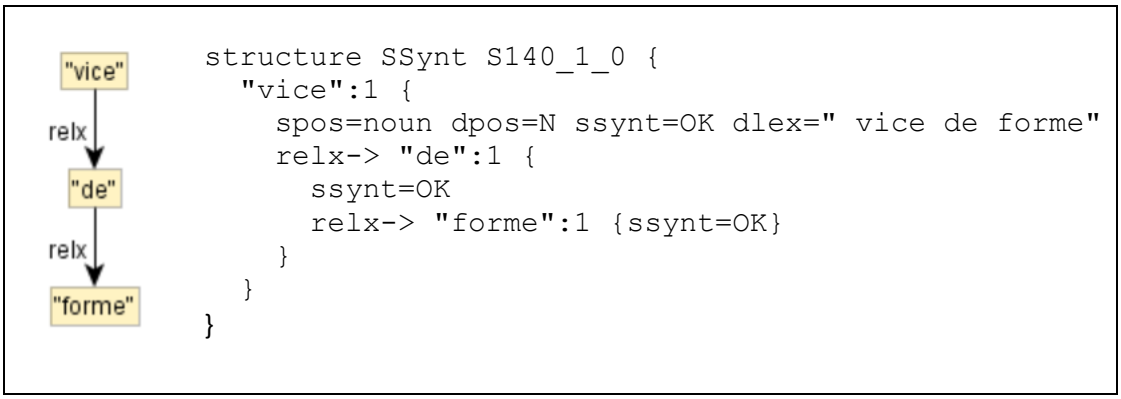


Figure 63 – Exemple de RSyntS d'une locution dans GenDR[34]

[34] Des guillemets anglais sont ajoutés autour des locutions pour faciliter le traitement d'unités lexicales complexes (qui comportent souvent des espaces ou de la ponctuation). Bien que ces guillemets soient systématiquement ajoutés pour le traitement des locutions, cela n'est pas systématique pour toutes les unités syntaxiques dans GenDR. Cette homogénéisation du traitement des lexèmes sera à faire dans une future révision.

Nous avons calculé l'accord interjuge à l'aide du Kappa de Cohen (Cohen, 1960), un coefficient qui prend en compte les effets du hasard dans la mesure de l'accord interjuge.

5.2.2 Résultats

En premier lieu, la lexicalisation de surface s'est appliquée pour toutes les RSyntP évaluées et toutes ont pu générer une RSyntS. Globalement, **97,7 % des RSyntS** avaient la configuration attendue. Seules 11 structures sur les 500 évaluées étaient problématiques. Les résultats sont présentés dans le tableau 22 ci-dessous.

	Juge 1			**Juge 2**			
Évaluation	**oui**	**non**	**% succès**	**oui**	**non**	**% succès**	**κ Cohen**
Individuelle	98	2	98 %	—	—	—	—
Individuelle	—	—	—	97	3	97 %	—
Commune	295	5	98,3 %	294	6	98 %	0,91
Total	**393**	**7**	**97,5 %**	**391**	**9**	**97,8 %**	

Tableau 22 – Résultats de l'évaluation manuelle

De plus, nous obtenons un **κ de Cohen de 0,91** à la suite de l'évaluation commune de 300 structures. Cela indique un accord interjuge presque parfait ; seule une structure ne faisait pas consensus.

Les problèmes répertoriés proviennent d'un patron linguistique potentiellement trop vague (5), d'une faiblesse de notre implémentation (4), d'une erreur dans notre annotation (1) et d'une irrégularité dans l'annotation des patrons linguistiques (1). La section suivante traite de ces problèmes.

5.2.3 Discussion

L'évaluation de la précision de notre implémentation nous a permis de repérer plusieurs problèmes. D'abord, nous avons pu vérifier si des erreurs s'étaient glissées lors de l'attribution des codes de correspondance pour la mise en correspondance des patrons. Seulement une erreur du genre a été repérée.

Par la suite, nous avons pu identifier une faiblesse dans notre implémentation quant au **traitement des amalgames** (*du*, *des*, *aux*, etc.). Dans le script générant le dictionnaire lexical, nous n'avons pas pris en compte le caractère vague du mot-forme *des*. En effet, celui-ci peut à la fois représenter les parties du discours `PRÉP ART` comme dans ⌜DU BOUT DES DENTS⌝ (où *des*=*de*+*les*) ou simplement `ART` comme dans ⌜AVOIR DES NOUVELLES⌝ (où *des* est le pluriel d'UN). Le premier cas de figure étant plus courant, c'est celui que nous avons d'abord rencontré et nous n'offrons donc pas un traitement différentiel de ces homographes. Ainsi, lorsqu'il s'agit d'un article indéfini pluriel, nous séparons simplement ce dernier en DE et LES, ce qui introduit un nœud en trop dans nos structures ; cela nuit à la bonne mise en correspondance des patrons. Pour régler ce problème, nous devrions créer une fonction qui se penche spécifiquement sur les homographes *des* en incluant une condition qui tiendrait compte de la partie du discours des patrons linguistiques.

La majorité des erreurs que nous avons relevées découlent de la **mise en correspondance des patrons**. Bien que les patrons linguistiques soient censés ne représenter qu'un seul arbre syntaxique, certains décrivent des locutions qui comprennent un nombre différent de constituants. Or, il importe que ce nombre soit le même entre les locutions d'un même patron linguistique pour la mise en correspondance. Rappelons que celle-ci se base sur le nombre de constituants de la locution et leurs positions respectives dans l'arbre (voir section 4.1.1.2). Cependant, les patrons syntaxiques du RL-fr ne spécifient pas les parties du discours des locutions imbriquées dans d'autres locutions. Par exemple, ⌜MANGER LA FEUILLE DE MATCH⌝ est décrit par le patron `V Art LocN` au même titre que ⌜FAIRE LE JOLI CŒUR⌝. Cela pose problème, puisque la locution ⌜FEUILLE DE MATCH⌝ comporte un nœud de plus que ⌜JOLI CŒUR⌝.

De plus, l'absence d'information dans le RL-fr sur les **relations de dépendance** entre les unités syntaxiques de surface des locutions et leurs actants pose problème lors de la lexicalisation de surface. En effet, à défaut d'avoir accès au patron de régime de la locution, chaque actant est dépendant de la tête syntaxique de la locution. Or, cette arborisation n'est pas toujours celle attendue. Par exemple, deux actants sont associés à la locution ⌜CASSER DU SUCRE SUR LE DOS⌝ (*X casse du sucre sur le dos de Y*) ; cependant, le second actant ne dépend pas de la tête syntaxique CASSER, mais plutôt du nom DOS. De plus, le patron de régime par défaut de GenDR vient ajouter la

préposition DE entre le nom et son complément. Comme nos données ne comprenaient pas les patrons de régime et ne décrivaient pas systématiquement la place désirée des actants, nous avons exclu le traitement des actants des critères d'évaluation pour la précision de notre implémentation. Nous y reviendrons dans des travaux ultérieurs.

Un des apports de notre implémentation est justement de vérifier la précision d'une ressource lexicographique et de repérer ses failles. Nous sommes ainsi ravie d'avoir trouvé quelques pistes de réflexion à partager avec nos collègues du RL-fr. Revenons premièrement sur les problèmes rencontrés lors du traitement des locutions imbriquées. Notre traitement des locutions ne nous permet pas de générer le genre de structure proposé par Pausé (2017), illustré à la figure 64. Cette structure conserve la locution imbriquée comme un seul nœud, sans toutefois conserver l'accès aux constituants de celle-ci. Pour cela, il serait possible de créer une colonne spécifiant les parties du discours de la locution imbriquée dans l'encodage des données de Spiderlex. Cela empêcherait d'avoir à attribuer un nouveau patron aux locutions « poupées russes ».

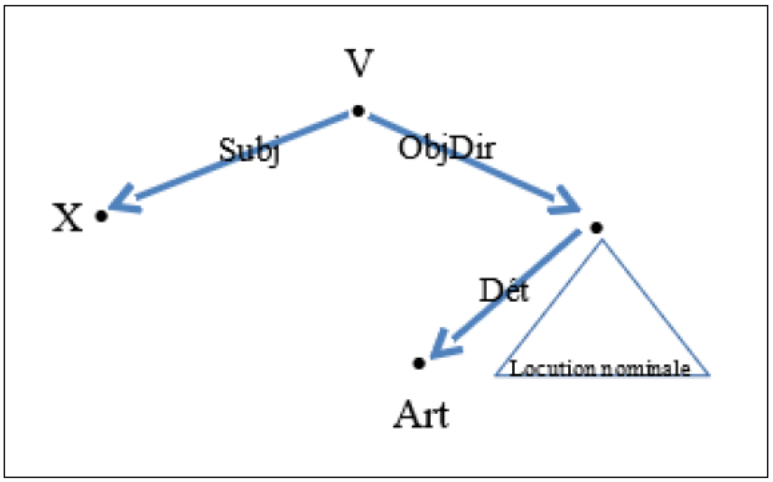


Figure 64 – RSyntS proposée pour le patron linéarisé V Art LocN (Pausé, 2017)

Deuxièmement, un des problèmes que nous avons rencontrés lors de notre traitement est la **flexion des nœuds** de certaines locutions ; par exemple, la locution ⌜ALLER AUX FRAISES⌝ contient deux nœuds fléchis (*aux* et *fraises*). Les nœuds en RSyntS ne sont pas des mots-formes ; ils sont plutôt des lexèmes accompagnés de traits grammaticaux spécifiant la flexion désirée. Cela permet de consolider l'information lexicale dans des entrées correspondant aux lexies (FRAISE) plutôt

qu'aux mots-formes (*fraises*). Or, bien qu'il soit possible de traiter rapidement la flexion des articles, la flexion des noms ou des verbes demanderait de repasser sur chacune des entrées des locutions. Il faudrait donc créer une colonne dans l'encodage des données de Spiderlex qui spécifie d'où vient cette flexion et à quel niveau de représentation celle-ci est ajoutée en TST.

Plusieurs problèmes rencontrés proviennent de nos toutes premières décisions quant à notre collecte des données du RL-fr. Si c'était à refaire, il faudrait apparier chaque unité des locutions avec ses parties du discours. Cela nous aurait permis de créer un script qui va chercher les parties du discours décrivant les locutions imbriquées. Cette solution nous permettrait de rapidement encoder la flexion des locutions sous forme de grammèmes et de la soustraire de la forme de citation des lexèmes en RSyntS.

Notre traitement des locutions gagnerait également en précision si les fiches lexicographiques des locutions du RL-fr décrivaient systématiquement les **patrons de régime**. Cela permettrait par exemple l'ajout de la bonne préposition régie gouvernée par la locution. De plus, ces fiches pourraient réunir des informations quant aux possibles coréférences entre les constituants de la locution et ses actants. Cela serait, entre autres, pertinent pour les locutions qui comportent un pronom déterminatif. Par exemple, la fiche pourrait décrire le lien de coréférence qui existe dans la locution ⌜AVALER SON CHAPEAU⌝ (`V ProDét NC`) entre son PRODÉT et son premier actant.

Finalement, notre évaluation nous a également permis de repérer certaines incohérences dans la classification des locutions. D'abord, l'utilisation de `Dét` pour décrire l'article dans le patron « `ProImpers LocV Dét NC (Prép_$1)` . » associé à la locution ⌜IL Y A DES LIMITES.⌝, alors que normalement `Art` est utilisé ailleurs. De plus, l'utilisation de QUELQUE CHOSE pour décrire un actant dans la locution ⌜SE METTRE QUELQUE CHOSE DANS LE CIBOULOT⌝ alors que cet ajout n'est pas présent dans les autres locutions décrites par le patron « `Vpro Prép Art NC ($2)` . », comme ⌜SE METTRE SOUS LA DENT⌝. Ailleurs dans le corpus, un `_N` est ajouté aux locutions, par exemple pour ⌜_N PEUT TOUJOURS COURIR.⌝ décrit par le patron `($1) V Adv V.`. Ce _N représente une position actancielle devant être remplie par un syntagme nominal ; celui-ci est représenté par une `($1)` dans le patron linguistique.

Conclusion

Dans le présent mémoire, nous avons donné un aperçu de la **GAT**, une branche du TAL centrée sur la production de textes qui se rapprochent le plus possible d'énoncés naturels. Nous avons aussi décrit quelques systèmes de génération à base de règles, dont certains se concentrent sur la dernière étape de la génération (réalisateurs de surface) et d'autres prennent en charge tous les modules de réalisation linguistique (réalisateurs profonds).

Par la suite, nous avons présenté le réalisateur profond multilingue dans lequel s'insère notre implémentation : **GenDR**. Celui-ci est basé sur la **TST** (Mel'čuk, 1997 ; Žolkovsky et Mel'čuk, 1967) et utilise la plateforme **MATE** (Bohnet et coll., 2000 ; Bohnet et Wanner, 2010), qui a été développée pour assurer la correspondance entre ses différents niveaux de représentation. GenDR se concentre sur l'**interface sémantique-syntaxe**, où ont lieu les processus d'arborisation et de lexicalisation. Il exécute six types de lexicalisation : la lexicalisation simple, grammaticale, liée, par classe, de secours et par patron.

Notre contribution à ce réalisateur réside dans un traitement systématique de la **lexicalisation par patron**. Nous avons bonifié GenDR en créant un dictionnaire et des règles de transduction propres aux locutions. Pour ce faire, nous nous sommes basée sur les données lexicales du **RL-fr** ; celui-ci offre une **classification grammaticale des locutions**, qui sont regroupées selon la suite de parties de discours de leurs constituants, en suivant l'approche de Pausé (2017). Elles forment ainsi des **patrons syntaxiques linéarisés**, que nous appelons *patrons linguistiques*. Cela nous permet de les différencier des *patrons génériques* que nous proposons. Ces derniers sont des arbres à trous qui englobent toutes les **représentations syntaxiques de surface** (RSyntS) possibles de deux à six nœuds. Nous les qualifions de *génériques* comme ils ne dépendent d'aucune langue. Leur création découle du parallèle entre la récursivité intrinsèque des structures arborescentes et la **partition des nombres entiers** (Andrews, 1998).

Nous avons donc **généré automatiquement** une grammaire de 36 **règles de lexicalisation** par patron. Celles-ci s'exécutent en utilisant le **dictionnaire lexical** que nous avons aussi créé automatiquement. Les informations qu'il contient sont encodées de sorte que les locutions

héritent des traits des patrons linguistiques qui, eux-mêmes, héritent des traits de leur partie du discours. Cela nous a demandé d'attribuer manuellement un patron générique à chacun des 514 patrons linguistiques et de spécifier la place dans l'arbre de chacun de ses constituants.

Nous offrons ainsi une **couverture** de 2 846 locutions, soit 97,5 % des 2 919 locutions répertoriées par le RL-fr. La précision de cette implémentation a été confirmée à l'aide d'une évaluation humaine d'un échantillon de 500 locutions (18 % des locutions du RL-fr). La structure de ces locutions a été analysée par deux linguistes afin de vérifier si elle correspondait à la structure attendue, ce qui était le cas dans 97,65 % des cas (avec un accord interjuge de 0,91).

Notre implémentation se limite aux arbres de six nœuds ou moins, ce qui permet de lexicaliser un grand nombre de locutions avec un nombre restreint de règles. Par ailleurs, notre travail peut servir de base pour un traitement ultérieur de locutions plus longues. À ce sujet, nous avons proposé de reconfigurer les locutions de plus de six nœuds en se basant sur le caractère récursif des patrons génériques, qui facilitent leur division en sous-arbres.

Comme nos patrons génériques ont été conçus dans une optique multilingue, il serait intéressant d'élargir la couverture des locutions à d'autres langues, notamment les branches de l'anglais et du mandarin de GenDR.

Pour conclure, il serait également intéressant d'intégrer la lexicalisation par patron au tronc principal du réalisateur, qui comprend la lexicalisation liée des collocations (Lambrey, 2016 ; Lambrey et Lareau, 2015) et la lexicalisation grammaticale des patrons de régimes (Galarreta-Piquette, 2018).

Références bibliographiques

Agel, V., Eichinger, L. M., Eroms, H. W., Heringer, H.-J., et Hellwig, P. (2003). *Dependenz und valenz/dependency and valency: Ein internationales handbuch der zeitgenossischen forschung/an international handbook of contemporary research* (Vol. 1). Walter de Gruyter.

Andrews, G. E. (1998). *The theory of partitions*. Cambridge university press.

Apresi͡an, I. D., Apresân, Û. D., et Apresjan, J. (2000). *Systematic lexicography*. Oxford University Press.

Apresjan, J. D., Boguslavsky, I. M., Iomdin, L. L. et Tsinman, L. L. (2007). Lexical functions in actual NLP-applications. Dans L. Wanner (dir.), *Selected lexical and grammatical issues in the Meaning Text Theory: In honour of Igor Mel'čuk*. John Benjamin Publishing.

Baroni, M., Bernardini, S., Ferraresi, A., et Zanchetta, E. (2009). The WaCky Wide Web: A Collection of Very Large Linguistically Processed Web-Crawled Corpora. *Language Resources and Evaluation*, 43(3), 209–226. https://doi.org/DOI:10.1007/s10579-009-9081-4

Bateman, J. A. (1997). Enabling technology for multilingual natural language generation: The KPML development environment. *Natural Language Engineering*, 1(1), 1–42.

Bateman, J. A., Kruijff-Korbayová, I. et Kruijff, G.-J. (2005). Multilingual Resource Sharing Across Both Related and Unrelated Languages: An Implemented, Open-Source Framework for Practical Natural Language Generation*. Research on Language and Computation*, 3(2), 191-219. https://doi.org/10.1007/s11168-005-1298-9

Bateman, J. A., Matthiessen, C., Nanri, K., et Zeng, L. (1991). The re-use of linguistic resources across languages in multilingual generation components. *IJCAI'91: Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 2, 966–971.

Bohnet, B., Langjahr, A., et Wanner, L. (2000). A development Environment for an MTT-Based Sentence Generator. *INLG'2000 Proceedings of the First International Conference on Natural Language Generation*, 14, Mitzpe Ramon, 260–263. https://doi.org/10.3115/1118253.1118292

Bohnet, B., et Wanner, L. (2010). Open Soucre Graph Transducer Interpreter and Grammar Development Environment. *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*. http://www.lrecconf.org/proceedings/lrec2010/pdf/585_Paper.pdf

Bollmann, M. (2011). Adapting SimpleNLG to German. *Proceedings of the 13th European Workshop on Natural Language Generation*, 133–38. Nancy. https://www.aclweb.org/anthology/W11-2817.

Braun, D., Klimt, K., Schneider, D., et Matthes, F. (2019). SimpleNLG-DE : Adapting SimpleNLG 4 to German. *Proceedings of the 12th International Conference on Natural Language Generation*. Tokyo, 415–420. https://doi.org/10.18653/v1/W19-8651

Cascallar-Fuentes, A., Ramos-Soto, A., et Bugarín Diz, A. (2018). Adapting SimpleNLG to Galician language. *Proceedings of the 11th International Conference on Natural Language Generation*, Tilburg, 67–72. https://doi.org/10.18653/v1/W18-6507

Chen, G., van Deemter, K., et Lin, C. (2018). SimpleNLG-ZH: A Linguistic Realisation Engine for Mandarin. *Proceedings of the 11th International Conference on Natural Language Generation*, Tilburg, 57–66. https://doi.org/10.18653/v1/W18-6506

Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1), 37-46. https://doi.org/10.1177/001316446002000104

Corriveau, H. (2021). *De FrameNet à la Théorie Sens-Texte : Conversion et correspondance* [mémoire de maîtrise, Université de Montréal].

Danlos, L. (1983). Présentation d'un modèle de génération automatique. *Revue québécoise de linguistique*, 13 (1), 203-228. https://doi.org/10.7202/602510ar

Danlos, L. (1987). The linguistic basis of text generation. *Proceedings of the third conference on European chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 1. https://doi.org/10.3115/976858.976859

Danlos, L. (1998). G-TAG : Un formalisme lexicalisé pour la génération de textes inspiré de TAG. *Traitement Automatique Des Langues*, 39 (2), 28 p.

Danlos, L. (2000). G-TAG : A Lexicalized Formalism for Text Generation inspired by Tree Adjoining Grammar. Dans A. Abeille et O. Rambow (dir.), *Tree Adjoining Grammars : Formalisms, Linguistic Analysis, and Processing*. CSLI Publications.

Danlos, L., Maskharashvili, A., et Pogodalla, S. (2014). Génération de textes : G-TAG revisité avec les Grammaires Catégorielles Abstraites. *Proceedings of TALN 2014 (long papers),* Vol.1, Marseille, 161–172. https://www.aclweb.org/anthology/F14-1015

de Jong, R., et Theune, M. (2018). Going Dutch: Creating SimpleNLG-NL. *Proceedings of the 11th International Conference on Natural Language Generation*, Tilburg, 73-78. https://doi.org/10.18653/v1/W18-6508

de Oliveira, R., et Sripada, S. (2014). Adapting SimpleNLG for Brazilian Portuguese realisation. *Proceedings of the 8th International Natural Language Generation Conference (INLG)*, Philadelphia, 93–94. https://doi.org/10.3115/v1/W14-4412

Dubinskaite, I. (2017). *Développement de ressources lituaniennes pour un générateur automatique de texte multilingue* [mémoire de master 2, Université Grenoble Alpes].

Elhadad, M., et Robin, J. (1996). An Overview of SURGE: A Reusable Comprehensive Syntactic Realization Component. *Eighth International Natural Language Generation Workshop.* https://www.aclweb.org/anthology/W96-0501

Elhadad, M. et Robin, J. (1997). SURGE: a Comprehensive Plug-in Syntactic Realization Component for Text Generation. *Computational Linguistics*, *99*(4).

Fonseca, A., Sadat, F. et Lareau, F. (2016). A Lexical Ontology to Represent Lexical Functions. *Proceedings of the Workshop on Cognitive Aspects of the Lexicon*, LREC, Japan, 145–155.

Gader, N., Koehl, A., et Polguère, A. (2014). A Lexical Network with a Morphological Model in It. *Proceedings of the 4th Workshop on Cognitive Aspects of the Lexicon* (CogALex), Dublin, 154-165. https://doi.org/10.3115/v1/W14-4720

Gader N., Ollinger S. et Polguère A. (2014). One Lexicon, Two Structures: So What Gives? *Proceedings of the Seventh Global Wordnet Conference* (GWC2014), Tartu, 163–171. http://www.aclweb.org/anthology/W14-0122

Galarreta-Piquette, D. (2018). *Intégration de VerbNet dans un réalisateur profond* [mémoire de maîtrise, Université de Montréal]. https://papyrus.bib.umontreal.ca/xmlui/bitstream/handle/1866/21112/

Gatt, A., et Krahmer, E. (2018). Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, 61(1), 65–170. https://doi.org/10.1613/jair.5477

Gatt, A., et Reiter, E. (2009). SimpleNLG: A Realisation Engine for Practical Applications. *Proceedings of the 12th European Workshop on Natural Language Generation* (ENLG 2009), Athens, 90–93. https://www.aclweb.org/anthology/W09-0613

Gougenheim, G., Michéa, R., Rivenc P. et Sauvageot, A. (1967). *L'élaboration du français fondamental (1er degré) : étude sur l'établissement d'un vocabulaire et d'une grammaire de base*. Didier.

Gross, G. (1996). *Les expressions figées en français : noms composés et autres locutions*. Editions Ophrys.

Hamalainen, M. (2018). Poem Machine - a Co-creative NLG Web Application for Poem Writing. *Proceedings of The 11th International Natural Language Generation Conference*, Stroudsburg, 195-196.

He, L. (2020). *Un dictionnaire de régimes verbaux en mandarin* [mémoire de maîtrise, Université de Montréal]. https://papyrus.bib.umontreal.ca/xmlui/handle/1866/25471

Iordanskaja, L., et Mel'čuk, I. A. (2017). *Le mot français dans le lexique et dans la phrase*. Hermann.

Jousse, A.-L. (2010). *Modèle de structuration des relations lexicales fondé sur le formalisme des fonctions lexicales* [thèse de doctorat, Université de Montréal]. https://papyrus.bib.umontreal.ca/xmlui/handle/1866/4347

Kahane, S. (2003). The Meaning-Text Theory. Dans V. Agel, P. Hellwig, H.-J. Heringer, L. M. Eichinger et H. W. Eroms (dir.), Dependenz und valenz/dependency and valency: Ein *internationales handbuch der zeitgenossischen forschung/an international handbook of contemporary research* (vol. 1, p. 32). Walter de Gruyter.

Kahane, S., et Polguère, A. (2001). *Formal foundation of lexical functions. Collocation: Computational Extraction*, Analysis and Exploitation, Toulouse, 8–15.

Kasper, R. T. (1989). A Flexible Interface for Linking Applications to Penman's Sentence Generator. *Speech and Natural Language: Proceedings of a Workshop Held at Philadelphia*, 153–158. https://www.aclweb.org/anthology/H89-1022

Kuanzhuo, Z., Lin, L., et Weina, Z. (2020). SimpleNLG-TI: Adapting SimpleNLG to Tibetan. *Proceedings of the 13th International Conference on Natural Language Generation*, Dublin, 86–90. https://www.aclweb.org/anthology/2020.inlg-1.12

Lambrey, F. (2016). *Implémentation des collocations pour la réalisation de texte multilingue* [mémoire de maîtrise, Université de Montréal]. https://papyrus.bib.umontreal.ca/xmlui/bitstream/handle/1866/18769/

Lambrey, F., et Lareau, F. (2015). Le traitement des collocations en génération de texte multilingue. *Actes de La 22e Conférence Sur Le Traitement Automatique Des Langues Naturelles*, Caen, 263–269.

Lapalme, G. (2020). The jsRealB Text Realizer: Organization and Use Cases. *arXiv.* http://arxiv.org/abs/2012.15425

Lareau, F., Dras, M., et Dale, R. (2011). Detecting interesting event sequences for sports reporting. *Proceedings of the National Conference on Artificial Intelligence*, Nancy, 200–205.

Lareau, F. et Lambrey, F. (2016). *GÉCO v1.0 : User Manual. Université de Montréal*, Département de linguistique et de traduction. [document technique non publié]

Lareau, F., Lambrey, F., Dubinskaite, I., Galarreta-Piquette, D., et Nejat, M. (2018). GenDR : A Generic Deep Realizer with Complex Lexicalization. *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*, Miyazaki, 3018–3025. https://www.aclweb.org/anthology/L18-1478

Lareau, F. et Wanner, L. (2007). Towards a generic multilingual dependency grammar for text generation. *Proceedings of the GEAF07 Workshop.* CSLI Publications. http://csli-publications.stanford.edu/GEAF/2007

Lavoie, B., et Rambow, O. (1997). A Fast and Portable Realizer for Text Generation Systems. *Fifth Conference on Applied Natural Language Processing*, Washington, 265–268. https://doi.org/10.3115/974557.974596

Mann, W. C. (1983). An overview of the PENMAN text generation system. *Proceedings of the National Conference on Artificial Intelligence*, 261–265.

Mann, W. C., et Matthiessen, C. M. I. M. (1982). *Two Discourse Generators. A Grammar and a Lexicon for a Text-Production System.* (No. ADA126353; p. 30). University of Southern California. https://apps.dtic.mil/sti/citations/ADA126353

Mann, W. C., et Matthiessen, C. M. I. M. (1983). *Nigel: A Systemic Grammar for Text Generation*. (p. 80). University of Southern California. https://apps.dtic.mil/sti/citations/ADA125253

Matthiessen, C., et Halliday, M. (1997). *Systemic functional grammar: A first step into the theory*.

Mazzei, A., Battaglino, C., et Bosco, C. (2016). SimpleNLG-IT: adapting SimpleNLG to Italian. *Proceedings of The 9th International Natural Language Generation Conference*, Edinburgh, 184–192.

Mel'čuk, I. (1988). *Dependency Syntax: Theory and Practice*. SUNY Press.

Mel'čuk, I. (1995). The future of the lexicon in linguistic description and the explanatory combinatorial dictionary. *Linguistics in the morning calm*, 3, 181-270.

Mel'čuk, I. (1996). Lexical functions: a tool for the description of lexical relations in a lexicon. *Lexical functions in lexicography and natural language processing*, 31, 37-102.

Mel'čuk, I. (1997). *Vers une linguistique Sens-Texte* [Leçon inaugurale]. Collège de France, Paris.

Mel'čuk, I. (1998). Collocations and lexical functions. Dans A. P. Cowie (Dir.), *Phraseology. Theory, analysis, and applications* (p. 23–53). Clarendon Press.

Mel'čuk, I. (2004). La non-compositionnalité en morphologie linguistique. *Verbum*, 26 (4), 439–458.

Mel'čuk, I. (2007). Lexical Functions. Dans H. Burger, D. Dimitri, P. Kuhn, et N. Neal (Dir.), *Phraseologie/Phraseology. Ein internationales Handbuch zeitgenössi_scher Forschung/An international Handbook of Contemporary Research*, (p. 119–131). De Gruyter.

Mel'čuk, I. (2012). *Semantics. From meaning to text* (Vol. 1). John Benjamins Publishing Company.

Mel'čuk, I. (2013). Tout ce que nous voulions savoir sur les phrasèmes, mais …. *Cahiers de lexicologie*, 102, 129–149.

Mel'čuk, I. (2014). *Semantics*. John Benjamins Publishing Company. https://benjamins.com/catalog/slcs.168

Mel'čuk, I. (2015). Clichés, an Understudied Subclass of Phrasemes. *Yearbook of Phraseology*, 6(1), 55-86. https://doi.org/10.1515/phras-2015-0005

Mel'čuk, I. Arbatchewsky-Jumarie, N., Dagenais, L., Elnitsky, L., Iordanskaja, L., Lefebvre, M.-N. et Mantha, S. (1988). *Dictionnaire explicatif et combinatoire du français contemporain : recherches lexico-sémantiques* (Vol. II). Presses de l'Université de Montréal.

Mel'čuk, I., Arbatchewsky-Jumarie, N., Elnitsky, L., Iordanskaja, L. et Lessard, A. (1984). *Dictionnaire explicatif et combinatoire du français contemporain : recherches lexico-sémantiques* (Vol. I). Presses de l'Université de Montréal.

Mel'čuk, I. Arbatchewsky-Jumarie, N., Iordanskaja, L. et Mantha, S. (1992). *Dictionnaire explicatif et combinatoire du français contemporain : recherches lexico-sémantiques* (Vol. III). Presses de l'Université de Montréal.

Mel'čuk, I. Arbatchewsky-Jumarie, N., Iordanskaja, L., Mantha, S. et Polguère, A. (1999). *Dictionnaire explicatif et combinatoire du français contemporain : recherches lexico-sémantiques* (Vol. IV). Presses de l'Université de Montréal.

Mel'čuk, I. et Polguère, A. (2007). *Lexique actif du français*. De Boeck Supérieur.

Mel'čuk, I. et Polguère, A. (2021). Les fonctions lexicales dernier cri. Dans S. Marengo (dir.), *La théorie Sens-Texte. Concepts-clés et applications* (p. 75-155). L'Harmattan. https://hal.archives-ouvertes.fr/hal-03311348

Meunier, F., et Danlos, L. (1998). *System Demonstration FLAUBERT: An User Friendly System for Multilingual Text Generation*. 284–287. https://www.aclweb.org/anthology/W98-1431

Milićević, J. (2006). A Short Guide to the Meaning-Text Linguistic Theory. *Journal of Koralex*, 8, 187–233.

Mille, S., Carlini, R., Burga, A., et Wanner, L. (2017). FORGe at SemEval-2017 Task 9: Deep sentence generation based on a sequence of graph transducers. *Proceedings of the 11th International Workshop on Semantic Evaluation* (SemEval-2017), Vancouver, 920–923. https://doi.org/10.18653/v1/S17-2158

Mille, S., Dasiopoulou, S., et Wanner, L. (2019). A Portable Grammar-based NLG System for Verbalization of Structured Data. *SAC '19: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 1054–1056. https://doi.org/10.1145/3297280.3297571

Mille, S., et Wanner, L. (2017). A demo of FORGe: The Pompeu Fabra Open Rule-based Generator. *Proceedings of the 10th International Conference on Natural Language Generation*, Santiago de Compostela, 245–246. https://doi.org/10.18653/v1/W17-3539

Molins, P. (2014). *JSrealB : Approche systématique pour la réalisation multilingue de textes*. INSA de Lyon. http://rali.iro.umontreal.ca/

Molins, P., et Lapalme, G. (2015). JSrealB: A bilingual text realizer for web programming. *Proceedings of the 15th European Workshop on Natural Language Generation* (ENLG), 109–111.

Ollinger S. et Polguère A. (2020). *Guide de navigation Spiderlex* (version du 19 mai 2020). Rapport technique, ATILF CNRS, Nancy https://lexical-systems.atilf.fr/wp-content/uploads/2020/05/Spiderlex-doc-FR-19v20.pdf

Ollinger, S., Polguère, A., Chudy, Y., & Gaume, B. (2020). Spiderlex et compagnie (Spiderlex & Co*). Actes de la 6e conférence conjointe Journées d'Études sur la Parole* (JEP, 33e édition), Traitement Automatique des Langues Naturelles (TALN, 27e édition), Rencontre des Étudiants Chercheurs en Informatique pour le Traitement Automatique des Langues (RÉCITAL, 22e édition). Volume 4 : Démonstrations et résumés d'articles internationaux (p. 60-63).

Pausé, M.-S. (2014). *Modélisation de la structure lexico-syntaxique des locutions : Approche lexicographique* [mémoire de master, Université de Lorraine].

Pausé, M.-S. (2017). *Structure lexico-syntaxique des locutions du français et incidence sur leur combinatoire* [thèse de doctorat, Université de Lorraine].

Polguère, A. (1990). *Structuration et mise en jeu procédurale d'un modèle linguistique déclaratif dans un cadre de génération de texte* [thèse de doctorat, Université de Montréal.]

Polguère, A. (1998). La théorie Sens-Texte. *Dialangue*, Vol. 8-9, Université du Québec à Chicoutimi, p. 9-30.

Polguère, A. (2007). Lexical function standardness. Dans L. Wanner (Dir.), *Selected lexical and grammatical issues in the Meaning Text Theory: In honour of Igor Mel'čuk*. John Benjamins Publishing.

Polguère A. (2009). Lexical systems: graph models of natural language lexicons. *Language Resources and Evaluation*, 43(1), p. 41–55.

Polguère A. (2014). From Writing Dictionaries to Weaving Lexical Networks. *International Journal of Lexicography*, 27(4), p. 396–418.

Polguère, A. (2015). Non-compositionnalité : Ce sont toujours les locutions faibles qui trinquent. *Verbum*, 37 (2), 257–280.

Polguère, A. (2016). *Lexicologie et sémantique lexicale : Notions fondamentales*. (troisième édition). Presses de l'Université de Montréal. https://www.pum.umontreal.ca/catalogue/lexicologie_et_semantique_lexicale_3e_ed

Portenseigne, C. (à paraître). *L'implémentation des relatives dans un réalisateur profond* [mémoire de maîtrise, Université de Montréal].

Ramos-Soto, A., Janeiro-Gallardo, J., et Bugarín Diz, A. (2017). Adapting SimpleNLG to Spanish. *Proceedings of the 10th International Conference on Natural Language Generation*, Santiago de Compostela, 144–148. https://doi.org/10.18653/v1/W17-3521

Reiter, E. et Dale, R. (1997). Building applied natural language generation systems. *Natural Language Engineering*, *3*(1), 57-87. https://doi.org/10.1017/S1351324997001502

Reiter, E., et Dale, R. (2000). *Building natural language generation systems*. Cambridge University Press.

Sripada, G., Burnett, N., Arria, R. T. Ross, Mastin, J. et Evans, E. (2014). A Case Study: NLG meeting Weather Industry Demand for Quality and Quantity of Textual Weather Forecasts. *Proceedings of the 8th International Natural Language Generation Conference*, 1-5.

Steinlin, J. (2003). *Générer des collocations [mémoire pour l'obtention du DEA*, Université de Paris VII - Denis Diderot].

Ters, F., Mayers G. et Reichenbach, D. (1988). *L'échelle Dubois-Buyse*, OCDL.

Ters, F., Mayer, G., & Reichenbach, D. (1995). L'Echelle Dubois-Buyse. *Editions MDI*.

Ters, F., Reichenbach, D., Mayer, G., & Mayer, G. (1970). L'échelle Dubois-Buyse d'orthographe usuelle française. H. Messeiller.

Tesnière, L. (1959). *Éléments de syntaxe structurale*. Klincksieck.

Vaudry, P. -L., et Lapalme, G. (2013). Adapting SimpleNLG for Bilingual English-French Realisation. *Proceedings of the 14th European Workshop on Natural Language Generation*, Sofia, 183–187. https://www.aclweb.org/anthology/W13-2125

Wanner, L. (1996). *Lexical Functions in Lexicography and Natural Language Processing*. John Benjamins Publishing.

Wanner, L., Bohnet, B., Bouayad-Agha, N., Lareau, F., Lohmeyer, A., et Nicklaß, D. (2007). On the Challenge of Creating and Communicating Air Quality Information. *Environmental Software Systems*, 7, 356–367. https://doi.org/10.13140/2.1.3773.6001

Wanner, L., Bohnet, B., Bouayad-Agha, N., Lareau, F. et Nicklaß, D. (2010). MARQUIS: Generation of user-tailored multilingual air quality bulletins. *Applied Artificial Intelligence*, *24*(10), 914-952. https://doi.org/10.1080/08839514.2010.529258

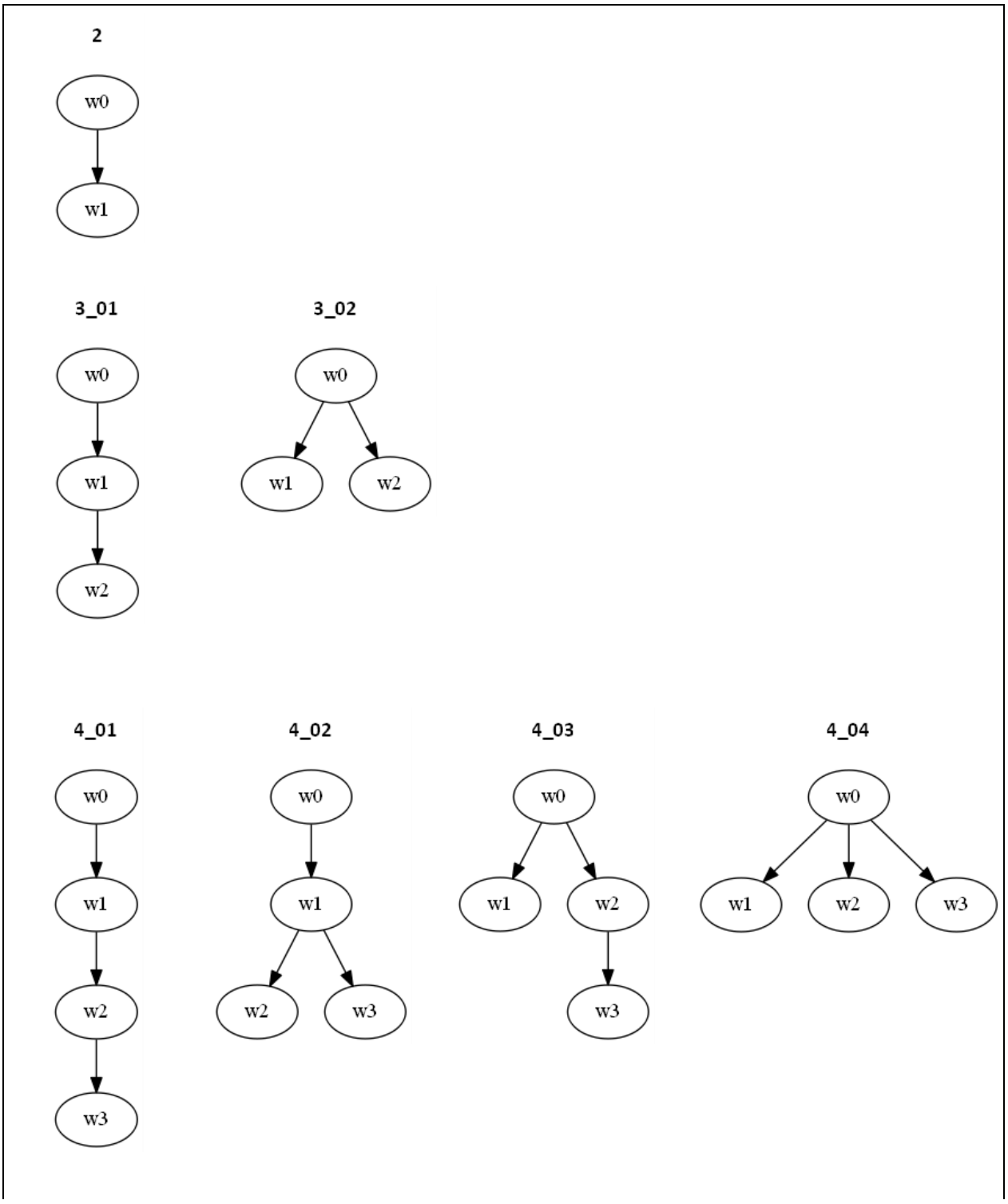
Wanner, L. et Lareau, F. (2009). *Applying the Meaning-Text Theory Model to Text Synthesis with Low- and Middle-Density Languages in Mind*. IOS Press.

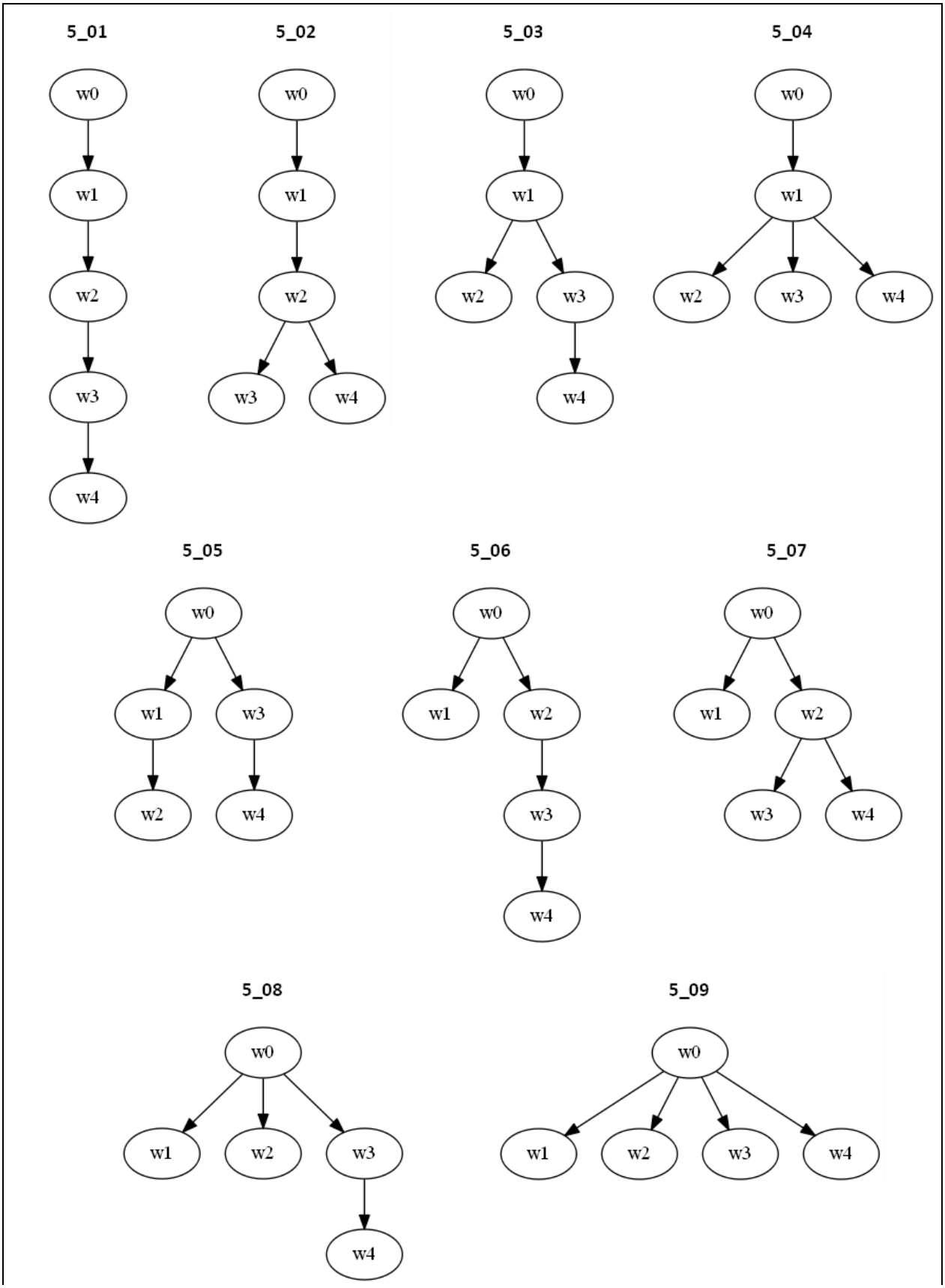
Wanner, L., Nicklaß, D., Bouayad-Agha, N., Bohnet, B., Bronder, J., Ferreira, F., Friedrich, R., Karppinen, A., Lareau, F., Lohmeyer, A., Panighi, A., Parisio, S., Scheu-Hachtel, H., et Serpa, J. (2007). From Measurement Data to Environmental Information: MARQUIS - A Multimodal AiR QUality Information Service for the General Public. *Environmental Software Systems*, 7, 43–53. https://doi.org/10.13140/2.1.3773.6001

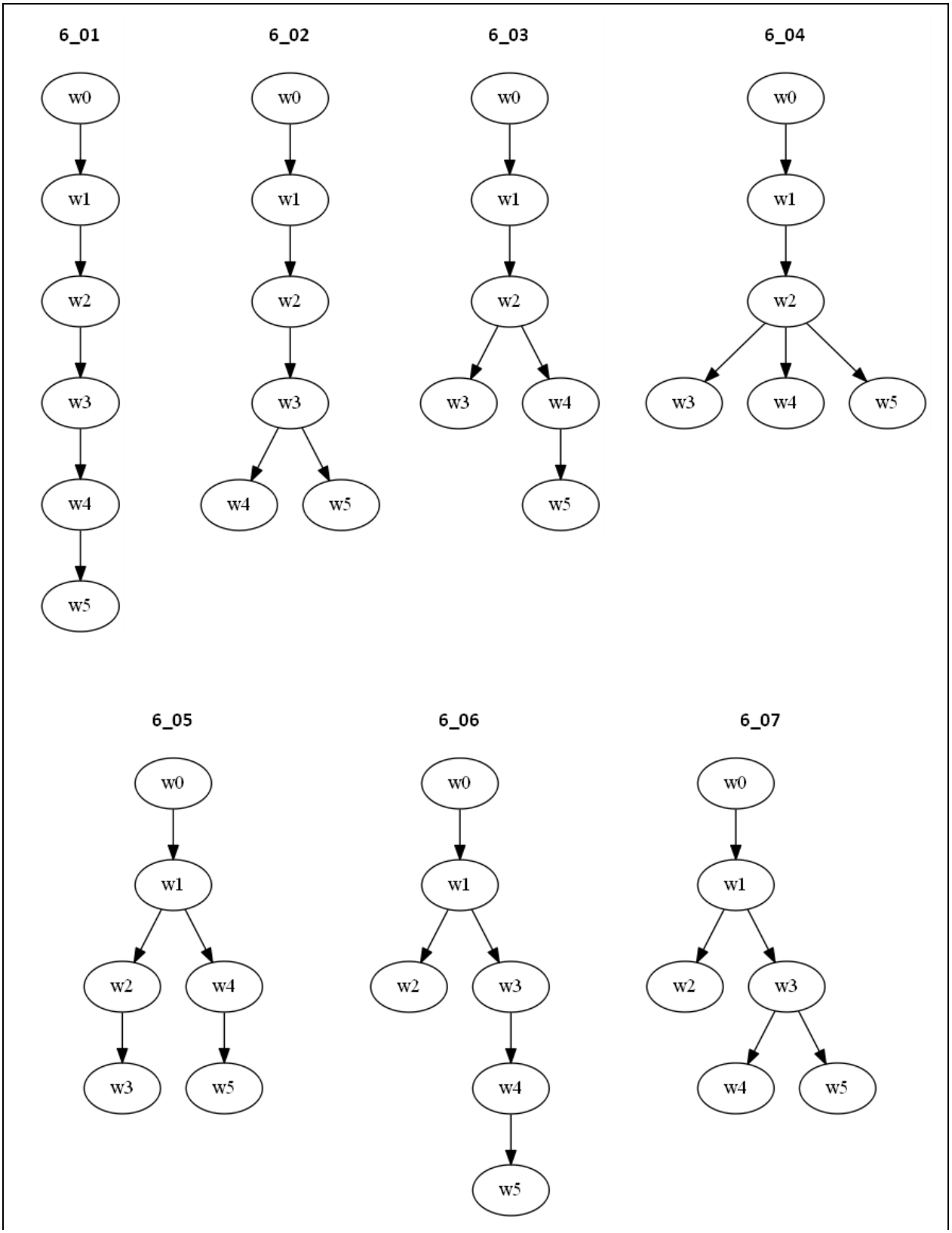
Zhao, X. (2018). Les collocations du champ sémantique des émotions en mandarin [mémoire de maîtrise, Université de Montréal]. https://papyrus.bib.umontreal.ca/xmlui/handle/1866/22133/

Žolkovsky, A. et Mel'čuk, I. (1967). O semanticeskom sinteze [Sur la synthèse sémantique]. Problemy kibernetiki, 19, 177-238.

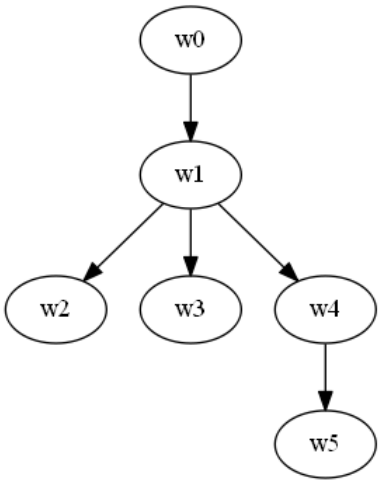
Annexe 1 : patrons génériques



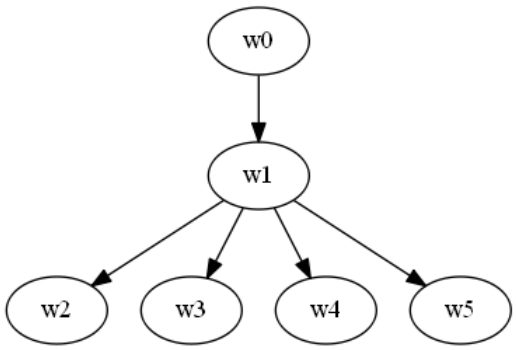

5_01
w0
w1
w2
w3
w4
5_02
w0
w1
w2
w3
w4
5_03
w0
w1
w2
w3
w4
5_04
w0
w1
w2
w3
w4
5_05
w0
w1
w3
w2
w4
5_06
w0
w1
w2
w3
w4
5_07
w0
w1
w2
w3
w4
5_08
w0
w1
w2
w3
w4
5_09
w0
w1
w2
w3
w4


6_01
w0
w1
w2
w3
w4
w5
6_02
w0
w1
w2
w3
w4
w5
6_03
w0
w1
w2
w3
w4
w5
6_04
w0
w1
w2
w3
w4
w5
6_05
w0
w1
w2
w4
w3
w5
6_06
w0
w1
w2
w3
w4
w5
6_07
w0
w1
w2
w3
w4
w5

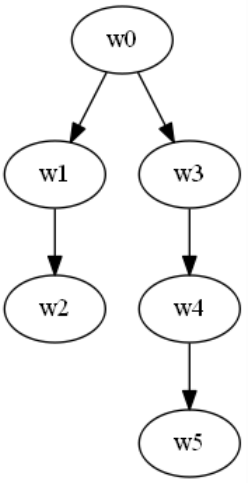
6_08



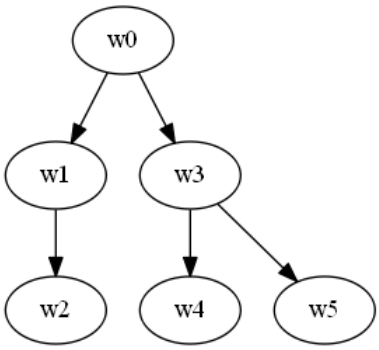
6_09



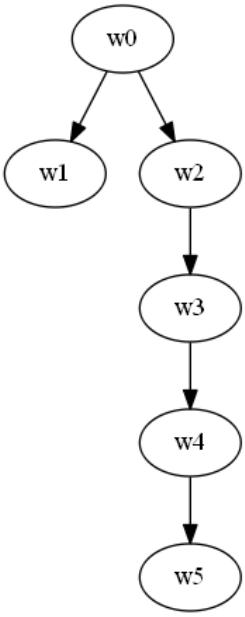
6_10



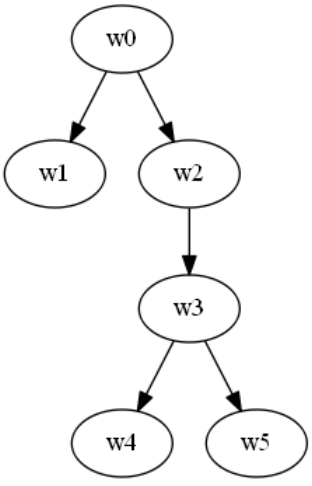
6_11



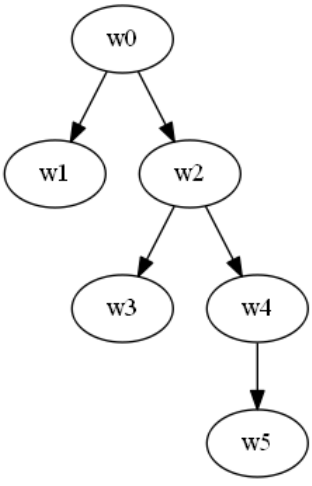
6_12



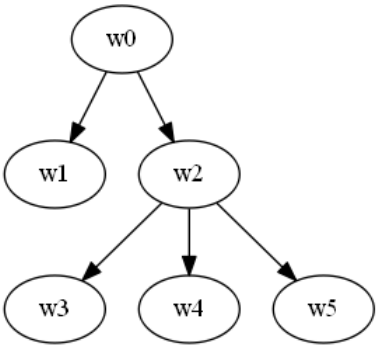
6_13



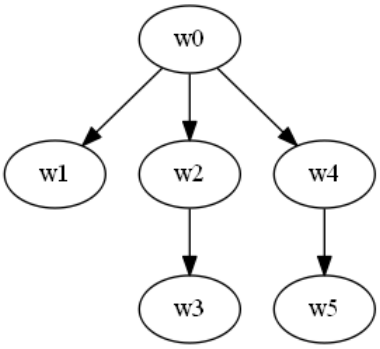
6_14



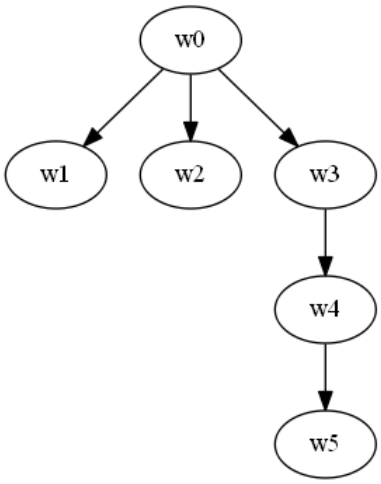
6_15



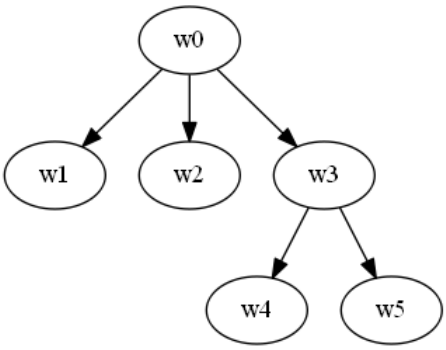
6_16



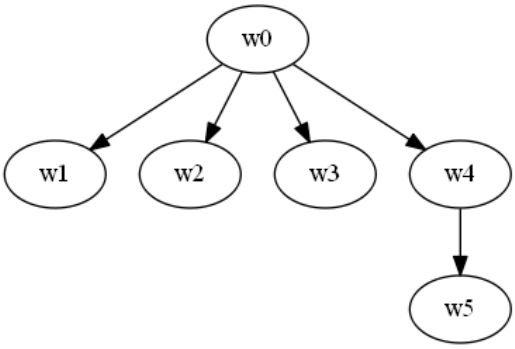
6_17



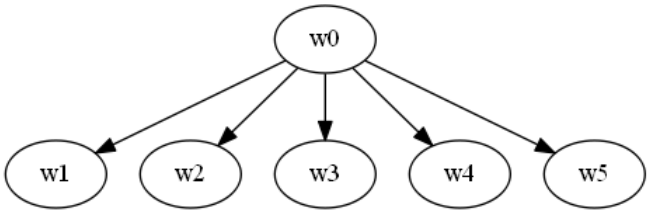
6_18



6_19



6_20



Annexe 2 : protocole de l'évaluation finale

Toujours faire «svn up» avant de débuter.

Pour chaque locution à tester :

1. Ouvrir le document d'annotation et modifier le type par défaut de la colonne map (type=str)
2. Bien regarder le nom de la locution, son patron linguistique et son tree_id
   a. lorsqu'une locution fait partie du patron linguistique (par ex. LocN, LocV, LocPrép) :
      - rechercher la locution imbriquée dans idioms_table.tsv pour connaître son patron linguistique
3. Juger si la structure est bonne
   a. Si la structure n'est pas la bonne :
      - noter tous les problèmes dans les colonnes appropriées
   b. Si le problème n'est pas déjà répertorié :
      - créer une étiquette et bien la définir dans ce même document
   c. L'évaluation est «pass/fail», il n'y a pas d'entre-deux
4. Noter si les actants sont liés ou non et combien de nœuds sont fléchis
   - cette partie n'influence par le statut «pass/fail»
   - cette partie sera utile pour partager nos résultats avec le RL-fr
5. Une fois l'évaluation terminée, bien enregistrer le fichier et faire un «svn ci -m»
   - il faut enregistrer la colonne map en format type=str
   - il ne faut pas laisser de «?» derrière et trancher
   - il ne faut pas connaître les résultats des autres juges avant la fin de l'évaluation

Description du document d'annotation :

#Dsynt : numéro de structure

std_name : nom de la locution

pat : nom du patron linguistique

tree_id : numéro du patron générique

map : mapping ou code de correspondance des patrons

Pass/fail: entrer 1 pour pass et 0 pour fail

- l'évaluateur indique la nature du problème dans une ou plusieurs des colonnes suivantes
- la flexion, la ponctuation et les actants n'influencent pas cette décision

tree choice : entrer le bon tree_id

- après consultation du patron générique (voir images graphviz)
- l'évaluateur juge que ce n'est pas le bon arbre pour cette locution
- regarder également le patron linguistique pour l'annotation : si ce choix d'arbre est bon que pour certaines locutions, notez «RL-fr» dans la colonne autre, car le problème vient probablement de leurs patrons et non de l'implémentation

tree applied : entrer 1

- si la colonne tree_id ne correspond pas à l'arbre généré
- après consultation du patron générique (voir images graphviz)

map choice : entrer le bon mapping

- après consultation du mapping (voir chapitre du mémoire sur implémentation)
- l'évaluateur juge que les nœuds de l'arbre de sont pas dans le bon ordre

map applied : entrer 1

- si les nœuds ne sont pas dans l'ordre indiqué en colonne map

absent node : entrer nombre de nœuds manquants

Loc in pat: entrer 1

- si le pat n'a pas été adapté au nombre de nœuds en Ssynt
- chaque unité syntaxique devrait avoir son nœud (e.g., LocN.1 LocN.2 LocN.3)

hyphen: entrer 1

- s'il y a un problème lié aux traits d'union
- consulter la colonne pat pour connaître la POS du mot comprenant un trait d'union

amalgame : entrer nombre d'amalgames

- si le problème est lié aux amalgames (absents en Ssynt)
- parfois ils sont représentés par la POS PhNC

Vpro : entrer 1

- si le problème est lié aux verbes pronominaux ou aux pronoms réfléchis
- le pronom réfléchi est séparé du verbe pronominal en Ssynt

AdvNeg : entrer le nombre de nœuds manquants

- s'il y a un problème avec les adverbes de négation
- consulter «General Inventory of Surface-Syntactic Relations in World Languages» de Mel'čuk (2019/01/20) et Iordanskaja et I. Mel'čuk (2009) pour les relations Ssynt

spacing: entrer 1

- s'il y a un problème lié aux espaces insécables (par ex. 10 000 mètres)

other: entrer une étiquette pour définir le problème

- faire une liste au fur et à mesure de ces étiquettes et de leur définition
- penser à regarder les majuscules et le noter si cela pourrait être exclu de la structure
- les noms propres prennent toujours les majuscules

actants : actants tels que décrits dans Spiderlex

actants_eval : entrer nombre d'actants liés

- seulement entrer un nombre s'il y a des actants décrits dans la colonne précédente
- il est normal que des nœuds soient ajoutés pour lier les actants (e.g. de)

flexion : entrer le nombre de nœuds fléchis

- même si le nœud comprend plusieurs marques de flexions, ne le compter qu'une seule fois
- ex de flexion : PL nom, PL/fém pronom, PL/Fém adj, PL/Fém dét, V fléchis
- faire attention aux pronoms (par ex. : la, elle, cet, leur...)
- un nœud qui est accordé au fém et au PL compte tout de même que pour 1

coref : entrer le nombre de nœuds qui pourraient avoir une coréférence

- être attentif aux POS ProDét et ProRel (par ex. mon, ta, ses, qui)
- le commentaire «coref» apparaît avant ces patrons génériques dans le lexicon
- penser aux pronoms personnels inclus dans la POS Vpro : se
- penser aux anaphores