Université de Montréal

Intégration de VerbNet dans un réalisateur profond

par

Daniel Galarreta-Piquette

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

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

Août 2018

RÉSUMÉ

La génération automatique de texte (GAT) a comme objectif de produire du texte compréhensible en langue naturelle à partir de données non-linguistiques. Les générateurs font essentiellement deux tâches : d'abord ils déterminent le contenu d'un message à communiquer, puis ils sélectionnent les mots et les constructions syntaxiques qui serviront à transmettre le message, aussi appellée la réalisation linguistique. Pour générer des textes aussi naturels que possible, un système de GAT doit être doté de ressources lexicales riches. Si on veut avoir un maximum de flexibilité dans les réalisations, il nous faut avoir accès aux différentes propriétés de combinatoire des unités lexicales d'une langue donnée. Puisque les verbes sont au cœur de chaque énoncé et qu'ils contrôlent généralement la structure de la phrase, il faudrait encoder leurs propriétés afin de produire du texte exploitant toute la richesse des langues. De plus, les verbes ont des propriétés de combinatoires imprévisibles, c'est pourquoi il faut les encoder dans un dictionnaire.

Ce mémoire porte sur l'intégration de VerbNet, un dictionnaire riche de verbes de l'anglais et de leurs comportements syntaxiques, à un réalisateur profond, GenDR. Pour procéder à cette implémentation, nous avons utilisé le langage de programmation Python pour extraire les données de VerbNet et les manipuler pour les adapter à GenDR, un réalisateur profond basé sur la théorie Sens-Texte. Nous avons ainsi intégré 274 cadres syntaxiques à GenDR ainsi que 6 393 verbes de l'anglais.

Mots-clés : génération automatique de texte ; réalisation linguistique ; patrons de régime ; cadres syntaxiques ; verbes ; Théorie Sens-Texte

ABSTRACT

Natural language generation's (NLG) goal is to produce understandable text from non-linguistic data. Generation essentially consists in two tasks : first, determine the content of a message to transmit and then, carefully select the words that will transmit the desired message. That second task is called linguistic realization. An NLG system requires access to a rich lexical ressource to generate natural-looking text. If we want a maximum of flexibility in the realization, we need access to the combinatory properties of a lexical unit. Because verbs are at the core of each utterance and they usually control its structure, we should encode their properties to generate text representing the true richness of any language. In addition to that, verbs are highly unpredictible in terms of syntactic behaviours, which is why we need to store them into a dictionary.

This work is about the integration of VerbNet, a rich lexical ressource on verbs and their syntactic behaviors, into a deep realizer called GenDR. To make this implementation possible, we have used the Python programming language to extract VerbNet's data and to adapt it to GenDR. We have imported 274 syntactic frames and 6 393 verbs.

Keywords : natural language generation ; linguistic realization ; government patterns ; syntactic frames ; verbs ; Meaning-Text theory

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET DES ABRÉVIATIONS

GAT	Génération automatique de texte
GP	Patron de régime, de l'anglais *Government Pattern*
DPOS	Partie du discours profonde, de l'anglais *Deep Part of Speech*
TST	Théorie Sens-Texte
TAL	Traitement automatique des langues
SPL	Sentence Planning Language
FD	Description fonctionnelle
RSem	Représentation sémantique
RSyntP	Représentation syntaxique profonde
RSyntS	Représentation syntaxique de surface
ND	Nœud dominant

REMERCIEMENTS

Je voudrais d'abord adresser mes remerciements à mon directeur de mémoire, François Lareau, pour m'avoir guidé dans le développement de ce projet, puis pour m'avoir engagé sur son projet GenDR, financé par la subvention FRQSC 2016-NP-191042.

Je voudrais aussi remercier mes collègues de l'Observatoire Linguistique Sens-Texte pour les beaux midis passés à discuter et à rire, j'en garderai de très bons souvenirs.

Je tiens également à remercier ma famille pour m'avoir aidé financièrement et pour m'avoir cuisiné des repas. Finalement, tous mes amis qui m'ont encouragé, mais spécialement mes colocataires qui ont facilité mes dernières semaines de rédaction.

INTRODUCTION

Les récentes percées en intelligence artificielle ne cessent de fasciner la population, notamment les applications liées au traitement automatique des langues (TAL) comme les systèmes de dialogue ordinateur-humain. Ainsi, de nombreuses recherches se sont orientées vers le développement de systèmes informatiques pouvant communiquer avec les humains. Or, la communication implique deux processus : la compréhension du langage et la production de celui-ci. Le deuxième processus a donné naissance à la génération automatique de texte (GAT), qui est une branche du TAL dont le but est de produire du texte compréhensible en langue naturelle à partir de données non-linguistiques. Les premiers systèmes de GAT ont été créés pour produire des rapports automatiquement dans le but d'alléger la charge de travail des humains (Reiter et Dale, 2000).

La GAT s'avère donc extrêmement utile en termes d'applications concrètes. On s'en sert régulièrement pour produire des documents résumant des informations complexes pour des gens n'ayant pas les connaissances requises pour les comprendre. Dans cette veine, le générateur MARQUIS (Wanner *et al.*, 2010) génère du texte à partir de données numériques brutes sur la qualité de l'air, ce qui permet à une large population d'avoir accès à ces informations en plusieurs langues. Toutefois, la GAT ne permet pas que de résumer des informations complexes, elle sert aussi à combler un manque de couverture dans certains contextes. Par exemple, la GAT permet la production d'articles sportifs dans des milieux où il n'y a pas de couverture médiatique, mais où il y a une demande (Lareau, Dras et Dale, 2011; Van Der Lee, Krahmer et Wubben, 2017).

Traditionnellement, un système de GAT se divise en deux parties. La première étant ce que Danlos (1983) appelle le *quoi-dire* et que Gatt et Krahmer (2018) appellent le *early process*, qui équivaut à déterminer et structurer le contenu à transmettre. Autrement dit, il s'agit d'évaluer, parmi les données s'offrant à nous, lesquelles nous voulons transmettre au lecteur, et dans quel ordre. Ensuite il y a ce que Danlos appelle le *comment-le-dire* et que Gatt et Krahmer appellent le *late process*, qui revient à choisir les unités lexicales qui serviront à véhiculer le *quoi-dire* et à construire des phrases grammaticales avec ces unités.

C'est ce qu'on appelle la **réalisation linguistique**. Cette étape du processus de génération a fait l'objet de plusieurs travaux, entre autre parce qu'il existe diverses approches pour l'effectuer. Nous nous concentrerons sur l'une d'entre elles : la réalisation linguistique à base de règles, qui correspond à modéliser les connaissances linguistiques d'une langue donnée dans une grammaire et un dictionnaire.

Comme le lexique d'une langue est incommensurable, la plupart des réalisateurs n'ont pas accès à tous les mots d'une langue ainsi que toutes leurs propriétés de combinatoire. La majorité des parties du discours présentent des régularités quant à leurs comportements syntaxiques, à l'exception des verbes qui forment la partie du discours la plus capricieuse. Ainsi, si on souhaite qu'un réalisateur génère du texte le plus naturel possible, on devrait récupérer les propriétés des verbes parce qu'ils sont les moins prévisibles et ils contrôlent essentiellement la structure de tout énoncé. Cette problématique ne s'applique pas uniquement à la GAT : toutes les branches du TAL bénéficieraient de connaissances fines des verbes. Korhonen, Krymolowski et Briscoe (2006); Schuler (2005) postulent qu'un meilleur traitement des langues naturelles passe par la connaissance des comportements syntaxiques des verbes. L'expression "comportements syntaxiques" réfère aux arguments qu'une unité lexicale sélectionne. Par exemple, lorsqu'on dit qu'un verbe peut être à la fois transitif direct et intransitif, il s'agit là de deux comportements syntaxiques (intransitivité du verbe MANGER : *Paul mange*, et transitivité : *Paul mange un dessert*). On encode généralement ces comportements dans des dictionnaires car on ne peut pas prédire les comportements des verbes à partir de règles grammaticales. Milićević (2009) illustre ce problème en montrant que deux verbes synonymiques montrent des comportements différents : *on se souvient de X*, mais *on se rappelle X*. L'obligation de la préposition dans la première construction est entièrement arbitraire, c'est pourquoi ce type d'information doit être encodé dans un dictionnaire pour qu'un système de GAT sache comment réaliser ces constructions syntaxiques.

Ainsi, en récupérant les propriétés syntaxiques des verbes d'une langue donnée, on arrive à couvrir une grande partie des constructions possibles pour cette langue. Nous pensons qu'un réalisateur profond bénéficierait d'une telle ressource, c'est pourquoi nous voulons intégrer à un réalisateur un dictionnaire de comportements syntaxiques décrivant de façon quasi exhaustive les constructions verbales possibles de l'anglais. Nous avons choisi GenDR (Lareau *et al.*, 2018; Lambrey et Lareau, 2015, 2016; Lambrey, 2017; Dubinskaite, 2017), un réalisateur profond multilingue qui reprend les bases du système MARQUIS et qui opère dans le cadre théorique de la théorie Sens-Texte (TST).

La ressource lexicale que nous avons choisie pour améliorer la couverture de GenDR est VerbNet (Schuler, 2005), un dictionnaire de verbes de l'anglais créé dans un contexte où

il y avait un réel besoin de faire un dictionnaire décrivant la richesse et la complexité des verbes (Kipper, Dang et Palmer, 2000). Schuler (2005) trouvait qu'il y avait un manque de lignes directrices par rapport à l'organisation des verbes dans les dictionnaires destinés à des applications TAL. Son dictionnaire est organisé en une hiérarchie de classes verbales héritées de Levin (1993), qui proposait une méthode de classification des verbes basée sur le partage des comportements syntaxiques entre ceux-ci. Levin a tenté de délimiter tous les patrons de régime possibles pour les verbes de la langue anglaise. Lorsque plusieurs présentaient des caractéristiques communes sur le plan syntaxique, elle rassemblait ces verbes dans une classe.

Pour intégrer VerbNet à GenDR, nous avons utilisé le langage Python, qui nous a permis de manipuler et d'extraire les données de la ressource lexicale choisie. En effet, nous avons du les manipuler pour que les informations concordent avec la TST. Pour compléter l'implémentation de cette ressource, nous avons du revoir certaines règles de grammaire et l'architecture de nos dictionnaires pour tenir compte des nouvelles informations lexicales auxquelles nous aurons accès.

Nous avons séparé ce mémoire en six chapitres, dont les trois premiers correspondent à la préparation du projet et les trois derniers à l'exécution du projet. Ainsi, le premier chapitre décrit ce qu'est la GAT en faisant un survol des six étapes du processus classique de génération de texte (Reiter et Dale, 2000). Nous retiendrons la dernière étape du processus, la réalisation, et nous la décrirons en détail. Après, nous présentons quelques réalisateurs pour montrer les différences théoriques et pratiques qui existent entre ceux-ci. Le deuxième chapitre porte entièrement sur le réalisateur profond GenDR. Nous décrivons comment ce réalisateur modélise les phénomènes langagiers et expliquons en détail les modules qui le composent. Le troisième chapitre est dédié à la description de diverses ressources lexicales potentielles, puis nous exposons pourquoi notre choix s'est arrêté sur GenDR et comment fonctionne ce système.

Dans le quatrième chapitre, nous expliquons comment nous utilisons Python pour préparer des scripts nous permettant d'extraire les données de VerbNet que nous désirions importer. Puis, le cinquième chapitre décrit comment nous avons adapté le réalisateur profond GenDR pour qu'il incorpore les données que nous avons extraites de VerbNet. Plus précisément, nous décrivons comment les règles de grammaire ont été adaptées, ainsi que les dictionnaires. Puis, le sixième chapitre comporte une évaluation du système basée sur le corpus de VerbNet. Enfin, la conclusion fait une synthèse de ce mémoire et évoque quelques pistes à explorer.

Chapitre 1

LA GÉNÉRATION AUTOMATIQUE DE TEXTE

La génération automatique de texte (GAT) est une branche du traitement automatique des langues (TAL) à la croisée des chemins entre l'intelligence artificielle et la linguistique computationnelle (Reiter et Dale, 2000). L'objectif de la GAT est de produire du texte compréhensible en langue naturelle à partir de données (non linguistiques). Bien que cet objectif soit commun à tous les générateurs de texte, il existe diverses méthodes pour y arriver. La diversité des méthodes est une conséquence directe de la combinaison des divers types d'inputs possibles et des multiples approches de réalisation. Par exemple, les systèmes de GAT peuvent prendre en input du texte, des données numériques et même des images (Thomason, Venugopalan, Guadarrama, Saenko et Mooney, 2014).

Les premiers systèmes de GAT ont été conçus pour générer des rapports automatiquement afin de faciliter le travail humain (Reiter et Dale, 2000). En effet, Daoust et Lapalme (2015) soulignent dans leur article que la GAT nous permet de générer un résumé d'information compréhensible pour un humain à partir de données brutes qui seraient normalement difficiles à lire. Ces tâches automatisées permettent d'éviter des coûts en termes de ressources et de temps. Autrement, la production de tels rapport est faite par un humain, ce qui entraîne des coûts importants souvent prohibitifs.

Les textes générés automatiquement n'ont pas besoin d'être lus par une grande quantité de gens pour être considérés utiles. On considèrera qu'ils remplissent leur fonction dès qu'ils comblent un besoin. De plus, les produits des systèmes de GAT peuvent aussi être *taillés sur mesure*, en fonction des besoins de l'utilisateur (Mahamood et Reiter, 2011). Par exemple, le système MARQUIS a implémenté cette composante en permettant à des utilisateurs de recevoir des bulletins sur la qualité de l'air en fonction de leur santé ou de leur profession (Wanner *et al.*, 2010).

D'autres chercheurs ont même poussé la GAT vers le domaine journalistique qu'on dénomme le robo-journalisme. Concrètement, ces systèmes prennent en entrée des données brutes concernant un match donné et produisent un article qui en décrit le déroulement (Van Der Lee *et al.*, 2017). En effet, il semble y avoir une demande pour cette branche de la GAT puisqu'il y existe un grand nombre d'évènements qui ne sont pas couverts, mais qui pourraient intéresser des gens. Ce problème a également incité Dras, Lareau, Börschinger, Dale, Motazedi, Rambow, Turpin et Ulinski à développer un système de GAT produisant des descriptions de matchs de football australien en anglais et en arrernte (langue aborigène) (Lareau *et al.*, 2011; Dras *et al.*, 2012) : les aborigènes australiens adorent le football australien, mais peu de couverture médiatique existe dans leur langue.

Finalement, il est aussi important de préciser que la GAT présente un intérêt pour la linguistique théorique. En effet, les linguistes peuvent s'en servir pour tester leurs théories et vérifier si leur modélisation de la langue contient des erreurs (Danlos, 1983).

1.1. Pipeline classique GAT

Selon Reiter et Dale (2000), un système de GAT se découpe en six modules, que nous illustrons à la figure 1.1. De façon plus grossière, on peut séparer ces modules en deux étapes majeures : le *quoi-dire* et le *comment-le-dire* (Danlos, 1983), qui correspondent aux *early process* et *late process* de Gatt et Krahmer (2018). Le *quoi-dire* fait référence à la sélection du contenu, la structuration du document et l'agrégation. Puis le *comment-le-dire* fait référence à toutes les étapes subséquentes : la lexicalisation, la génération d'expressions référentielles et la réalisation linguistique. Pour mieux comprendre ce processus séquentiel, nous décrirons en quelques lignes chacune des étapes qui le composent.

Un système de GAT doit différencier les informations qui seront exprimées dans le texte de celles qui n'y seront pas. Il s'agit donc de déterminer ce qui est pertinent dans les données brutes. Par exemple, si on souhaite générer le compte rendu d'un match de soccer, on ne voudrait probablement pas mentionner toutes les passes et toutes les fautes commises durant ce match, même si ces informations figurent dans les données de base. La **sélection du contenu** dépend aussi beaucoup du public à qui le texte s'adresse et de l'objectif du texte : on ne présente pas la même information à un expert et à un novice. Dans le cas d'un compte rendu sportif, par exemple, on pourrait vouloir adapter le texte en fonction de l'équipe préférée du lecteur.

Ensuite, on crée **le plan du texte** à générer. Il s'agit de décider dans quel ordre présenter les informations sélectionnées. Cette étape est une représentation ordonnée et structurée du message à transmettre. Reprenons l'exemple du soccer. En fonction des données choisies, le

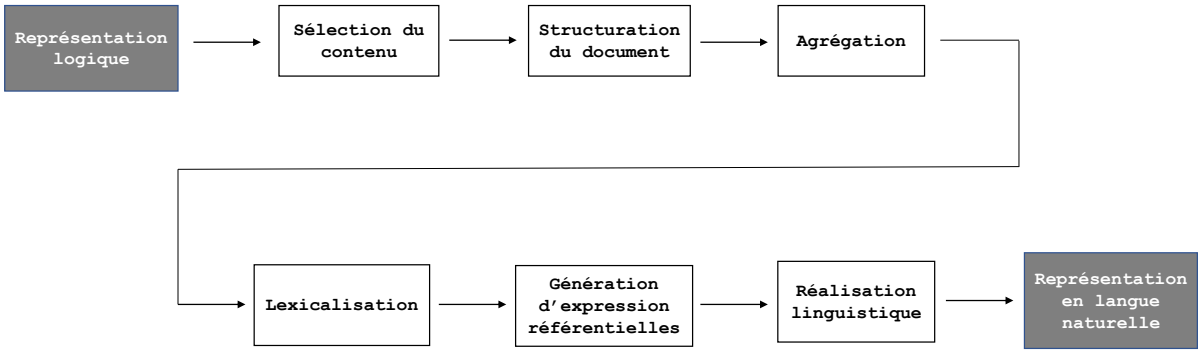


FIGURE 1.1. Pipeline classique, tiré de (Reiter et Dale, 2000)

texte devra débuter par les informations générales liées au match (où et quand le match s'est déroulé), suivi du nom des équipes qui s'opposaient, de qui a gagné, puis des faits saillants du match.

L'agrégation est l'étape où on combine des messages en une seule phrase afin de rendre le texte plus fluide et agréable à lire. Les messages sélectionnés dans le plan (structuration du document) n'ont pas à être exprimés dans des phrases individuelles si on est capable de les combiner (Cheng et Mellish, 2000). Bref, cette étape sert à éviter la redondance et un style trop télégraphique.

La **lexicalisation** est l'étape où l'on traduit les données non linguistiques en langue naturelle. Il s'agit de choisir les lexèmes qui seront utilisés pour transmettre le message voulu. Comme il existe naturellement plusieurs manières de rendre une même idée en mots, cette tâche peut devenir assez complexe si on veut que le système tienne compte des subtilités de la langue. La sélection des lexies peut ainsi se faire à divers niveaux d'abstraction. Un niveau plus abstrait requiert plus de travail et est plus complexe à mettre en place. Toutefois, il génère plus de possiblités au moment de la réalisation. Elhadad, Robin et McKeown (1997) prônaient une approche en faveur de la lexicalisation profonde. Par exemple, un système de GAT permettant une lexicalisation profonde pourrait générer les paraphrases suivantes : *Paul marque un but spectaculaire pendant la deuxième période* et *Paul réussi un but à couper le souffle lors de la seconde période.*

La génération d'expressions référentielles est très similaire à la lexicalisation car on choisit comment se réaliseront certaines entités. Le but est de s'assurer que le lecteur puisse

distinguer correctement chaque entité. Pour cela, il faut trouver la meilleure façon de référer à une entité. Pour ces raisons, on l'appelle souvent "l'étape discriminatoire". Continuons avec l'exemple du soccer. Cette étape du processus nous permet de référer à une même joueur de diverses manières dans le texte. Par exemple : l'attaquant, la jeune recrue, Joe McCain, McCain, Little-Joe, le récipiendaire du prix McKenna, etc.

La dernière étape est **la réalisation linguistique**. Lorsque tous les mots et les expressions référentielles ont été choisis, on peut réaliser le texte final. Cette tâche implique l'application de traits morpho-syntaxiques aux lexèmes et la linéarisation des structures. Elle inclut aussi l'insertion des mots fonctionnels (auxiliaires, déterminants, etc.) et la ponctuation. Cette étape du processus nous intéresse particulièrement, donc nous élaborerons sur celle-ci.

1.2. Réalisation

Puisque nous avons traité du processus global de GAT et des différentes méthodes de réalisation, nous pouvons entrer dans les détails de la tâche qu'est la réalisation linguistique, dans laquelle s'insère notre travail.

Tel qu'explicité à la figure 1.1, la réalisation est la dernière étape dans le processus de GAT. Toutefois, pour beaucoup de chercheurs, elle ne représente pas uniquement les tâches décrites précédemment. Il règne en effet un certain flou autour de ce qui constitue la réalisation. Pour certains, elle correspond exactement à la description de la réalisation que nous fournissons en 1.1. On appellera cela la réalisation de surface, puisque la réalisation se fait à partir d'un input beaucoup plus près du texte (des structures syntaxiques lexicalisées). Pour d'autres, la réalisation se fait à partir de données plus abstraites, en amont de la lexicalisation. Ce type de réalisation plus profonde prend généralement en input des structures pré-syntaxiques. On les appellera des réalisateurs profonds. Dans ces systèmes, les informations lexicales et grammaticales seront encodées dans des dictionnaires et des grammaires plus complexes permettant de traiter l'interface sémantique-syntaxe. Finalement, ces systèmes profonds sont généralement liés à une théorie linguistique leur permettant de modéliser la langue et de l'encoder dans un système informatique.

1.2.1. Types d'approches

Il existe traditionnellement trois types d'approches pour la réalisation.

Nous nommerons d'abord, l'**approche à base de patrons**. Elle est généralement utilisée pour produire du texte dans des domaines spécifiques comme la météo ou les sports. Cette

approche réduit la variation linguistique à un minimum puisque chaque réalisation de texte se fait à partir d'un moule fixe (McRoy, Channarukul et Ali, 2003). Les phrases en (1), provenant de l'article de Gatt et Krahmer (2018), démontrent comment de tels patrons s'utilisent.

(1) a. $player scored for $team in the $minute minute.
b. Ivan Rakitic scored for Barcelona in the 4th minute.

En (1-a), le patron contient trois variables, marquées par les $. Celles-ci peuvent être respectivement comblées par un nom de joueur, suivi d'un nom d'équipe et d'une indication temporelle. L'avantage de cette approche est qu'on peut prévoir exactement ce qui sera généré et on diminue, par le fait même, les risques d'erreurs. L'inconvénient est que cette approche est très rigide et laisse place à quasi aucune paraphrase. Cependant, les réalisateurs à base de patrons peuvent être complémentés par des règles de grammaire, ce qui les rend plus flexibles. Toutefois, puisqu'ils sont codés à la main, ces systèmes ont l'inconvénient d'être coûteux à développer. Gatt et Krahmer (2018) soulignent qu'ils peuvent aussi être combinés à de l'apprentissage machine pour automatiser l'écriture des patrons, ce qui rend la tâche moins coûteuse. D'ailleurs, la combinaison d'approches est une démonstration de l'affaissement des frontières (création de systèmes hybrides) entre les méthodes traditionnelles de réalisation.

Ensuite, il y a l'approche **basée sur des règles grammaticales**. Celles-ci s'emploient autant dans les domaines spécifiques (météo, sport, etc.) que les domaines généraux (conversation libre). Ces systèmes de réalisation se prêtent bien à la langue courante puisqu'ils sont conçus pour représenter le langage à travers des règles de grammaire. Cependant, les grammaires et les dictionnaires nécessaires au bon fonctionnement de cette approche sont codés manuellement, ce qui demande du temps et des ressources.

Finalement, il y a les **méthodes statistiques**. On peut les utiliser pour filtrer les sorties d'un système à l'aide d'un modèle probabiliste qui déterminera quel output est le meilleur candidat (Langkilde, 2000) (aussi appellée l'approche *generate-and-filter*). On peut aussi développer un modèle probabiliste permettant à un système de GAT d'apprendre automatiquement les règles nécessaires pour générer du texte (White et Rajkumar, 2012). Les méthodes statistiques diminuent considérablement la charge de travail manuelle (Langkilde, 2000).

Considérant les approches que nous venons de présenter, une question surgit quant à la nécessité de continuer à développer des méthodes à base de règles lorsqu'on peut sauver du temps et employer des méthodes probabilistes. À ce sujet, Belz et Kow (2009) se sont

demandé si le temps gagné en automatisant la réalisation se faisait au détriment de la qualité et si c'était le cas, à quel point. Leur évaluation consistait à tester les deux méthodes en leur demandant de faire la même tâche. Ils ont ainsi considéré la qualité des textes via une évaluation humaine, puis une évaluation métrique faite automatiquement. Suite à l'analyse, ils nous expliquent que les évaluations humaines pointaient en faveur des systèmes à base de règles et que certaines évaluations métriques surévaluaient le système statistique et sous-évaluaient le système à base de règles.

Reiter s'est aussi penché sur la question et en a fait un survol dans un article tiré de son blog. [1] Selon lui, même si les systèmes d'apprentissage automatique produisent généralement de bons textes, ils génèrent occasionnellement des bizarreries. Ce comportement n'est pas approprié dans des domaines professionnels où des utilisateurs comptent sur la qualité des textes, entre autres, car ils prendront potentiellement des décisions en fonction des textes générés. De plus, les systèmes basés sur des méthodes statistiques ont souvent plus de difficulté à corriger les incongruités. En revanche, les systèmes à base de règles permettent de cerner les problème avec plus de facilité et de les corriger. Cependant, Reiter termine son blog en soulignant que la GAT a aussi beaucoup à gagner de l'emploi des méthodes statistiques. Il suggère qu'on se serve d'elles pour automatiser des étapes du processus de réalisation (à base de règles).

1.2.2. Réalisateurs de surface

Dans cette section, nous présenterons trois réalisateurs de surface connus : SimpleNLG, JSrealB et RealPro.

1.2.2.1. *SimpleNLG*

SimpleNLG (Gatt et Reiter, 2009) est un réalisateur de surface pour l'anglais écrit en java. Le texte est réalisé à partir d'une structure syntaxique déjà lexicalisée encodée en *XML*. Par la suite, le réalisateur effectue les opérations morphologiques nécessaires (flexion, dérivation, ajout d'auxiliaires, gestion de l'accord) tout en linéarisant le texte.

Puisqu'il s'agit d'un réalisateur à base de règles, SimpleNLG est doté d'une grammaire et d'un dictionnaire. Ce dernier encode les propriétés syntaxiques et morphologiques des unités lexicales. Puis, le module grammatical contient des règles permettant le passage de la syntaxe à la morphologie.

1. https://ehudreiter.com/2016/12/12/nlg-and-ml/, 12-03-18

SimpleNLG sépare son processus de réalisation en quatre étapes. Premièrement, les lexèmes compris dans la structure d'input sont mis en correspondance avec leurs entrées de dictionnaire. Deuxièmement, on détermine les traits spécifiques des lexèmes. Troisièmement, on combine les lexèmes en créant des syntagmes de plus en plus larges, jusqu'à ce que l'entièreté de la phrase forme un syntagme phrastique. Finalement, les règles morphologiques sont appliquées et les lexèmes sont accordés afin d'obtenir les formes fléchies.

SimpleNLG a également été adapté à d'autres langues, dont l'espagnol (Ramos Soto, Janeiro Gallardo et Bugarín Diz, 2017), l'italien (Mazzei, Battaglino et Bosco, 2016), le français (Vaudry et Lapalme, 2013), le portugais (de Oliveira et Sripada, 2014) et l'allemand (Bollmann, 2011).

La figure 1.2 illustre l'input nécessaire permettant de réaliser la phrase *Refactoring is needed*. Elle provient du GitHub de SimpleNLG. [2]

```
<Document>
  <child xsi:type="SPhraseSpec">
    <subj xsi:type="VPPhraseSpec" FORM="PRESENT_PARTICIPLE">
      <head cat="VERB">
        <base>refactor</base>
      </head>
    </subj>
    <vp xsi:type="VPPhraseSpec" TENSE="PRESENT" >
      <head cat="VERB">
        <base>be</base>
      </head>
      <compl xsi:type="VPPhraseSpec" FORM="PAST_PARTICIPLE">
        <head cat="VERB">
          <base>need</base>
        </head>
      </compl>
    </vp>
  </child>
</Document>
```

FIGURE 1.2. Structure d'input permettant de réaliser *Refactoring is needed* dans SimpleNLG

1.2.2.2. *JSrealB*

JSrealB (pour *JavaScript Realiser Bilingual*) (Molins et Lapalme, 2015a) est conçu pour les programmeurs web, car il génère des expressions et phrases bien formées qui peuvent être formatées en *HTML* pour ensuite être utilisées dans un fureteur. Toutefois, JSrealB peut aussi s'employer seul à des fins purement linguistiques.

2. https://github.com/simplenlg/simplenlg/blob/master/docs/XMLRepresentationOfTextSpecifications.pdf

Pour générer du texte, ce réalisateur prend en input des structures syntaxiques lexicalisées. Dans ce système, la construction de la phrase découle de l'application de règles syntaxiques et morphologiques déclenchées par les lexèmes. De plus, JSrealB fonctionne aussi avec un dictionnaire et une grammaire. Son dictionnaire définit la catégorie des lexèmes qui le peuplent et leurs traits lexicaux (genre, nombre, irrégularités, etc.), puis sa grammaire contient des règles morpho-syntaxiques qui lui permettent de faire l'accord entre les constituants. Finalement, il existe aussi une version bilingue de JSrealB (Molins et Lapalme, 2015b) qui incorpore le français et l'anglais. L'input qu'illustre la figure 1.3 permet au réalisateur de générer la phrase : *The cat sat on the couch*. Cet exemple est tiré du GitHub de JSrealB. [3]

```
JSrealLoader({
       language: "en",
       lexiconUrl: URL.lexicon.en,
       ruleUrl: URL.rule.en,
       featureUrl: URL.feature
   }, function() {
   QUnit.test( "Sentence EN", function( assert ) {
       assert.equal(
           S(
               NP(D("the"), N("cat")),
               VP(V("sit"), PP(P("on"), NP(D("the"), N("couch"))))).t("ps")
           )
```

FIGURE 1.3. Structure d'input réalisant *The cat sat on the couch* dans JSrealB

1.2.2.3. *RealPro*

RealPro (Lavoie et Rambow, 1997) est implémenté en *C++* et c'est le plus profond des réalisateurs de surface présentés ici. Il prend en input des arbres de dépendances, ce qui le distingue des deux autres systèmes qui utilisent des arbres de constituants (Molins et Lapalme, 2015a; Gatt et Reiter, 2009). Un arbre de dépendance est un arbre non linéarisé dont les arcs et les nœuds (des lexèmes) sont étiquetés. On appelle ça un arbre syntaxique car les branches liant les nœuds entre eux correspondent à des relations syntaxiques. Seuls les mots sémantiquement pleins figurent dans l'input (cela exclut les déterminants, prépositions, etc.).

Cette architecture est basée sur la TST (Mel'čuk, 1988), une théorie linguistique qui a fait ses preuves en GAT (Iordanskaja, Kittredge et Polguère, 1988; Lavoie et Rambow, 1997; Wanner *et al.*, 2010; Mille et Wanner, 2017; Vicente, Barros, Peregrino Torregrosa, Agulló Antolín et Lloret, 2015). En bref, il s'agit d'une théorie qui divise le langage en quatre niveaux de représentations majeurs : sémantique, syntaxique (profonde et de surface),

3. https://github.com/rali-udem/JSrealB

morphologique (profonde et de surface), phonologique (profonde et de surface) – ce dernier étant court-circuité pour la génération de texte écrit. Pour plus de détails concernant les arbres de dépendances et la TST, voir le chapitre 2.

RealPro (Lavoie et Rambow, 1997) a deux grammaires qui permettent successivement de passer d'une représentation profonde à une représentation de surface, puis de celle-ci à une linéarisation morphologique de l'input. Pour effectuer toutes les transitions, le système fait aussi appel à un dictionnaire qui encode les propriétés lexicales et morphosyntaxiques des unités. La figure 1.4 illustre l'interaction de ces deux ressources pour réaliser du texte.

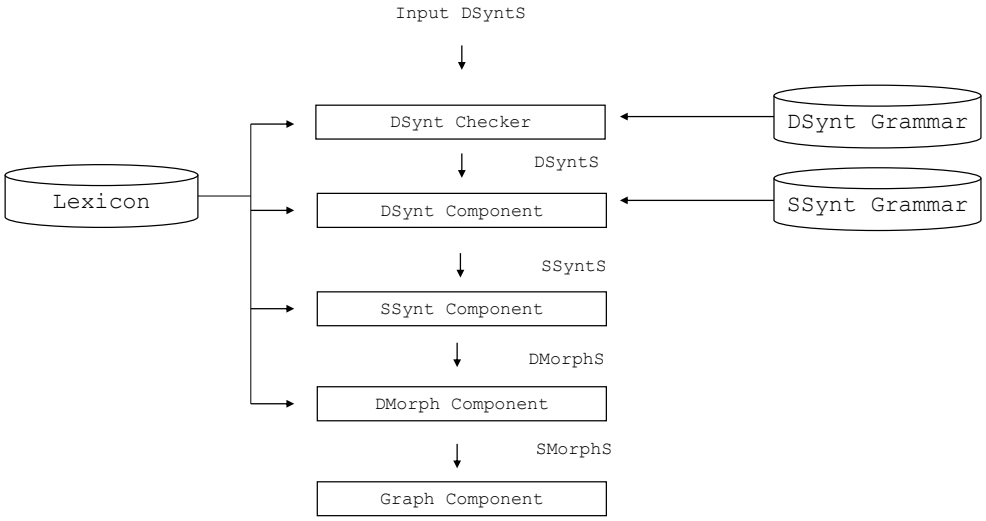


FIGURE 1.4. Modules dictionnairiques et grammaticaux de RealPro

La figure 1.5 illustre un arbre de dépendance (l'input) représentant la phrase *March had some rain days* ; elle provient de Reiter et Dale (2000).

```
HAVE1 [tense:past]
(I March [class:proper-noun]
II day [class:common-noun numberpl]
(ATTR rainy [class:adjective]))
```

FIGURE 1.5. Input de la phrase *March had some rain days* dans RealPro

1.2.3. Réalisateurs profonds

Les réalisateurs profonds prennent généralement en input des structures plus abstraites, ce qui permet de réaliser des phénomènes langagiers plus profonds. Comme les réalisateurs

profonds incorporent généralement la lexicalisation, cela fait en sorte qu'un seul input produit souvent plusieurs outputs (Polguère, 1998b), autrement dit, ces systèmes ont un grand pouvoir paraphrastique. Dans le pipeline classique, comme nous l'avons vu à la section 1.1, la lexicalisation est opérée avant la réalisation. Cet ordre a pour conséquence que les inputs fournis au réalisateur sont déjà lexicalisés et cela restreint grandement les réalisations possibles (puisque les lexèmes incorporent des propriétés de combinatoire bien précises autour desquelles la phrase doit s'articuler).

Dans cette section, nous présenterons les réalisateurs profonds suivants : KPML, SURGE, FORGe et MARQUIS.

1.2.3.1. *KPML*

KMPL (Bateman, 1997) est un réalisateur multilingue issu du système PENMAN (Mann, 1983). La théorie linguistique sous-jacente à ce système est la grammaire systémique fonctionnelle (SFG) (Matthiessen et Halliday, 1997), qui postule que les choix linguistiques sont déclenchés par l'exécution de fonctions grammaticales. D'un point de vue multilingue, les différentes langues issues de KPML partagent la majorité des fonctions grammaticales. Ces fonctions forment le cœur du système, mais il existe aussi quelques fonctions propres à chaque langue pour réaliser les phénomènes linguistiques spécifiques à chacune.

La grammaire de ce système est implémentée à la manière d'un réseau orienté. Chaque intersection dans le réseau correspond à un choix grammatical à faire entre différents traits fonctionnels. Ces choix deviennent de plus en plus précis lorsqu'on avance dans le réseau. Ainsi, la forme de surface d'un input donné est la conséquence du parcours de ces réseaux.

Les informations comprises dans les inputs de ce système sont d'ordre sémantique et syntaxique. Ainsi, l'input de KPML est un Sentence Planning Language (SPL), ce qui est une matrice d'attributs et de valeurs. La figure 1.6 illustre un SPL et elle est issue de Reiter et Dale (2000).

1.2.3.2. *SURGE*

SURGE, qui signifie *Systemic Unification Realisation Grammar of English*, est une grammaire de l'anglais (Elhadad et Robin, 1998) qui est écrite en Functional Unification Formalism (FUF) : une grammaire basée sur Functional Unification Grammar (FUG) (Kay, 1984). FUF est un langage de programmation créé pour construire des grammaires d'unification informatisées.

```
(S1 \ generalized-possession
  :tense past
    :domain (N1 \ time-interval
               :lex march
                          :determiner zero)
    :range (N2 \ time-interval
              :number plural
                       :lex day
                       :determiner some
                       :property ascription
                       (A1 \ quality :lex rainy)))
```

FIGURE 1.6. SPL : input pour réaliser *March had some rainy days* dans KPML

SURGE prend en input des arbres thématiques non linéarisés dont les nœuds sont des unités sémantiques pleines (donc ça exclut les prépositions, etc.). Ces arbres sont représentés comme des structures de traits encodés dans le formalisme FUF. Ces structures de traits sont appellées des descriptions fonctionnelles (FD). Il s'agit d'ensemble de paires d'attributs-valeurs qui peuvent être sucessivement enchâssées. Les entrées lexicales et leurs propriétés sont directement encodées dans les FD ce qui fait en sorte que SURGE n'a pas besoin de dictionnaire. Chaque constituant se fait attribuer une catégorie syntaxique et c'est grâce à ces traits que la grammaire sait comment unifier la phrase.

Dans ce système, la grammaire est aussi décrite en termes de FD. Ainsi, pour réaliser du texte, SURGE prend en entrée à la fois les FD du sens de la phrase désirée et les FD représentant les règles de grammaire du système afin d'unifier les structures. Le résultat est une structure d'entrée enrichie par des spécifications grammaticales (syntaxique et morphologique). Cela permet au réalisateur de passer à l'étape suivante qui est la linéarisation. SURGE applique ainsi les traits morphologiques en fonction des spécifications encodées lors de l'unification. L'output est une phrase anglaise exprimant le sens de la FD en fonction des contraintes imposées par la grammaire.

Nous reprenons encore un exemple tiré de Reiter et Dale (2000) à la figure 1.7 : *March had some rainy days.*

```
((cat clause)
 (proc ((type possessive)))
 (tense past)
 (partic ((possessor ((cat proper) head ((lex "March"))))
                  (possessed ((cat common) head ((lex day)))
                                     (describer ((lex rainy)))
                                     (selective yes) (number plural)))))
```

FIGURE 1.7. FD : input pour réaliser *March had some rainy days* dans SURGE

1.2.3.3. *MARQUIS*

Comme RealPro, MARQUIS est basé sur la TST, mais contrairement aux autres systèmes présentés ici, MARQUIS n'est pas qu'un réalisateur profond. Il s'agit d'un système de GAT complet qui effectue toutes les étapes du processus de génération automatique de texte (voir 1.1). Cependant, nous ne nous intéresserons ici qu'à son module de réalisation profonde (Lareau, Wanner, Fabra, Holloway et Bender, 2007). Pour plus d'informations concernant le pipeline complet de MARQUIS, voir (Wanner *et al.*, 2010; Bohnet, Lareau et Wanner, 2007).

Le but du projet MARQUIS était de générer, à partir de données brutes, des bulletins météorologiques multilingues sur la qualité de l'air. Ces bulletins sont produits en fonction de l'utilisateur. Autrement dit, ceux-ci se créent des profils avec leurs informations personnelles et cela permet à MARQUIS de réaliser du texte en fonction de leur niveau de connaissance du domaine et de leurs besoins de santé. Le tout est fait à partir d'un input conceptuel partagé par toutes les langues traitées par MARQUIS (l'anglais, l'allemand, l'espagnol, le catalan, le portugais, le français, le finnois et le polonais).

La figure 1.8 illustre le processus de réalisation que MARQUIS entreprend pour générer du texte. On peut y voir que le système prend en input des graphes conceptuels et qu'il produit du texte. Pour arriver à la réalisation finale, le réalisateur prend la structure #1 en input, puis il lui applique des règles pour qu'il en résulte la structure #2. Celle-ci lui servira d'input au prochain niveau de représentation. Cette transduction est ainsi répétée jusqu'à ce que le système se rende au dernier niveau de représentation, le texte.

Nous expliquerons brièvement ici les mécanismes qui permettent la transition d'une représentation à une autre (plus de détails suivront à la section 2.2). Il faut d'abord que le système analyse la structure conceptuelle à l'aide des règles de transduction qui permettent de passer de la représentation conceptuelle à la représentation sémantique. Le passage des unités conceptuelles aux unités sémantiques se fait grâce aux informations encodées dans un dictionnaire à cet effet. Ensuite, la création de la structure sémantique est effectuée et MARQUIS la prend pour en dériver un arbre de dépendances syntaxique profond grâce aux règles de transduction de cette interface (sémantique-syntaxe). La mise en correspondance d'unités sémantiques et lexicales est opérée grâce au dictionnaire sémantique (*semanticon*) et lexical (*lexicon*) ainsi qu'au dictionnaire de fonctions lexicales. Finalement, les autres transitions de représentations sont assurées par des règles de transductions successives et les informations contenues dans le *lexicon*. La figure 1.9 présente ce mécanisme.

Pour mieux comprendre à quoi servent les dictionnaires mentionnés dans le paragraphe précédent, nous les décrirons rapidement. Le dictionnaire conceptuel comprend tous les

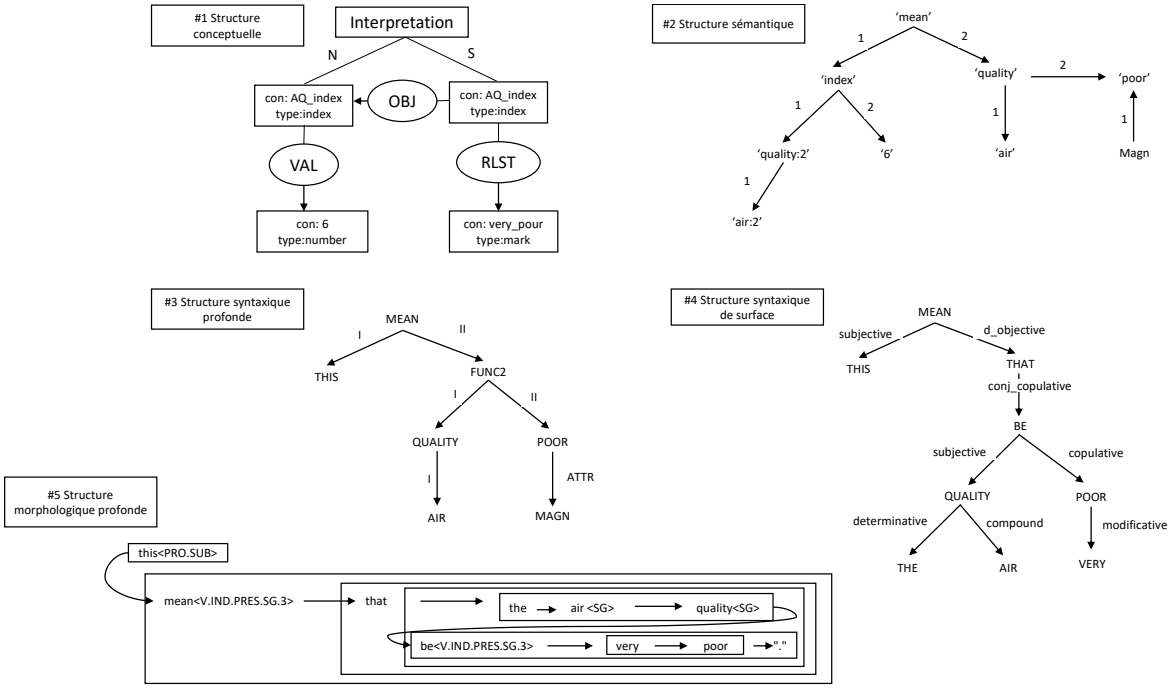


FIGURE 1.8. Pipeline de MARQUIS (Wanner *et al.*, 2010)

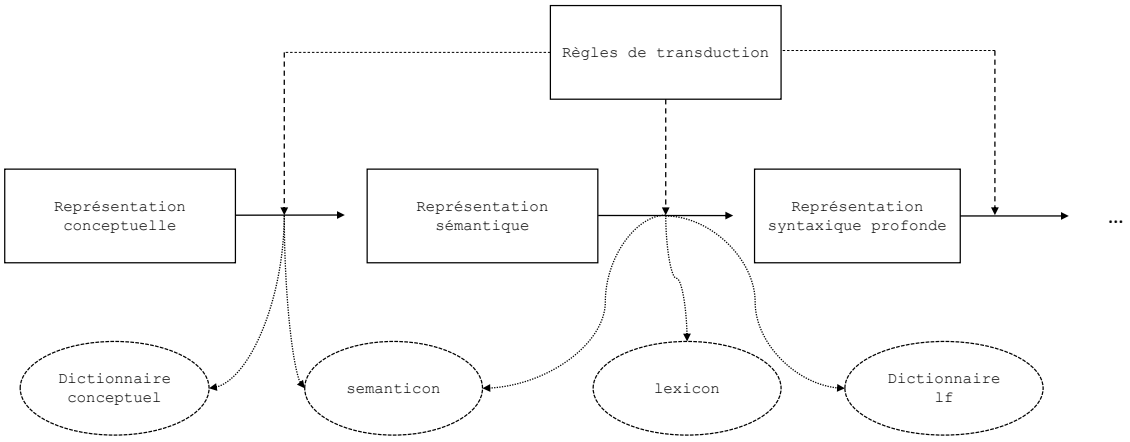


FIGURE 1.9. Combinaison des règles et dictionnaires pour effectuer les transductions de graphes (Lambrey, 2017)

concepts utiles à la génération des rapports sur la qualité de l'air et des termes du domaine général. Ce dictionnaire mappe les concepts aux unités sémantiques pour chaque langue traitée par le système. Le dictionnaire sémantique mappe les unités sémantiques aux

lexies. Finalement, il nous faut un dictionnaire lexical qui contient toutes les unités lexicales avec leurs propriétés syntaxiques et morphologiques et leur combinatoire lexicale pour que le texte généré soit grammatical.

1.2.3.4. *FORGe*

FORGe est un transducteur de graphes qui génère du texte à l'aide de ressources lexicales dictionnairiques et grammaticales (Mille et Wanner, 2017; Mille, Carlini, Burga et Wanner, 2017b). C'est un réalisateur profond qui a hérité de l'architecture de Marquis (Wanner *et al.*, 2010). FORGe a été conçu pour l'anglais à la base, mais il se veut multilingue (espagnol, allemand, français et polonais sont en développement). C'est un réalisateur qui peut aisément générer du texte en différentes langues grâce à ses règles grammaticales, qui seront majoritairement partagées par toutes les langues.

La théorie linguistique sous-jacente à ce système est la TST (Mel'čuk, 1988, 2012; Polguère, 1998a; Milićević, 2007) [4].

FORGe prend en input des représentations sémantiques sous forme de graphes orientés acycliques représentant les relations prédicats-arguments entre des sémantèmes, ce qui correspond à la structure #2 dans la figure 1.8 de la section MARQUIS 1.2.3.3. Plus haut, nous avons vu que RealPro (Lavoie et Rambow, 1997) réalisait du texte en passant par la représentation syntaxique profonde (RSyntP) en traversant la syntaxe de surface, puis la morphologie pour que le tout soit linéarisé. FORGe fonctionne essentiellement sur les mêmes bases, mais son input est encore plus profond que la RSyntP, il s'agit de la représentation sémantique (RSem). Autrement dit, une représentation dépourvue de structure où les relations entre les nœuds ne sont que prédicats-arguments. Ainsi, la réalisation de texte dans FORGe se découpe en trois étapes : le transfert de la RSem à la RSyntP, suivi du transfert de la RSyntP à la représentation syntaxique de surface (RSyntS), et finalement de la RSyntS au texte linéarisé et morphologisé. Le fonctionnement est donc très similaire à celui de RealPro, mais FORGe peut ainsi traiter beaucoup plus de phénomènes langagiers puisqu'il n'est pas contraint par les propriétés de combinatoire des unités lexicales dans sa structure d'input.

La première étape de réalisation est le passage de la sémantique à la syntaxe profonde qui est effectué via un algorithme récursif qu'on appelle "*top-down*". Tel que nous le verrons dans le chapitre suivant à la section 2.2.1, on rèfère à cet algorithme dans la TST comme **l'arborisation**. En résumé, on crée un arbre syntaxique de dépendances à partir d'une RSem qui précise un nœud dominant. Celui-ci sera l'équivalent du *top* (qu'on appelle la

4. Nous verrons plus en détail au chapitre 2 comment cette théorie linguistique se prête à des transducteurs de graphes.

racine) qui se voit généralement imposer une partie du discours verbale. Cette restriction s'impose souvent, car on veut que le lexème qui constituera la racine soit un verbe, puisque ce sont eux qui contrôlent généralement la phrase. Une fois que la racine est lexicalisée, on peut désormais ajouter des branches qui font office de relation syntaxique entre la racine et les arguments qu'elle sélectionne. Les nœuds au bout de ces branches sont vides et contraints par le gouverneur afin que seuls les lexèmes respectant les contraintes soient lexicalisés. On dit que cet algorithme est récursif car ces opérations se répètent jusqu'à ce que l'arbre représente en syntaxe profonde la RSem qu'on lui avait donné en entrée. Cette construction se fait sur les bases des dictionnaires et de la grammaire pour s'assurer que ce sera un arbre bien formé.

Ensuite, la seconde étape est le passage de la RSyntP à la RSyntS. Cela correspond à l'étape où on introduit les mots fonctionnels (prépositions, auxiliaires, déterminants) et les relations de surface (sujet, objet direct, etc.) dans l'arbre de dépendance créé à l'étape précédente. C'est là que FORGe se démarque de MARQUIS. L'héritier de ce dernier possède un dictionnaire de verbes qui explicite les comportements syntaxiques de ceux-ci. Cela permet de rendre compte de la richesse linguistique des verbes et de générer encore plus de paraphrases puisque c'est cette partie du discours qui contrôle généralement la construction d'un arbre. Nous reviendrons à cette problématique dans la section 2.5 au chapitre 2.

Finalement, la dernière étape pour achever la réalisation dans FORGe consiste à linéariser la structure syntaxique de surface et à appliquer les règles morpho-syntaxiques aux lexèmes pour produire le texte final.

En conclusion, MARQUIS et FORGe partent de niveaux d'abstraction plus profonds que les autres systèmes présentés. Cela leur permet d'être plus flexibles dans leurs réalisations. C'est pourquoi nous travaillerons avec un réalisateur profond très similaire à ces deux réalisateurs. Comme FORGe, nous utiliserons aussi un système basé sur MARQUIS, le système GenDR (Lambrey et Lareau, 2015; Lambrey, 2017; Lareau *et al.*, 2018), que le chapitre suivant décrira en détail.

Chapitre 2

GENDR : UN RÉALISATEUR PROFOND GÉNÉRIQUE

GenDR (pour *Generic Deep Realizer*) est un réalisateur profond multilingue (Lareau *et al.*, 2018) qui a hérité de l'architecture de MARQUIS (Wanner *et al.*, 2010) (section 1.2.3.3). Comme son prédécesseur, GenDR est un transducteur de graphes basé sur la TST, et ses dictionnaires et grammaires fonctionnent essentiellement de la même manière.

Cependant, contrairement à MARQUIS, GenDR génère uniquement des structures syntaxiques de surface, car ce réalisateur se concentre sur la modélisation de phénomènes langagiers profonds comme la lexicalisation et l'arborisation. Pour compléter la réalisation jusqu'au texte, il faudrait utiliser un réalisateur de surface qui prenne les outputs de GenDR en entrée et réalise du texte fléchi et linéarisé. À ce sujet, Mille, Bohnet, Wanner et Belz (2017a) proposent une compétition pour créer un réalisateur de surface multilingue capable de prendre en input des arbres de dépendances universels (Nivre, de Marneffe, Ginter, Goldberg, Hajic, Manning, McDonald, Petrov, Pyysalo, Silveira, Tsarfaty et Zeman, 2016). On pourrait alors envisager de mapper l'output de GenDR sur ce type de structure.

En tant qu'héritier de MARQUIS, GenDR en reprend les composantes de base pour modéliser les phénomènes langagiers élémentaires comme la lexicalisation simple, la complémentation et la modification. Ces règles forment le noyau du système et sont généralement partagées par l'ensemble des langues, tandis que les phénomènes grammaticaux spécifiques comme l'insertion des auxilaires ou des déterminants sont régis par des règles propres à chaque langue.

2.1. Architecture de GenDR

Les composantes lexicales et grammaticales de GenDR sont prises en charge par le transducteur de graphes MATE (Bohnet, Langjahr et Wanner, 2000; Bohnet et Wanner, 2010a).

```
owe { lex = owe
      lex = debt }
```

FIGURE 2.1. Échantillon du *semanticon*

Ainsi, pour mieux comprendre comment la réalisation se déroule dans GenDR, il faut d'abord présenter les composantes de ce logiciel.

MATE a été conçu à la base pour implémenter la TST dans le but de générer du texte. Tous les niveaux de représentation (voir section 2.2) de la TST correspondent à des graphes et la transformation d'un graphe en un autre est assurée par des ensembles de règles modélisant chaque interface (module sémantique et module syntaxique). En plus d'opérer la transduction de graphes, MATE a aussi été conçu pour tester, développer et maintenir une grammaire computationnelle, ce qui nous permet de réaliser du texte tout en testant les fondements théoriques de la TST.

MATE comprend un éditeur de dictionnaires, de graphes et de grammaires. Les dictionnaires encodent les propriétés des unités sémantiques et lexicales, alors que les grammaires sont composées de règles modélisant le passage d'un niveau de représentation à un autre. L'éditeur de graphes permet de construire et de visualiser ceux-ci. MATE comprend aussi un module d'inspection permettant de voir le déroulement de l'application des règles. Cet outil s'avère très utile au développement d'une grammaire, puisqu'il permet de cerner à quel endroit la réalisation a échoué. Pour plus de détails, voir Bohnet et Wanner (2010b); Lambrey (2017); Lambrey et Lareau (2016).

2.1.1. Dictionnaires

GenDR utilise trois dictionnaires pour décrire le vocabulaire qui sera utilisé dans les textes : un qui fait le pont entre les unités sémantiques et les unités lexicales, un qui décrit le comportement syntaxique des unités lexicales, et un qui décrit les fonctions lexicales. Nous excluerons le troisième dictionnaire de la discussion parce que nous ne traitons pas les fonctions lexicales dans ce mémoire.

Le dictionnaire sémantique, qu'on appelle *semanticon* (illustré à la figure 2.1), est utilisé par le système lors du passage de la RSem à la RSyntP. Cette ressource décrit les lexicalisations possibles pour un sémantème donné. C'est une source importante de paraphrasage puisqu'une unité sémantique peut souvent se réaliser de plus d'une manière. Par exemple, en anglais, le sens 'owe' peut se réaliser par les lexèmes OWE (un verbe) et DEBT (un nom).

Le dictionnaire lexical, appelé *lexicon*, contient de l'information détaillée à propos des lexies d'une langue donnée. Les entrées contiennent : la partie du discours profonde (DPOS), la partie du discours superficielle (SPOS), la diathèse et les patrons de régime (GP). Ces deux derniers concepts seront clarifiés à la fin du chapitre, section 2.5, mais en résumé, la diathèse d'une lexie assure la correspondance entre ses actants sémantiques et ses actants syntaxiques, puis un GP correspond à la description d'une construction syntaxique sélectionnée par un lexème.

Dans le *lexicon*, ces propriétés (DPOS, SPOS, GP et diathèse) ne sont pas toujours explicitées pour chaque entrée lexicale, puisque GenDR renferme un mécanisme d'héritage permettant de transmettre des traits entre lexèmes. Par exemple, à la figure 2.2, le verbe OWE contient très peu d'information syntaxique parce qu'il hérite de plusieurs propriétés de la classe absraite `verb_dit`, qui correspond à la classe des verbes ditransitifs. Cela permet à OWE d'hériter d'une partie du GP de la classe `ditransititive` qui le domine. À son tour, la classe *verb_dit* hérite des traits de sa classe mère, `verb_dit : verb_dt`. Autrement dit, la classe des verbes ditransitifs imbriquent les propriétés de la classe transitive, en plus de préciser la sélection d'un actant syntaxique de plus. Ainsi, le mécanisme d'héritage facilite grandement l'encodage des unités lexicales, car on n'a qu'à associer chaque nouvelle entrée lexicale à la classe abstraite qui lui correspond. Il en existe pour chaque partie du discours (verbes, noms, adjectifs, adverbes, prépositions, etc.).

On peut aussi bloquer l'héritage de traits pour une entrée lexicale donnée. Par exemple, la figure 2.2 illustre que la lexie OWE permet deux DPOS différentes pour le deuxième actant syntaxique qu'elle contrôle : `gp={II={dpos=Num}}` et `gp={II={dpos=N}}`. Elle permet ainsi la sélection d'un actant de type nominal ou numéral, ce qui ne fait pas partie du régime de la classe abstraite `verb_dit`.

Finalement, les dictionnaires anglais et français de GenDR contiennent les 1 500 lexèmes les plus courants de chaque langue. Pour leur part, les dictionnaires lituanien (Dubinskaite, 2017) et persan contiennent respectivement ∼180 et ∼60 lexèmes, assez pour tester le système.

2.1.2. Grammaires

La grammaire de GenDR est organisée autour de deux modules de règles : le module sémantique (RSem–RSyntP) et le module syntaxique (RSyntP–RSyntS).

Le module RSem–RSyntP s'occupe de faire l'arborisation (algorithme *top-down*) et la lexicalisation profonde (sélection des lexèmes pour représenter les sémantèmes) de l'input. Ce module est divisé en trois groupes de règles : le cœur partagé par toutes les langues, les

```
predicate {
  gp = { 1 = I
         2 = II
         3 = III
         4 = IV
         5 = V
         6 = VI }
}
verb : predicate {
  dpos = V
  spos = verb
  gp = {
     I = {
        dpos = N
        rel = subjective
     }
  }
}
verb_dt : verb { // direct transitive
  gp = {
     II = {
        dpos = N
        rel = dir_objective

     }
  }
}
verb_dit : verb_dt { // direct ditransitive
  gp = {
     III = {
        dpos = N
        rel = indir_objective
        prep = to

     }
  }
}
[...]
owe : verb_dit {
  gp = { II = { dpos=Num } }
  gp = { II = { dpos=N } }
}
```

FIGURE 2.2. Échantillon du *lexicon*

fonctions lexicales, et les règles spécifiques à chaque langue. Le module sémantique contient 21 règles générales, dont la plupart sont héritées de MARQUIS, et 132 règles de lexicalisation, dont la plupart gèrent des patrons de collocations (Lambrey, 2017).

Le module RSyntP–RSyntS s'occupe de l'arborisation et la lexicalisation de surface (ajout des mots fonctionnels et des relations de surface). Il est divisé en deux : les règles partagées

par toutes les langues et les règles propres à chaque langue. Le module syntaxique en contient nettement moins : vingt règles, dont douze partagées entre les langues. Nous construirons donc la nouvelle version de GenDR à partir des bases mises en place par Lambrey (2017), Dubinskaite (2017) et Lareau *et al.* (2018). Toutefois, nous avons retiré les patrons de collocation de notre système pour mieux nous concentrer sur le régime des verbes sans nous perdre dans des interactions inutilement complexes entre les modules.

Nous présenterons en détail les règles principales de ces deux modules dans les sections 2.2.1 et 2.3.

Pour donner une meilleure idée de l'allure de l'éditeur de règles, la figure 2.3 présente une règle du module sémantique-syntaxe profonde. Toutes les règles sont décrites dans ce format. D'abord, en haut, nous avons l'interface dans laquelle la règle opère, suivis du nom de la règle et de l'identification de son groupe. Dans la partie gauche (*Leftside*) on retrouve les nœuds et arcs du graphes de départ, puis dans la partie droite (*Rightside*) on retrouve le graphe créé en sortie. Finalement, la partie du bas contient les conditions d'application de la règle donnée. Nous verrons en détails l'application concrète de la règle illustrée à la figure 2.3 dans la section 2.2.1. Pour une présentation plus détaillée de la création de règles, voir le manuel de GÉCO (Lambrey et Lareau, 2016).

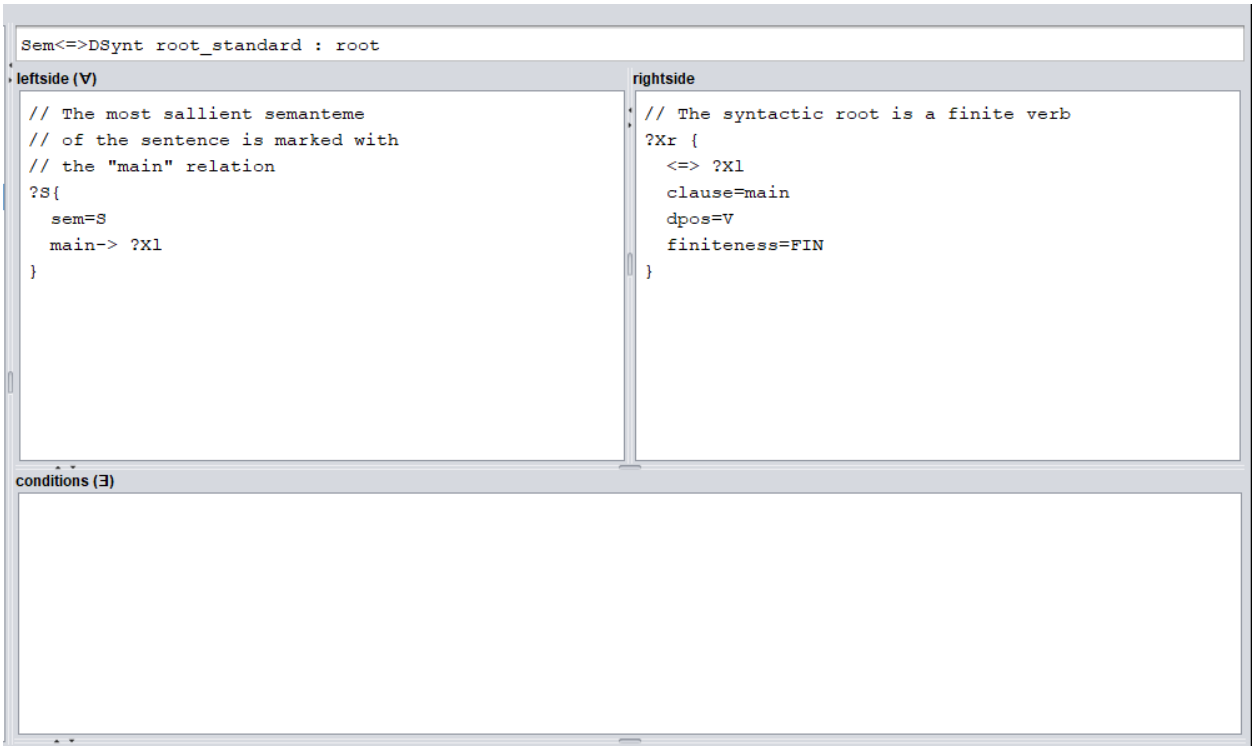


FIGURE 2.3. Règle créant la racine syntaxique

```
structure Sem debt {
  S {
    owe {
    tense=PRES
    1-> Paul {class=proper_noun}
    2-> "\$500K" {class=amount}
    3-> bank {number=SG definiteness=DEF}}
    main-> owe } }
```

FIGURE 2.4. Graphe sémantique en mode textuel

2.1.3. Graphes

Maitenant que nous avons fait un survol des dictionnaires et de la grammaire, il ne nous reste qu'à présenter les graphes. Dans un premier temps, nous présenterons brièvement la construction d'un graphe d'input, puis nous montrerons à quoi ressemblent les graphes en output.

L'input du réalisateur GenDR est un graphe sémantique (Mel'čuk, 2012) où les prédicats sont liés à leurs arguments par des relations numérotées qui indiquent la position logique de l'argument. La figure 2.4 montre comment on construit un graphe sémantique dans MATE.

Toutefois, MATE offre aussi une version graphique pour encoder et visualiser les graphes (voir la figure 2.5).

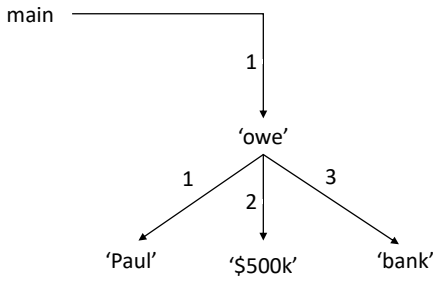


FIGURE 2.5. Graphe sémantique en mode visuel

L'éditeur de graphes de MATE nous permet de construire les structures sémantiques d'input et de visualiser les graphes en output. Pour l'input donné à la figure 2.4 , GenDR réalise six structures syntaxiques de surface. Celles-ci peuvent être visualisées dans MATE grâce au module de graphes. La figure 2.6 est un exemple d'output visuel qu'offre GenDR et la figure 2.7 montre à quoi ressembleraient les six représentations possibles si on les linéarisait. Cette variété de représentations est possible grâce au mécanisme de paraphrasage de GenDR.

Pour obtenir une réalisation linéarisée et fléchie, il nous faudrait utiliser un réalisateur de surface. Pour cela, on peut envisager d'interfacer la sortie de GenDR avec divers réalisateurs

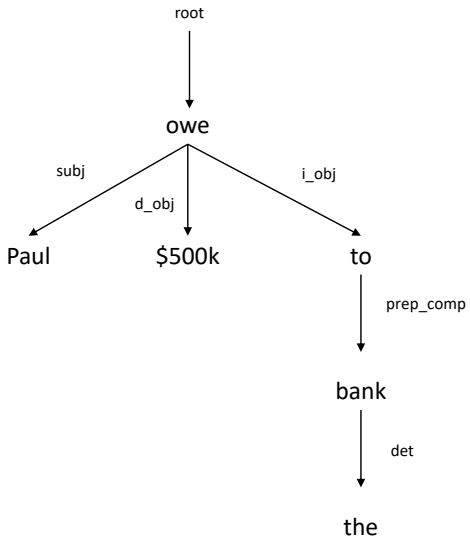


FIGURE 2.6. Réalisation de surface

de surface (Daoust et Lapalme, 2015; Molins et Lapalme, 2015a; Gatt et Reiter, 2009; Belz, White, Espinosa, Kow, Hogan et Stent, 2011).

2.2. INTERFACE SÉMANTIQUE-SYNTAXE EN TST

Dans cette section, nous décrirons l'interface sémantique-syntaxe dans le cadre de la TST afin de mieux rendre compte des deux processus nécessaires au passage de la sémantique à la syntaxe : l'arborisation et la lexicalisation. Ces opérations sont la force de GenDR en termes de paraphrasage et de la flexibilité du système à réaliser des phénomènes langagiers profonds. Pour mieux comprendre où se situent ces procédés dans la modélisation du langage, nous ferons un très bref retour sur les fondements de ce cadre théorique.

La TST vise la modélisation formelle de la correspondance entre les *Sens* et les *Textes* (Jolkovsky et Mel'čuk, 1967; Mel'čuk, 1997; Polguère, 1998a). La figure 2.8 (qui est empruntée à Polguère (1998a)) présente comment fonctionnent ces modèles qui sont des machines virtuelles prenant en entrée des représentations sémantiques d'énoncés pour générer du texte.

Pour se rendre au *Texte* final, le *Sens* traverse de nombreux niveaux de représentation. La figure 2.9 illustre les transformations que l'input doit subir successivement et, parallèlement, le formalisme utilisé pour représenter les niveaux.

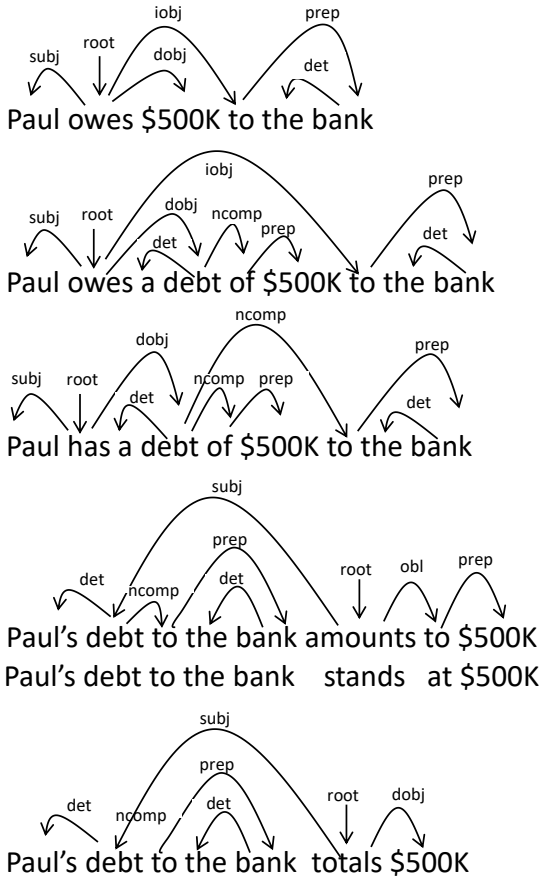


FIGURE 2.7. Six réalisations syntaxiques de surface (Lareau *et al.*, 2018)

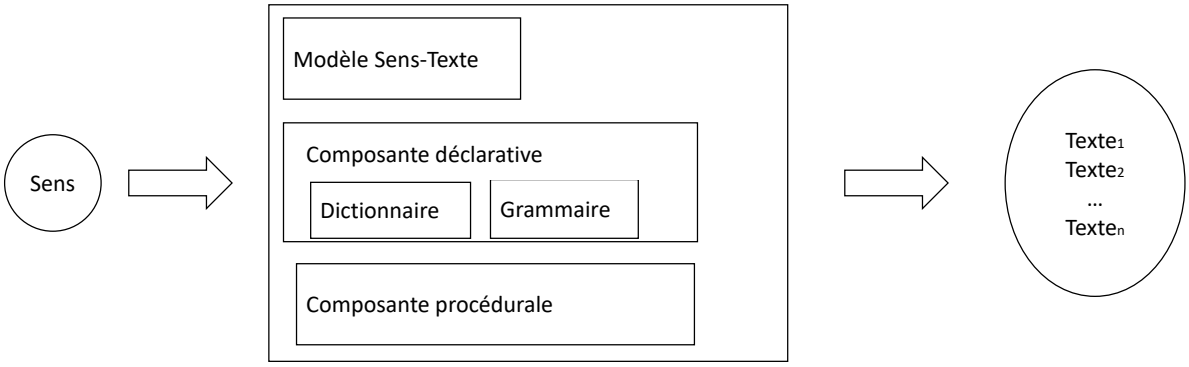


FIGURE 2.8. Structure d'un modèle Sens-Texte (Polguère, 1998a)

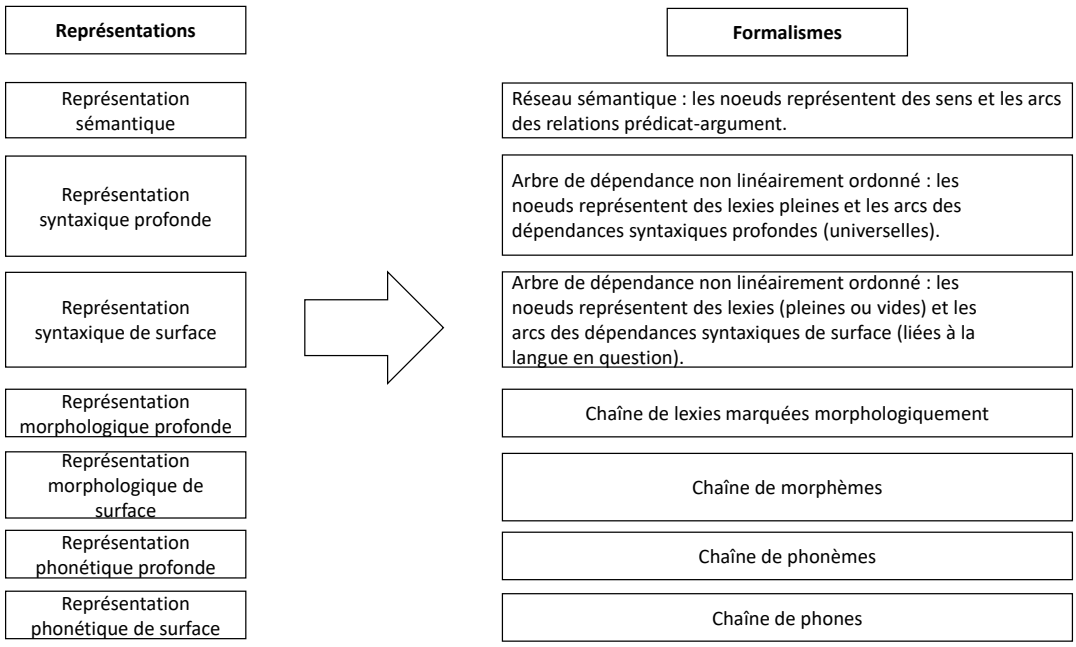


FIGURE 2.9. Processus d'un modèle Sens-Texte (Polguère, 1998a)

2.2.1. Arborisation

L'arborisation est le processus de transformation d'un graphe sémantique acyclique (DAG) en arbre de dépendance. Nous décrirons ce processus à la fois d'un point de vue théorique et d'un point de vue appliqué.

Selon la TST, la structure syntaxique d'un énoncé correspond à l'ensemble des liens de dependances fonctionnelles qui existent entre les unités lexicales de cet énoncé (Mel'čuk, 1988). Cette approche syntaxique provient de Tesnière (1965) qui est le premier à l'avoir théorisée. Formellement, le passage de la RSem à la RSyntP se fait grâce à la combinaison de l'arborisation et la lexicalisation. L'arborisation se fait ainsi à partir d'une structure sémantique qui subit des transformations grâces à l'application de règles de correspondances sémantiques pour qu'il en résulte un arbre de dépendances profond. Pour illustrer ce qu'est l'arborisation, nous reprendrons la description de Lareau *et al.* (2018) dans le contexte de GenDR. L'arbre syntaxique profond est construit avec un algorithme *top-down* qui est emprunté de MARQUIS, mais qui est, à la base, inspiré de Polguère (1990). Nous l'avions brièvement illustré avec la figure 1.8 lorsque nous avons présenté MARQUIS et FORGe.

Dans GenDR, l'arborisation s'opère en trois étapes (Lareau *et al.*, 2018) :

1. **Création de la racine**. La première règle appliquée s'appelle *root_standard*, elle construit la racine de l'arbre syntaxique à partir du nœud dominant (ND) de la

structure sémantique. À cette étape, la racine n'est pas étiquetée par un lexème encore, donc le nœud est vide. Toutefois, ce nœud se fait généralement imposer les contraintes suivantes dans les langues européennes : on exige que ce soit un verbe et qu'il soit fini, puisque ce sont généralement les verbes qui contrôlent la structure d'un énoncé. Bref, la détermination de la racine correspond au processus de hiérarchisation de Polguère (1990) puisqu'on détermine le nœud qui dominera tous les autres.

2. **Lexicalisation de la racine**. Une fois que la racine a été créée et contrainte, on applique une règle de lexicalisation (appellée *lex_standard*) qui permet au système de fouiller dans les dictionnaires pour trouver un lexème qui correspondra au sens demandé tout en respectant les contraintes imposées. Ainsi, il faut que le lexème sélectionné corresponde au ND en sémantique et qu'il respecte les contraintes sur le nœud.

3. **Application des règles actancielles**. Une fois que le nœud racine est lexicalisé, GenDR récupère les informations encodées dans le patron de régime de la racine pour savoir comment effectuer le passage des arcs sémantiques aux arcs syntaxiques (la diathèse). Autrement dit, cette étape crée les arcs dépendant de la racine qui correspondent aux arguments liés au ND dans la RSem. Les règles actancielles fonctionnent exactement comme Polguère l'avait théorisé :

   > chaque arc est considéré successivement dans l'ordre du parcours, puis est traduit en une micro-structure syntaxique profonde grâce aux règles de correspondance sémantique de la grammaire.
   >
   > (Polguère, 1990, p. 273)

   Les micro-structures syntaxiques correspondent aux branches créées et aux nœuds au bout de celles-ci qui se font, à leur tour, imposer des contraintes par le GP du nœud qui les gouverne (dans ce cas, la racine). C'est ainsi qu'on retourne à la seconde étape pour lexicaliser ces nœuds. Puis, nous répétons la troisième étape qui est de créer les arcs en partance de ces nœuds, puisque de nouveaux nœuds lexicalisés amènent leur patron de régime avec eux. Puis on boucle à travers les étapes 2 et 3 jusqu'à ce que le graphe sémantique soit complètement réalisé en syntaxe profonde.

Une fois que l'arborisation et la lexicalisation sont effectuées, GenDR passe à la prochaine étape : l'arborisation et la lexicalisation de surface qui impliquent les opérations suivantes. Dans un premier temps, les relations syntaxiques profondes seront réalisées en surface par l'étiquette qui leur correspondront (ex : la relation *I* deviendra *sujet*, la relation *II* deviendra *complément d'objet direct*, etc.). Puis, il faudra incorporer les lexies vides (prépositions, déterminants, etc.) qui sont encodées dans les lexies profondes sous formes d'attributs. Cela

est effectué grâce aux règles de correspondances profondes (le module syntaxique) et grâce aux informations contenues dans les patrons de régime des entrées concernées.

2.2.2. Lexicalisation

Le processus de lexicalisation est réparti sur plusieurs niveaux de représentation (RSem, RSyntP, RSyntS) formant deux interfaces, RSem–RSyntP et RSyntP–RSyntS (Lareau *et al.*, 2018; Polguère, 1998b). La première interface modélise la lexicalisation profonde qui consiste à assurer la correspondance entre une unité sémantique et une (ou des) unité(s) lexicale(s) profonde(s). Cette correspondance est encodée dans le dictionnaire d'un système TST. La seconde interface consiste à effectuer la lexicalisation de surface qui introduit les mots fonctionnels (déterminants, auxiliaires ou prépositions) et les lexies de surface (comme les collocations).

> La lexicalisation vise donc à sélectionner les Sens qui seront directement exprimés par des lexèmes au niveau syntaxique profond.
>
> (Polguère, 1990, p. 154)

Afin d'illustrer le fonctionnement de la lexicalisation dans GenDR, nous reprendrons la description de Lareau *et al.* (2018).

Les lexicalisations simples sont traitées par la règle *lexicalization_standard* qui prend une unité sémantique donnée dans un graphe d'input et récupère dans le *semanticon* les correspondances lexicales de ce sémantème (cf. 2.1). S'ensuit la sélection de l'unité lexicale : GenDR s'assure que la DPOS de la lexie choisie correspond à celle qui est demandée par le nœud contraint dans l'arbre syntaxique profond en construction. Comme il existe parfois plus d'une lexicalisation possible pour un sémantème donné, le système créera un arbre profond différent pour toutes les variantes lexicales possibles, ce qui engendre une bonne partie du paraphrasage de GenDR.

Le lexicalisation par classe permet de lexicaliser les nombres, nom propres, dates, etc. car ce sont des unités sémantiques que nous ne voulons pas dans notre dictionnaire en raison de leur quantité quasi-infinie et de la prévisibilité de leur comportement syntaxique. Pour identifier ces sémantèmes, on leur attribue un trait `class`, qui déclenchera l'application de la règle *lex_class* et qui fera en sorte que ces sémantèmes hériteront des traits nécessaires associés à leur classe respective. En effet, la valeur de l'attribut `class` (ex :`class=proper_noun`) pointera vers l'une des classes prévues qui sont encodées dans le *lexicon* et qui regroupent l'information sur la partie du discours et les traits grammaticaux à transmettre. Finalement, comme nous n'avons pas d'entrées spécifiques pour les lexicalisations de ces sémantèmes,

GenDR se sert de la règle *lex_class*, qui copiera l'étiquette sémantique pour en faire la lexicalisation de l'unité de type classe.

La lexicalisation de secours permet à GenDR de lexicaliser une unité sémantique ou lexicale dont il n'a pas les informations. Comme le système possède les 1 500 lexies les plus fréquentes de l'anglais, il est évident que certaines lexies ou sémantèmes manqueront à l'appel. Pour remédier à la situation, il y a d'abord un mécanisme qui vérifie si l'input existe sous la même forme dans le *lexicon*. Si c'est le cas, alors le système réalisera le sémantème à l'aide de ce lexème. Cette particularité du système a pour conséquence qu'on n'a pas besoin d'inscrire le sémantème dans le *semanticon* s'il n'y a qu'une seule lexicalisation possible de celui-ci (cela permet de sauver du temps). Cependant, si le sémantème ne figure pas dans le *lexicon*, alors le système suppose que l'étiquette de l'unité sémantique est la même que l'unité lexicale, et s'il y a des contraintes sur le nœud d'arrivée, alors le système supposera que l'unité lexicale ainsi créée satisfait ces contraintes. Les lexèmes devinés seront mis en évidence dans la structure d'output afin qu'on ait une trace de cette lexicalisation de secours (on pourra alors filtrer ou post-traiter ces phrases).

La lexicalisation grammaticale effectue des lexicalisations de surface (module RSyntP–RSyntS). Elle introduit deux types de mots fonctionnels. D'abord, les lexies qui expriment un sens grammatical apparaissant en attribut sur un nœud donné (voir le listing 2.4), comme les auxiliaires et les déterminants. Ils n'apparaissent pas en tant que nœuds en syntaxe profonde, mais plutôt comme des traits grammaticaux sur le verbe ou le nom qu'ils modifient. De plus, ils doivent être introduits comme des nœuds supplémentaires en syntaxe de surface, puisqu'il n'étaient pas exprimés lexicalement avant. Ces mots, qui sont peu nombreux, appartiennent à des classes fermées, donc on a des règles spécifiques pour les réaliser en syntaxe de surface, en fonction de la langue dans laquelle le système opère.

Puis, il y a un second type de lexies fonctionnelles, notamment les prépositions et les complémenteurs qui sont imposées par les GP d'autres lexies. Ils n'apparaissent pas non plus en syntaxe profonde, c'est pourquoi ils sont introduits en syntaxe de surface comme des nœuds supplémentaires entre un gouverneur et son dépendant. Pour lexicaliser ce type de lexie, le système récupère l'étiquette lexicale dans le GP du gouverneur, puis trouve l'entrée lexicale correspondante de la préposition ou du complémenteur dans le dictionnaire.

2.3. Exemple

Pour illustrer le fonctionnement des règles et des dictionnaires que nous venons de présenter, nous décortiquerons la réalisation profonde de l'exemple vu à la figure 2.4.

2.3.1. Arborisation et lexicalisation profonde

Cette section illustre les étapes successives qui ont permis à GenDR de passer de la RSem à la RSyntP.

2.3.1.1. *Application de la règle* root_standard

La règle crée un nœud vide qui sera la racine de l'arbre à construire et qui se voit imposer les contraintes suivantes : `dpos=V` (la racine doit être un verbe) et `finiteness=FIN` (fini). La figure 2.10 illustre la création du nœud non-étiqueté.

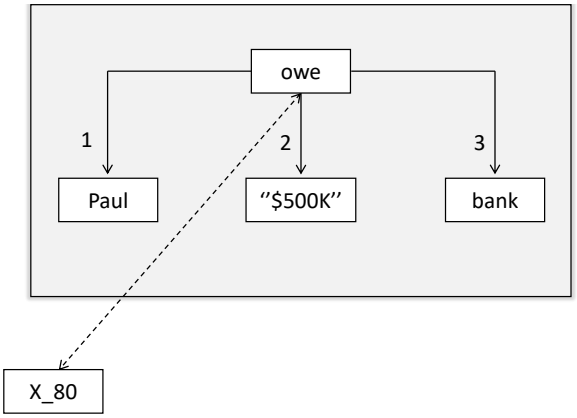


FIGURE 2.10. Application de *root_standard*

2.3.1.2. *Application de la règle* lex_standard

La racine vide déclenchera l'application d'une règle de lexicalisation, c'est pourquoi GenDR fouillera dans le dictionnaire à la recherche des lexèmes représentant le sémantème 'owe' : `owe { lex=owe lex=debt}`. GenDR peut ainsi lexicaliser la racine en choisissant DEBT, ce qui entraînera l'application d'une fonction lexicale permettant de réaliser un verbe support sur la racine (ce qui satisfait la contrainte `dpos=V`). Il s'agit d'une lexicalisation complexe, donc nous renvoyons le lecteur à Lambrey et Lareau (2015); Lambrey (2017); Lareau *et al.* (2018) pour nous concentrer sur la lexicalisation simple effectuée grâce au lexème OWE, ce qui est illustré à la figure 2.11.

2.3.1.3. *Application de la règle* actant_gp

Une fois que la racine est pourvue d'un lexème, l'application de la règle *actant_gp* est déclenchée. Celle-ci traduit les relations sémantiques prédicat-argument de 'owe' en arcs syntaxiques profonds. La correspondance est confirmée par les informations sur la diathèse

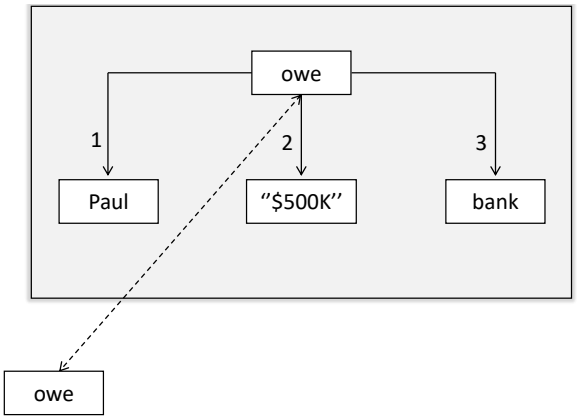


FIGURE 2.11. Application de *lex_standard*

encodées dans le régime de la lexie OWE. Comme le prédicat lie trois actants sémantiques, alors la règle s'applique trois fois. Ensuite, les nœuds au bout des arcs syntaxiques seront contraints en fonction des restrictions prévues par le GP de OWE. Ce mécanisme, illustré à la figure 2.12, assure la grammaticalité de la construction de l'arbre. Pour cet exemple, le GP de OWE permet à son deuxième actant syntaxique d'être soit un nom soit un nombre (la figure 2.2 montre le régime du verbe).

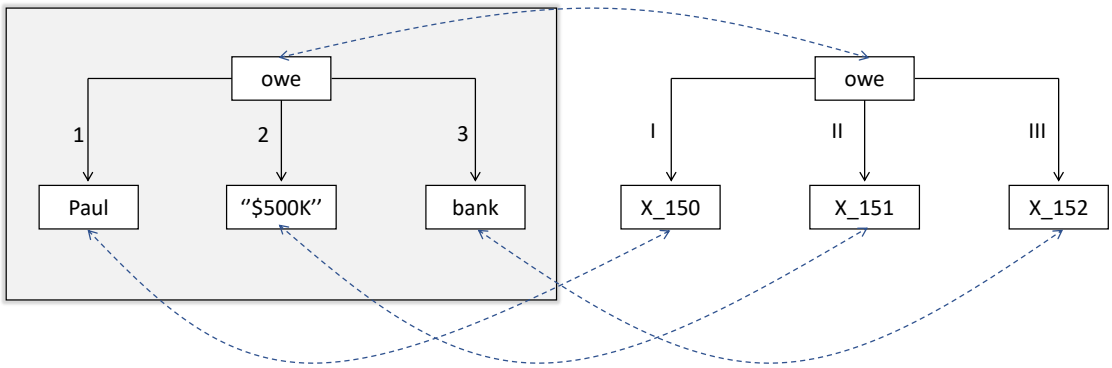


FIGURE 2.12. Application de *actant_gp*

2.3.1.4. *Application des règles* lex_class *et* lex_standard

L'étape précédente a généré des arcs contraints en partance de la racine. Les nœuds de ces arcs devront donc être lexicalisés afin de poursuivre la construction de l'arbre (voir la figure 2.13). GenDR appliquera deux règles de lexicalisation pour réaliser les lexèmes correspondant à 'Paul', 'bank' et '$500'. D'abord, la règle *lex_class* sera utilisée puisque 'Paul' et '$500' affichent, respectivement, les traits `class=proper_noun` et `class=amount` dans l'input sémantique. Cette règle fait en sorte que GenDR passe directement au *lexicon*,

puisque les sens ne sont pas décrits dans le *semanticon*. L'étiquette du nœud sémantique est directement copiée en syntaxe puis GenDR s'assure finalement que les traits de PAUL et $500 respectent les contraintes des nœuds générés par la règle précédente. Ensuite, la règle *lex_standard* s'applique pour lexicaliser 'bank'.

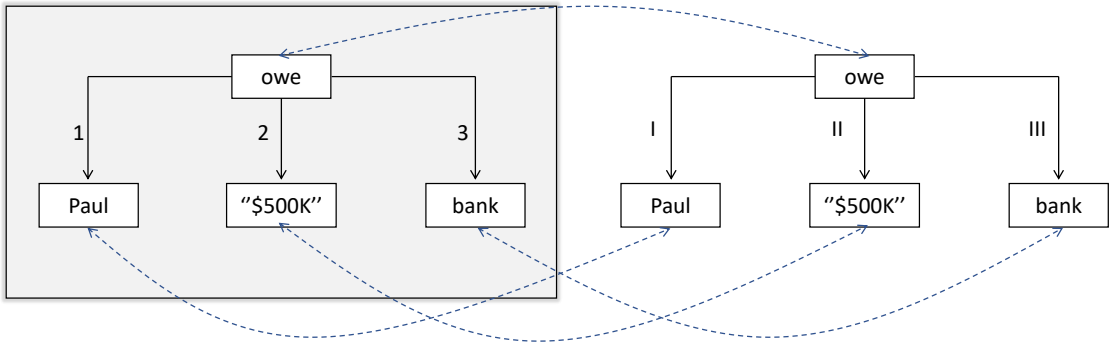


FIGURE 2.13. Application des règles de lexicalisation

2.3.2. Arborisation et lexicalisation de surface

2.3.2.1. *Application des règles de lexicalisation de surface*

On récupère les lexicalisations de surface de chacune des unités lexicales avec les règles *lex_class* (qui s'occupe des lexèmes comme PAUL) et *lex_lu*, ce qui est illustré à la figure 2.14. Il y en a deux, car il faut une règle spécifique pour les lexèmes appartenant à des classes puisque leurs traits de surface ne sont pas encodés dans leurs entrées lexicales, mais dans la classe qui leur est assignée.

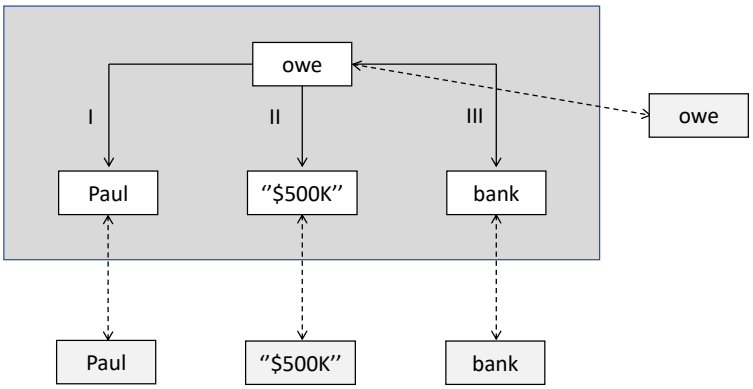


FIGURE 2.14. Application des règles de lexicalisation de surface

2.3.2.2. *Application des règles actancielles de surface*

Ces règles remplacent les étiquettes des branches de l'arbre profond (I, II, III,...) par des étiquettes de surface (sujet, objet direct, objet indirect,...). Elles sont encodées dans le GP du gouverneur de cette manière : `gp = { I = { dpos = N rel = subjective} }`. Pour remplacer l'étiquette profonde par l'étiquette de surface de l'actant I, GenDR emploie la règle *actant_subj* qui récupère dans le GP la valeur `subjective` et fait le remplacement nécessaire. Ensuite, il y a une règle qui s'occupe de la la relation objective directe. Celle-ci fonctionne de la même manière que la règle subjectale sauf qu'elle récupère la valeur de l'actant II et réalisera la valeur `dir_objective`.

Finalement, les relations indirectes/obliques sont à part car elles nécessitent la réalisation de nœuds intermédiaires en surface. C'est la règle *actant_prep* qui se charge de créer un nœud intermédiaire en syntaxe de surface permettant d'accueillir le lexème fonctionnel nécessaire à la bonne formation de la phrase. La préposition est directement encodée dans le GP du verbe, de cette manière : `gp = {III = { dpos = N rel = indir_objective prep = to} }`.

2.3.2.3. *Application de la règle des déterminants*

La règle *det_def* ajoute les déterminants aux lexèmes en fonction de leur définitude. Parmi les règles que nous avons présentées, c'est la seule règle de GenDR qui est propre à l'anglais. Dans l'exemple présent, elle lexicalise THE puisque l'unité sémantique 'bank' était marquée par le trait défini dans la structure d'input.

La figure 2.15 démontre l'application des règles actancielles et de la règle des déterminants.

2.4. Problématique

L'exemple (section 2.3) démontre que GenDR modélise bien les phénomènes linguistiques profonds. Toutefois, le dictionnaire lexical de ce système ne couvre que les 1 500 lexies les plus fréquentes (en anglais et en français). Parmi celles-ci, 500 sont des verbes, ce qui est non-négligeable, certes, mais il est clair que GenDR gagnerait en couverture s'il se dotait d'une ressource lexicale exhaustive décrivant les comportements des verbes. Nous pensons que de telles ressources sont nécessaires puisque les verbes contrôlent la structure de la plupart des énoncés et ils démontrent une grande variété quant à leur comportement syntaxique (qui est très imprévisible). Ainsi, en détenant les propriétés lexicales des verbes d'une langue donnée, on peut couvrir une grande partie des constructions syntaxiques possibles pour cette langue.

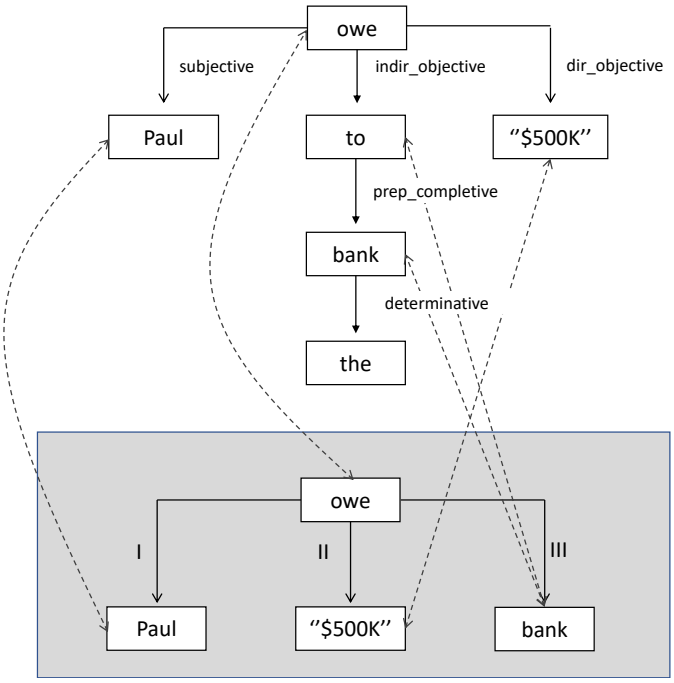


FIGURE 2.15. Application des règles actancielles de surface et des déterminants

> In particular, since verbs often convey the main idea of a sentence, such a resource must represent verb meanings. These require a particularly precise and well deffined representation that captures both their predicate-argument structure as well as their semantic content.
>
> Schuler (2005)

Nous sommes en accord avec cette déclaration et nous ne sommes pas les seuls : Korhonen *et al.* (2006) suggèrent aussi que la modélisation informatique des langues naturelles passe par la création de dictionnaire renfermant les propriétés syntaxiques des verbes. Notre objectif est d'intégrer au réalisateur profond GenDR un dictionnaire de cadre syntaxique des verbes de la langue anglaise. De plus, comme GenDR se veut multilingue, si l'expérience fonctionne bien, l'objectif serait d'acquérir d'autres ressources lexicales similaires dans d'autres langues afin d'exploiter le réalisateur au maximum de sa capacité.

2.5. PATRONS DE RÉGIME

Ce que plusieurs appellent des "comportements syntaxiques" sont encodés dans des patrons de régime en TST. D'abord, il faut préciser que ce qu'on appelle GP se nomme de différentes manières selon le cadre théorique utilisé : cadre valenciel, valence, cadre de sous-catégorisation, cadre syntaxique ou schéma de régime. Selon Milićević (2009), les GP décrivent les cooccurrences syntaxiques d'un lexème avec les actants qu'il régit, par exemple,

la relation entre un verbe et son sujet, ou bien la relation entre un nom et le complément qu'il sélectionne. Autrement dit, les GP d'une lexie correspondent à l'ensemble des constructions syntaxiques que la lexie régit. On encode ces constructions syntaxiques dans un dictionnaire, car les GP des lexèmes sont généralement imprévisibles. En effet, on ne peut pas prédire le nombre d'actants qu'un prédicat gouverne, ou bien les prépositions qu'il régit. En ce sens, même des verbes sémantiquement proches ne possèderont pas nécessairement les mêmes constructions syntaxiques, ce qui est illustré par Milićević (2009) : *on se souvient de X*, mais *on se rappelle X*.

> Comme le choix lexical détermine le choix des constructions syntaxiques possibles (la lexie sélectionnée « amène » avec elle son régime) et vice-versa (le choix d'une structure impose certains choix lexicaux), on peut dire que c'est le schéma de régime qui, au sein d'un [modèle Sens-Texte], fait le pont entre le lexique et la grammaire.
>
> (Milićević, 2009, p. 105)

À la section 2.2, nous avions parlé de l'interface sémantique-syntaxe et des règles de correspondance sémantiques qui font la transition RSem–RSyntP. Une partie du rôle du GP est d'encoder les informations lexicales nécessaires pour que les règles de correspondances sémantiques génèrent des arbres profonds grammaticaux.

En effet, les GP contiennent l'information pertinente à l'arborisation (et à la lexicalisation) parce qu'ils renferment les données sur la diathèse d'un lexème. La diathèse indique la correspondance entre les actants sémantiques d'une lexie et ses actants syntaxiques profonds. Elle est spécifiée dans les entrées lexicales d'un dictionnaire de cette manière : `{1 = I, 2 = II, etc.}`. Dans ce scénario, le premier actant sémantique correspond au premier actant syntaxique et ainsi de suite (il s'agit d'une diathèse triviale), comme dans *Paul aime le ski*, où le sémantème 'aimer' et le lexème AIMER placent leurs actants dans le même ordre. Le premier actant sémantique est 'Paul' et le second 'ski', puis l'ordre entre ces actants est conservé en syntaxe profonde. Toutefois, si on lexicalise 'aimer' par le lexème PLAIRE, alors la diathèse est différente : `{ 2 = I, 1 = II }`. Cela n'affecte en aucun cas le sens de l'énoncé : 'Le ski plaît à Paul'='Paul aime le ski'. Ensuite, au niveau du passage à la syntaxe de surface, le GP encode aussi la réalisation des actants syntaxiques en relation de surface où on passe d'une étiquette en chiffre romains à une étiquette superficielle : `{ I = { rel = subjective } }`, l'actant `I` correspond au sujet.

De plus, le GP encode les conditions dans lesquelles une structure syntaxique est exprimée. Par exemple, c'est dans le GP qu'on précise que le troisième actant syntaxique doit être de telle partie du discours et qu'il sélectionne telle préposition. Ces conditions décrivent la combinatoire lexicale et syntaxique d'une

lexie donnée. Par exemple, dans le dictionnaire de GenDR, cela est encodé ainsi : `{ gp = {III = { dpos = N rel = indir_objective prep = to} } }`. Ainsi, grâce à ces restrictions, les règles de correspondances construiront des arbres bien formés.

Également, l'une des raisons qui nous a poussé à nous tourner vers une ressource lexicale verbale provient des limites du logicel MATE quant à l'encodage de multiples patrons de régime pour une même entrée [1]. Par exemple, pour un verbe donné, on ne peut pas avoir deux parties du discours différentes compétitionnant pour la même position syntaxique. Autrement dit, si nous voulions exhaustivement représenter les comportements du verbe WANT, il nous faudrait un GP qui puisse tenir compte du fait que le second actant syntaxique de ce verbe peut avoir une DPOS de type verbal ou nominal : *I want to eat* vs *I want a dog*. Cela nous était impossible à encoder dans MATE avec les paramètres que nous avions car le système ne nous laissait pas donner deux versions de l'actant syntaxique II de WANT. Ce qui s'offrait à nous comme solution inélégante était de créer deux verbes WANT qui encoderaient séparément ces comportements syntaxiques afin que notre système puisse réaliser toutes les variantes possibles. Cependant, cela n'était pas viable puisque notre dictionnaire aurait été peuplé inutilement d'entrées verbales quasi identiques se distinguant uniquement par leur régime. De plus, l'inélégance de cette solution provenait du fait qu'il est illogique de construire un lexique où chaque entrée verbale équivaut à un patron de régime (ex : WANT avec objet direct nominal et WANT avec objet direct verbal). Ce n'est donc pas viable car pour générer du texte, il faudrait préciser dans nos données d'input à quel WANT on fait référence. Cela devient rapidement désagréable lorsqu'un verbe contient des patrons de régimes multiples.

Nous nous sommes alors tournés vers l'idée d'ajouter un dictionnaire supplémentaire à notre ressource, qui encoderait tous les régimes existants de l'anglais. Puis, nous n'aurions qu'à encoder l'identification des GP dans les unités lexicales appropriées, ce qui nous permettait de contourner le problème des GP multiples.

En résumé, nous enrichissons considérablement le contenu lexical de notre dictionnaire en plus de trouver une solution au problème mentionné ci-haut. Pour trouver la bonne ressource, nous avons regardé les candidats qui s'offraient à nous, puis nous avons choisi celui qui correspondait le plus à nos besoins et qui pouvait s'intégrer le mieux à notre système. Le chapitre suivant est dédié à ces ressources lexicales, et nous décrirons celui que nous avons choisi, VerbNet.

1. Pour bien présenter le problème, il nous aurait fallu présenter le fonctionnement de MATE en détail dès le début, mais comme c'est relativement complexe, nous présenterons plus en détail le système et ses enjeux dans les chapitres 4 et 5.

Chapitre 3

VERBNET ET LES DICTIONNAIRES DE VERBES

Avant de parler de VerbNet, nous expliquerons pourquoi nous l'avons choisi parmi tant de candidats possibles. Dans la section qui suit, nous ferons un bref survol de ces candidats. Nous avons regardé les composantes de : WordNet, FrameNet, XTAG, LCS, Comlex, Valex,et le VDE. Parmi ces dictionnaires, il y en a qui traitent d'autres parties du discours tandis que certains ne traitent que des verbes.

3.1. WordNet

Wordnet (Fellbaum, 1998) [1] est une base de données lexicales traitant les verbes, noms, adjectifs et adverbes de la langue anglaise. C'est un large réseau lexical qui ressemble à un thésaurus car il regroupe les lexies en fonction de leur sens. WordNet propose une désambiguïsation exhaustive des vocables et contient 21 000 acceptions verbales dans la version de 1990 (Miller, Beckwith, Fellbaum, Gross et Miller, 1990). On réfère généralement à cette base de données comme un modèle relationnel du lexique car le sens des lexies est représenté en termes de ses relations avec d'autres lexies, qui peuvent être de nature lexicale ou conceptuelle.

WordNet regroupe les mots en ensembles de synonymes, ou *synsets*. Ce faisant, on dit que WordNet est un réseau lexical où les synsets sont les noeuds et les relations entre synsets sont les arcs. Parmi celles-ci, il peut y avoir des relations lexicales (antonymique) et des relations conceptuelles (méronymiques, hyperonymiques, hyponymiques, troponymiques). Ainsi, la toile que couvre WordNet est établie grâce à tous ces liens sémantiques et hiérarchiques entre les différents synsets de l'anglais. Cela explique aussi pourquoi on dit qu'il s'agit d'un modèle relationnel du lexique car la toile est tissée en fonction des relations sémantiques qu'ont les lexies entre elles.

1. https://wordnet.princeton.edu/

WordNet fournit des définitions et des relations sémantiques entre les lexies mais ne fournit pas explicitement de données syntaxiques. Toutefois, la base de données se justifie en soulignant que les gloses (appellées *sentence frames* en anglais) qui servent à exemplifier l'emploi d'une lexie peuvent aussi servir à déduire le comportement syntaxique de celle-ci. Bref, cela fournit implicitement des données sur les comportements syntaxiques des lexies de WordNet (Fellbaum, 2014). Comme nous voulions acquérir un dictionnaire verbal pour les patrons de régime, il nous fallait un dictionnaire qui explicite ce genre d'information lexicale, c'est pourquoi nous avons laissé de côté WordNet.

3.2. FrameNet

FrameNet [2] est une base de données construite par extraction d'information sémantique et syntaxique à partir de corpus électroniques manuellement et automatiquement annotés (Fillmore, Johnson et Petruck, 2003). Son nom provient de la théorie linguistique de la *Frame Semantics* (Baker, Fillmore et Lowe, 1998). Initialement, ce projet couvrait les domaines suivants : santé, chance, perception, communication, transactions, temps, espace, corps, motion, étapes de la vie, contextes sociaux, émotions et cognition. Pour mieux comprendre le fonctionnement de FrameNet, nous devons présenter les bases de la théorie des *Frame Semantics*. L'idée centrale derrière celle-ci est que le sens des lexies doit être décrit en termes de cadres sémantiques qui décrivent les intéractions sémantiques entre la lexie décrite et les participants de la situation dénotée (appelés *frame elements*). Autrement dit, FrameNet analyse le sens des lexies en faisant appel aux contextes (cadres) dans lesquels ces mots apparaissent et en explicitant les propriétés syntaxiques de ces lexies.

Les cadres sémantiques et les lexies forment les unités de base de ce système. Ainsi, FrameNet désambiguïse ses entrées en les traitant par paire de cadre sémantique-lexie. De cette manière, on pourra différencier les sens d'une lexie donnée en fonction des cadres sémantiques qui lui sont associés.

Dans FrameNet, les patrons de valence sont explicitement identifiés, contrairement à WordNet, puis ils contiennent de l'information sémantique et syntaxique. La partie sémantique est rendue par les *frame elements*. La valence sémantique est aussi notée en termes de structure argumentale (à la manière des structures logiques). Pour ce qui est de l'information syntaxique, on décrit les arguments de la lexie en leur attribuant une partie du discours (groupe nominal, groupe prépositionnel, etc.) et une fonction grammaticale (sujet, objet, etc.). De plus, FrameNet a annoté les citations extraites des corpus pour démontrer comment les patrons de valence sont instanciés dans de véritables phrases.

2. https://framenet.icsi.berkeley.edu/fndrupal/

Les données de FrameNet sont stockées dans une base de données relationnelle qui reflète les bases théoriques du projet, composée de trois modules : un dictionnaire des unités lexicales traitées, un dictionnaire des cadres sémantiques, et des exemples annotés manuellement. Finalement, nous n'avons pas pris FrameNet bien qu'il s'agisse d'un excellent candidat. En effet, cette base de données explicite très bien les comportements sémantiques et syntaxiques des unités lexicales, mais contrairement à VerbNet, elle n'est pas hiérarchisée et elle ne rassemble pas en groupe les verbes se comportant identiquement. En effet, dans FrameNet, les verbes sont classés dans des *frames* différents même lorsqu'ils présentent des comportements syntaxiques identiques. Cela est une conséquence du fait que FrameNet assigne des étiquettes (*frame elements*) aux arguments d'un prédicat en fonction du cadre sémantique dans lequel le prédicat se trouve. De cette manière, FrameNet contient une quantité beaucoup plus imposante de *frames* que VerbNet, puisque ce dernier rassemble les verbes présentant des similitudes au point de vue syntaxique. Puisque ces distinctions ne sont pas utiles pour notre travail, VerbNet correspondait plus à ce que nous cherchions.

3.3. XTAG

XTAG [3] est une grammaire d'arbres adjoints (TAG) de l'anglais développée chez Xerox (XTAG Research Group, 2001; Paroubek, Schabes et Joshi, 1992) qui offre des descriptions syntaxiques riches de 9 000 verbes. Dans ce dictionnaire, chaque unité lexicale se voit assigner un ensemble d'arbres qui décrivent les constructions syntaxiques permises pour celle-ci.

En TAG, les phénomènes linguistiques sont représentés par des arbres élémentaires qui doivent se combiner par unification (Joshi et Schabes, 1997). Dans le cas de lexies prédicatives, ces arbres élémentaires sont pourvus de branches au bout desquelles il y a des nœuds non lexicalisés (mais contraints, notamment quant à leur partie du discours) destinés à acceuillir les arguments qu'elles sélectionnent. Ce sont les contraintes sur les nœuds qui contrôlent l'unification. Il n'existe que deux types d'unifications : substitution et adjonction. La substitution permet à un arbre de se substituer à un nœud pour le combler (par exemple, un arbre représentant un groupe nominal pourra se substituer à un nœud nominal dans un autre arbre, s'il en satisfait toutes les contraintes). L'adjonction permet à un arbre de se joindre à un autre arbre sans substituer de nœud. Par exemple, un arbre élémentaire de type adverbial pourra s'adjoindre à un arbre représentant un groupe verbal. La substitution modélise la complémentation, et l'adjonction modélise la modification.

Le dictionnaire XTAG organise l'information syntaxique en regroupant les arbres TAG en familles. Un arbre individuel représente une construction syntaxique donnée, mais une

3. http://www.cis.upenn.edu/~xtag/

famille d'arbres comprend toutes les variantes syntaxiques possibles pour un arbre canonique. Ainsi, dans XTAG, chaque lexie se voit attribuer un nombre de familles d'arbres. Grâce à ce mécanisme, XTAG n'a pas à lister tous les arbres possibles permis pour une lexie verbale (Doran, Egedi, Hockey, Srinivas et Zaidel, 1994).

Bien que c'était un candidat prometteur, le formalisme dans lequel s'insère XTAG est beaucoup trop attaché à une théorie linguistique (TAG). Cela nous aurait forcé à effectuer de nombreuses manipulations de données pour les convertir en TST. Il nous fallait donc une ressource plus détachée d'une théorie linguistique, comme VerbNet.

3.4. La base de données LCS

La base de données LCS[4] de Dorr (2001) s'est construite à partir de la théorie de Jackendoff (1972, 1992), qui argumente en faveur d'une décomposition sémantique des verbes. Ceux-ci sont décrits en termes de leur structure conceptuelle lexicale. Une LCS est un graphe sémantique dont les structures syntaxiques de surface en découlent. Ces graphes sont des représentations hiérarchiques non-linéaires composées d'une tête logique (la racine du graphe), d'un sujet logique (un seul), d'arguments logiques et de modificateurs logiques. En ce qui concerne le traitement des verbes dans LCS, le verbe est la racine du graphe et les arguments du verbe (sujet et objets) sont les arguments logiques liés à la racine.

Chaque nœud des LCS ont trois attributs : type, primitif sémantique et champ. Ceux-ci permettent de contraindre les nœuds des graphes sémantiques pour que les unités lexicales sélectionnées soient les bonnes.

Une décomposition sémantique des verbes en termes de LCS explique leur propriété syntaxiques. Tel que Levin (1993) l'avait perçu, les propriétés sémantiques des verbes influencent leur comportement syntaxique. À l'intérieur de ce cadre théorique, on pense que les verbes avec des LCS similaires partagent aussi des comportements syntaxiques comme des alternances de diathèses. La base de données LCS de Dorr (2001) s'inspire fortement des travaux de Levin. Les verbes y sont rassemblés en classes verbales par le partage d'une structure LCS commune. Ainsi, tous les membres d'une classe partagent la même structure sémantique (Traum et Habash, 2000; Ayan et Dorr, 2002).

Bien que cette ressource ressemble beaucoup à VerbNet en termes d'architecture (héritage de Levin (1993)), elle possède des lacunes. LCS ne couvre pas aussi large que VerbNet en termes de cadres syntaxiques et la base de données ne désambiguïse pas les différents sens des verbes.

4. http://users.umiacs.umd.edu/~bonnie/Demos/verbs-English.lcs

3.5. Comlex

Comlex [5] est une base de données lexicales développée pour l'anglais par Grishman, Macleod et Meyers (1994). C'est dictionnaire syntaxique des verbes créé à des fins computationnelles, riche mais pas libre d'accès. Ses auteurs ont opté pour un système qui se voulait le plus neutre possible d'un point de vue théorique afin d'être utilisé par un grand nombre de systèmes. Ce dictionnaire ne traite pas uniquement les verbes, mais ce sont les 6 000 entrées verbales qui nous intéressent ici. Comlex décrit pour chaque verbe les compléments possibles qu'il peut sélectionner et il explicite les attributs propres à certaines constructions (comme le choix d'une préposition). Les entrées lexicales ont été manuellement décrites parce que les auteurs du sytème ne croyaient pas que les sytèmes automatiques d'acquisition étaient capables de traiter correctement les verbes à faibles fréquences.

Nous n'avons pas choisi cette ressource d'abord parce qu'elle est payante, mais aussi parce que dans son évaluation, Schuler (2005) soulignait que Comlex ne fait pas la distinction entre les sens des verbes.

3.6. Valex

Valex [6] (Korhonen *et al.*, 2006) est un dictionnaire de cadre de sous-catégorisation (SCF) qui contient 6 397 verbes de l'anglais. Korhonen *et al.* ont bâti cette ressource par acquisition automatique. Contrairement à Comlex, les auteurs stipulent que les dictionnaires bâtis manuellement comportent naturellement plus d'erreurs que ceux construits automatiquement. Ils soulignent aussi que la méthode automatique est moins coûteuse en termes de temps et de ressources. Ils suggèrent également que les dictionnaires manuellement construits comportent une faille cruciale : le manque d'information statistique. Grâce aux informations statistiques acquises via le traitement de corpus, on a de l'information quant à la fréquence d'utilisation d'un SCF pour un verbe donné. Finalement, ils soulignent qu'en raison du nombre d'applications TAL fonctionnant avec des méthodes probabilistes, la présence d'information statistique ajoute à leur bon fonctionnement. En ce qui nous concerne, notre application TAL fonctionne à base de règles, donc les statistiques n'ajoutent rien à notre système. Toutefois, si nous le souhaitions, MATE (Bohnet *et al.*, 2000; Bohnet et Wanner, 2010a; Bohnet *et al.*, 2007) pourrait tenir compte des statistiques lors de la réalisation linguistique, c'est une fonctionnalité qu'il possède.

Dans leur article, Korhonen *et al.* expliquent avoir utilisé le système d'acquisition de Briscoe, Carroll et Watson (2006), qui se basent sur la méthode RASP. En bref, les SCF sont

5. https://nlp.cs.nyu.edu/comlex/
6. https://ilexir.co.uk/valex/index.html

extraits grâce au système RASP à partir de textes non-annotés. Ce système segmente, étiquette puis lemmatise les données brutes. Ensuite, les SCF sont extraits des phrases analysées du corpus (qui est composé de 5 corpus différents). Finalement, un filtrage est effectué pour se débarrasser du bruit. Le dictionnaire est ainsi construit automatiquement en récupérant les verbes des corpus ainsi que les SCF qui leurs sont associés.

Dans Valex, une entrée lexicale comprend, entre autres, la combinaison d'un verbe et d'un SCF, la syntaxe des arguments et la fréquence d'utilisation du SCF. Bien que cette ressource soit intéressante, nous avons préféré nous tourner vers VerbNet en raison de son architecture hiérarchisée, qui nous est très utile. L'architecture de Valex ne nous permet pas de tirer profit du mécanisme d'héritage des traits que nous avons vu à la section 2.1.1, ce qui fait en sorte qu'en utilisant Valex, notre dictionnaire serait très lourd et saturé d'informations redondantes.

3.7. Valency dictionary of English

Le Valence Dictionary of English (VDE)[7] est un dictionnaire de valences de l'anglais de Herbst, Heath, Roe et Götz (2004) qui contient les patrons de régime de 511 verbes (il traite aussi les noms et les adjectifs). Dans ce dictionnaire, chaque entrée décrit une valence possible pour un verbe. Le tout est accompagné d'un exemple provenant de la *Bank of English* (Järvinen, 1994).

Les 511 verbes du VDE ont été choisis sur la base de leur fréquence dans la langue anglaise, de leurs propriétés complexes et de leur utilité pour des apprenants de l'anglais. Les patrons de valence qu'on retrouve dans le VDE proviennent d'une étude de corpus faite sur le COBUILD.

Lors de sa création, le VDE n'était pas destiné à des applications TAL, mais les auteurs se sont rapidement rendu compte que ce dictionnaire pourrait intéresser les linguistes computationnels. Cela a entraîné la création de l'*Erlangen Valency Pattern Bank* (Herbst et Uhrig, 2009), une ressource électronique qui liste les patrons de valence identifiés par le VDE. Les patrons y sont décrits en termes de syntaxe de surface. Le dictionnaire est divisé en deux : une liste des 511 verbes désambiguïsés et les patrons de valence leur étant associés dans un premier dictionnaire, et les patrons de valence de la langue anglaise dans un dictionnaire séparé.

Ce dictionnaire ne couvre que les verbes les plus fréquents, et nous cherchions une ressources avec une bonne couverture. Toutefois, il s'agit d'un travail manuel, donc on s'attend

7. http://www.patternbank.uni-erlangen.de/cgi-bin/patternbank.cgi

à ce qu'il ne comporte pas beaucoup d'erreurs, et on pourrait ainsi en extraire une partie pour complémenter le dictionnaire de VerbNet si tel est le besoin.

3.8. VerbNet

VerbNet [8] a été créé dans un contexte où il y avait un réel besoin pour un dictionnaire décrivant la richesse et la complexité des verbes (Kipper *et al.*, 2000). Schuler (2005) trouvait qu'il y avait un manque de lignes directrices par rapport à l'organisation des verbes dans les dictionnaires destinés à des applications TAL, et c'est pour remédier à cela qu'elle a construit VerbNet. Son dictionnaire est organisé en une hiérarchie de classes verbales héritées de Levin (1993). Nous présenterons d'abord ce classement avant de voir en détail VerbNet.

3.8.1. Classes verbales de Levin

Levin (1993) proposait une méthode de classification des verbes qui a inspiré plusieurs dictionnaires, dont VerbNet (Schuler, 2005) et la LCS database (Ayan et Dorr, 2002; Dorr, 1992). Dans sa classification, les verbes de la langue anglaise sont placés dans un nombre fini de classes verbales. L'appartenance d'un verbe à l'une d'entre elles est motivée par le partage de comportements syntaxiques communs. Levin remarquait que tout locuteur natif est conscient des alternances de diathèses possibles pour un verbe, et ce sans avoir de connaissances méta-linguistiques préalables. Ainsi, en se basant sur son intuition, Levin a tenté de délimiter tous les patrons de régime possibles pour les verbes de la langue anglaise. Lorsque plusieurs présentaient des caractéristiques communes sur le plan syntaxique, elle rassemblait ces verbes dans une classe.

Bien que son travail s'insère dans le cadre de la syntaxe, elle supposait que les verbes qui se comportent de la même façon syntaxiquement possèdent probablement des propriétés sémantiques sous-jacentes communes. Ainsi, elle démontre que deux verbes en apparence synonymiques peuvent très bien appartenir à deux classes différentes, tout comme deux verbes qui, en apparence, ne se ressemblent pas du tout, peuvent appartenir à une même classe. Bref, le classement des verbes permettait de prouver sa théorie en plus de faciliter la classification des verbes de l'anglais.

Voici un exemple tiré de la thèse de (Schuler, 2005, pp. 12–13). On prend les verbes BREAK et CUT, et on teste diverses configurations possibles pour décider s'ils appartiennent à la même classe. À prime abord, on pourrait penser que c'est le cas puisque leurs signifiés se ressemblent. 'Break' et 'cut' partagent évidemment des composantes sémantiques car le

8. http://verbs.colorado.edu/verbnet_downloads/downloads.html

sens d'altérer quelque chose est présent dans ces deux verbes. Cependant, les faits suivants nous démontrent qu'ils appartiennent à deux classes distinctes :

(1) *Transitive construction*

a. John broke the window.
b. John cut the bread.

(2) *Middle construction*

a. Glass breaks easily.
b. This loaf cuts easily.

(3) *Intransitive construction*

a. The window broke.
b. *The bread cut.

(4) *Conative construction*

a. *John broke at the window.
b. John valiantly cut at the frozen loaf, but his knife was too dull to make a dent in it.

On voit d'abord que les constructions en (1) et en (2) sont possibles pour ces deux verbes. Toutefois, en (3) et en (4), on remarque qu'ils ne partagent pas ces cadres syntaxiques : BREAK prend seulement la construction intransitive et exclut la conative, tandis que CUT prend la construction conative et exclut l'intransitive. Selon la logique de Levin, cela est dû à des différences de composantes sémantiques. Le verbe CUT décrit une série d'actions entreprises dans le but de séparer un objet en morceaux. Toutefois, il est possible de commencer à découper un objet sans que l'objet ne soit séparé. Dans ce scénario, on peut tout de même percevoir que l'objet a été découpé. En ce qui concerne BREAK, le changement d'état (le fait d'être séparé en morceaux) est ponctuel. Si on n'arrive pas au résultat final, une tentative de briser quelque chose ne peut être perçue.

Toutefois, dans ces exemples de Levin, *break* n'a pas le même sens en (3) et en (1). Dans l'exemple (3), on pourrait traduire le sens de *break* par 'se briser' tandis que le sens de *break* dans l'exemple (1) serait plutôt 'briser'. Cela a un impact direct sur la syntaxe, puisque le premier sens ne peut prendre qu'un seul argument, tandis que le second en prend nécessairement au moins deux. Cette lacune théorique de Levin est aussi présente dans VerbNet.

Bref, le projet de Levin a inspiré beaucoup de chercheurs, notamment l'équipe de VerbNet, qui a repris une grande partie de son travail, en particulier le regroupement des verbes en une hiérarchie de classes. Toutefois, les concepteurs de VerbNet ont retravaillé l'architecture de Levin et y ont apporté des corrections et améliorations (Kipper, Korhonen, Ryant et Palmer, 2006).

3.8.2. Composantes de VerbNet

Dans VerbNet, les verbes sont regroupés en classes, chacune contenant un ensemble de membres, une liste de rôles thématiques (accompagnés de restrictions sélectionnelles) utilisés pour décrire les arguments, et un ensemble de cadres syntaxico-sémantiques. Chaque cadre est composé d'une brève description, suivi d'un exemple, puis d'une description syntaxique et sémantique (Schuler, 2005). Nous allons décrire chacune de ces composantes ci-dessous.

3.8.2.1. *Classes verbales : organisation hiérarchique*

Les auteurs de VerbNet se sont fortement inspirés de Acquilex Lexical Knowledge Base (Copestake, 1992) pour l'organisation du lexique. Acquilex ordonnait l'information lexicale en hiérarchie. VerbNet a donc aussi implémenté un aspect hiérarchique à son dictionnaire en créant jusqu'à trois niveaux de profondeur pour organiser les classes verbales.

Cela a entraîné la création des sous-classes qui héritent de tout le contenu lexical de leur classe mère. Elles ont été créées pour spécifier qu'un sous-ensemble de verbes issus d'une classe mère montrent des comportements syntaxiques différents du reste de la classe. Ceux-ci comprennent : les constructions syntaxiques, les prédicats sémantiques et les restrictions sélectionnelles sur les rôles thématiques (Schuler, 2005). Prenons un exemple tiré de VerbNet pour illustrer cette hiérarchie à plusieurs niveaux (CLEAR, 2005).

```
<VNCLASS ID="spray-9.7">
    <SUBCLASSES>
        <VNSUBCLASS ID="spray-9.7-1">
                <VNSUBCLASS ID="spray-9.7-1-1">
        <VNSUBCLASS ID="spray-9.7-2">
            <SUBCLASSES/>
        </VNSUBCLASS>
    </SUBCLASSES>
</VNCLASS>
```

FIGURE 3.1. Hiérarchie des classes verbales dans VerbNet

`Spray-9.7` (illustré à la figure 3.1) est le nom de la classe qui englobe toutes les autres ici. À l'intérieur de celle-ci, on spécifie tous les membres appartenant à cette classe, les rôles thématiques, les cadres syntaxiques et les prédicats sémantiques. Puis `Spray-9.7-1` représente un sous-ensemble de `Spray-9.7` dont les comportements syntaxiques lui sont propres . Puis, `Spray-9.7-1-1` est une sous-classe d'une sous-classe, et ainsi de suite. Elle héritera des traits de sa classe mère ainsi que de la classe qui domine sa classe mère. Finalement `Spray-9.7-2` est la classe sœur de `Spray-9.7-1` donc, elle hérite aussi des traits de `Spray-9.7` mais ne partage pas les particularités de `Spray-9.7-1`.

Tel que démontré dans l'exemple 3.1, les classes et sous-classes sont numérotées. Cette numérotation sert à expliciter la hiérarchie à l'intérieur d'une classe de VerbNet, mais elle sert aussi à regrouper des classes verbales en fonction de leur signifié. Cette numérotation est directement héritée du système de Levin (1993) et va de 9 à 109. Le numéro associé à une classe sert à représenter le partage de caractéristiques sémantiques (et syntaxiques) entre les classes qui partagent ce numéro. Par exemple, les classes signifiant 'mettre quelque chose' commenceront par le chiffre 9 :

— `put 9.1`
— `put spatial 9.2`
— `funnel 9.3`
— `put direction 9.4`
— `pour 9.5`
— `coil 9.6`
— `spray 9.7`
— `fill 9.8`
— `butter 9.9`
— `pocket 9.10`

3.8.2.2. *Membres*

Traditionnellement, les entrées lexicales dans un dictionnaire représentent un seul et unique verbe. En ce qui concerne VerbNet, les entrées sont des classes verbales regroupant plusieurs verbes à la fois. Cela permet à VerbNet de couvrir largement l'anglais sans recourir à un grand nombre d'entrées. Pour garnir leur section `<MEMBERS>`, VerbNet a puisé dans les travaux de Levin Levin (1993) et dans la base de données LCS (Ayan et Dorr, 2002), et a mené sa propre enquête pour déterminer à quelle classe verbale un verbe appartient.

Concrètement, cette information est encodée directement dans les entrées lexicales de VerbNet en *XML*. La figure 3.2 montre à quoi ressemble la section `<MEMBERS>`. On voit que DEAL, LEND, LOAN, PASS, PEDDLE et REFUND sont les membres de la classe `give-13.1`.

```
<VNCLASS ID="give-13.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="vn_schema-3.xsd">
    <MEMBERS>
        <MEMBER name="deal"/>
        <MEMBER name="lend"/>
        <MEMBER name="loan"/>
        <MEMBER name="pass"/>
        <MEMBER name="peddle"/>
        <MEMBER name="refund"/>
        <MEMBER name="render"/>
    </MEMBERS>
```

FIGURE 3.2. Les membres d'une classe verbale

3.8.2.3. *Rôles thématiques*

Schuler (2005) critiquait les autres dictionnaires verbaux qui n'offraient pas de contenu sémantique, c'est pourquoi VerbNet inclut 23 rôles thématiques pour identifier les arguments sélectionnés par les verbes dans chaque cadre syntaxique. Il existe d'autres approches, dont la numérotation des arguments (*Arg-1 Verbe Arg-2*) comme dans PropBank (Palmer, Gildea et Kingsbury, 2005), mais Schuler considérait que l'usage des rôles thématiques permettait d'ajouter de l'information sémantique. En effet, l'assignation d'un rôle thématique à un argument nous donne de l'information quant au type d'argument nécessaire pour un verbe donné.

À la base, les rôles thématiques ont été mis de l'avant par Fillmore (1968) et Jackendoff (1972). Toutefois, VerbNet a créé sa propre banque de rôles thématiques. Beaucoup sont inspirés de Fillmore et Jackendoff, mais de nouveaux ont été créés. Les auteurs de VerbNet précisent donc que le nombre de rôles thématiques et la qualité des rôles thématiques est assez arbitraire. Il n'y a pas de justification théorique, mais c'est ce qu'ils ont convenu d'utiliser. Schuler voulait des rôles pouvant identifier tous les arguments possibles contenus dans les patrons de régime, donc, des rôles assez génériques pour se prêter à divers cadres. Ces rôles ne sont pas spécifiques à des classes en particulier. Les rôles thématiques utilisés dans VerbNet sont : `actor`, `agent`, `asset`, `attribute`, `beneficiary`, `cause`, `location`, `destination`, `source`, `experiencer`, `extent`, `goal`, `instrument`, `material`, `product`, `patient`, `predicate`, `recipient`, `stimulus`, `theme`, `time`, `topic`.

Les rôles thématiques sont listés dans la section <THEMROLES> de chaque classe verbale, ce qu'on peut voir à la figure 3.3. Une section <THEMROLES> peut revenir dans une sous-classe lorsque celle-ci possède des rôles thématiques plus spécifiques à cette sous-classe de verbes. Ils sont ensuite mappés sur les arguments dans les cadres syntaxiques et sémantiques (qu'on peut voir aux figure 3.5 et 3.6).

```
<THEMROLES>
    <THEMROLE type="Agent">
        <SELRESTRS logic="or">
            <SELRESTR Value="+" type="animate"/>
            <SELRESTR Value="+" type="organization"/>
        </SELRESTRS>
    </THEMROLE>
    <THEMROLE type="Theme">
        <SELRESTRS/>
    </THEMROLE>
    <THEMROLE type="Recipient">
        <SELRESTRS logic="or">
            <SELRESTR Value="+" type="animate"/>
            <SELRESTR Value="+" type="organization"/>
        </SELRESTRS>
    </THEMROLE>
</THEMROLES>
```

FIGURE 3.3. Les rôles thématiques

Pour les besoins de notre travail, nous n'utilisons pas les rôles thématiques puisqu'ils sont incompatibles avec la théorie Sens-Texte. Toutefois, nous voulions souligner qu'ils avaient une importance sémantique pour les créateurs de VerbNet. Mel'čuk (2012) souligne que les rôles thématiques contiennent des failles importantes pour identifier les arguments d'un prédicat. C'est pourquoi la théorie Sens-Texte fait appel à une autre méthode : les actants sémantiques. Dans ce cadre théorique, la relation prédicat-argument est représentée en étiquettant le nœud sémantique qui lie un prédicat à son argument. Puisqu'un prédicat peut gouverner plusieurs arguments, chaque relation de dépendance se doit d'être distinguable. Une numérotation des arcs est donc effectuée pour remplir ce besoin, mais tel que (Mel'čuk, 2012, pp. 227–234) le précise, les chiffres sur ces arcs ne sont pas porteurs de sens car ils n'aident qu'à distinguer les différents arguments d'un prédicat.

3.8.2.4. *Restrictions sélectionnelles*

Les restrictions sélectionnelles s'ajoutent aux rôles thématiques. Ces traits imposent des contraintes aux arguments possibles pour un patron de régime donné. Dans l'exemple fourni à la figure 3.4, on remarquera que l'`Agent` doit être soit un être animé, soit une organisation.

```
<THEMROLES>
    <THEMROLE type="Agent">
        <SELRESTRS logic="or">
            <SELRESTR Value="+" type="animate"/>
            <SELRESTR Value="+" type="organization"/>
        </SELRESTRS>
    </THEMROLE>
```

FIGURE 3.4. Les restrictions sélectionnelles sur les rôles thématiques

3.8.2.5. *Cadres syntaxiques*

Voici maintenant la section qui nous intéresse le plus : les cadres syntaxiques, qui sont décrits dans la section `<FRAMES>` de VerbNet. À l'intérieur de cette balise, on retrouve une autre balise se nommant `<FRAME>` qui contient la balise `<SYNTAX>`. Celle-ci donne de l'information de nature syntaxique (et sémantique, via les rôles thématiques). Elle présente un patron de régime linéairement en listant les syntagmes de haut en bas selon leur ordre en surface. De plus, chaque cadre syntaxique est accompagné d'une phrase servant d'exemple.

Nous avons choisi VerbNet car nous voulions un dictionnaire qui énumère exhaustivement tous les comportements syntaxiques possibles d'un verbe. Or, cette ressource décrit explicitement comment chaque verbe se combine en surface, avec quel type d'argument, et quelle préposition est sélectionnée.

Le cadre syntaxique 3.5 ci-dessous provient de la classe verbale `give-13.1`. Ce cadre permet la réalisation de surface *They lent a bicycle to me*. THEY est le `<NP value=‘‘Agent’’>`, LEND est le `<VERB/>`, BICYCLE est le `<NP value=‘‘Theme’’>` et ME est le `<NP value=‘‘Recipient’’>`.

3.8.2.6. *Prédicats sémantiques*

Dans la revue de littérature de sa thèse, Schuler (2005) remarque que plusieurs dictionnaires pour le TAL manquent d'information sémantique. C'est pourquoi elle a inséré un segment sémantique à VerbNet : `<SEMANTICS>` (illustré à la figure 3.6). Cette section est constituée d'une suite de prédicats sémantiques. Chaque prédicat est décrit par une liste d'arguments qui sont, à leur tour, décrits par deux caractéristiques : *type* et *value*. Le cadre sémantique ci-dessous complémente le cadre syntaxique que nous venons d'exposer à la figure 3.5. Il décrit à la fois *They lent a bicycle to me* et *They lent me a bicycle*.

Cela conclut notre présentation des composantes de VerbNet. Pour plus d'informations, voir la thèse de Schuler (2005) et le guide d'annotation [9].

9. https://verbs.colorado.edu/verb-index/VerbNet_Guidelines.pdf, 15-02-18

```
            <SYNTAX>
                <NP value="Agent">
                    <SYNRESTRS/>
                </NP>
                <VERB/>
                <NP value="Theme">
                    <SYNRESTRS/>
                </NP>
                <PREP value="to">
                    <SELRESTRS/>
                </PREP>
                <NP value="Recipient">
                    <SYNRESTRS/>
                </NP>
            </SYNTAX>
```

FIGURE 3.5. Cadres syntaxiques

```
<SEMANTICS>
                <PRED value="has_possession">
                    <ARGS>
                        <ARG type="Event" value="start(E)"/>
                        <ARG type="ThemRole" value="Agent"/>
                        <ARG type="ThemRole" value="Theme"/>
                    </ARGS>
                </PRED>
                <PRED value="has_possession">
                    <ARGS>
                        <ARG type="Event" value="end(E)"/>
                        <ARG type="ThemRole" value="Recipient"/>
                        <ARG type="ThemRole" value="Theme"/>
                    </ARGS>
                </PRED>
                <PRED value="transfer">
                    <ARGS>
                        <ARG type="Event" value="during(E)"/>
                        <ARG type="ThemRole" value="Theme"/>
                    </ARGS>
                </PRED>
                <PRED value="cause">
                    <ARGS>
                        <ARG type="ThemRole" value="Agent"/>
                        <ARG type="Event" value="E"/>
                    </ARGS>
                </PRED>
            </SEMANTICS>
```

FIGURE 3.6. Section <SEMANTICS> de VerbNet

3.9. Synthèse

VerbNet est largement utilisé en TAL, notamment pour construire des graphes conceptuels automatiquement (Hensman et Dunnion, 2004), faire de l'analyse sémantique (Shi et Mihalcea, 2005), faire de la désambiguïsation (Abend, Reichart et Rappoport, 2008) et dans les systèmes de question-réponse (Wen, Jiang et He, 2008). Pfeil (2016) s'en est aussi servi en GAT dans le cadre du projet S-STRUCT. De leur côté, Mille et Wanner (2015) ont publié un court article expliquant qu'ils prévoyaient utiliser un dictionnaire de GP comme VerbNet, car ils avaient besoin d'une ressource lexicale riche pour faire de la GAT. Ils mentionnent que des réalisateurs classiques comme KPML (Bateman, 1997), SURGE (Elhadad et Robin, 1998), et RealPro (Lavoie et Rambow, 1997) auraient bénéficié d'un tel enrichissement lexical.

Comme d'autres, nous avons choisi ce dictionnaire d'abord pour son imposante couverture de la langue anglaise, avec 6 393 acceptions de 4 423 vocables, puis pour son architecture héritée de Levin (1993). De plus, les descriptions syntaxiques encodées en *XML* sont facilement exportables dans un format qui nous convient. Par le fait même, le traitement en Python devenait très accessible puisque le module NLTK [10] avait déjà fait un pré-traitement de VerbNet. Ainsi il existait des modules dont nous pouvions nous inspirer pour extraire l'information dans les balises de VerbNet.

> Rich lexical resources are particularly important for verbs that typically act as main predicates of sentences and carry key syntactic-semantic information for language understanding. For verbs, one of the richest lexical resources currently available is VerbNet.
>
> Majewska, Vulić, McCarthy, Huang, Murakami, Laippala et Korhonen (2017)

Un autre point important est qu'il existe beaucoup de ressources linguistiques à la VerbNet dans d'autres langues que l'anglais : français (Danlos, Pradet, Barque, Nakamura et Constant, 2016), portugais (Scarton et Alu, 2012), italien (Busso et Lenci, 2016), espagnol (Taulé, 2010), tchèque (Pala et Horák, 2008), mandarin (Liu, Chiang et others, 2008), ce qui est très intéressant dans le cadre d'un système multilingue comme GenDR.

Dans le chapitre suivant, nous expliquerons comment nous avons importé de VerbNet les informations dont nous avions besoin pour GenDR.

10. https://www.nltk.org/,01-06-17

Chapitre 4

IMPORTATION DE VERBNET DANS GENDR

Nous avons écrit un script en Python pour automatiser en grande partie l'importation des données de VerbNet dans GenDR. Nous avons utilisé le module *xml.etree.cElementTree*[1] pour manipuler et extraire les fichiers *XML* qui contiennent l'information de VerbNet. Ensuite, les scripts génèrent en sortie des informations dans des fichiers *.dict* que nous avons formattés pour MATE. Nous avons ainsi créé trois dictionnaires temporaires à l'aide des scripts Python : un dictionnaire qui décrit les classes verbales, un dictionnaire qui mappe les unités lexicales sur leur classe VerbNet, et un dictionnaire de patrons de régime. L'information contenue dans les deux premiers dictionnaires sera intégrée au *lexicon* de GenDR. Puis, le dictionnaire de patrons de régime est placé dans un fichier à part, ce sera le *gpcon*.

Pour mieux distinguer les dictionnaires en jeu, nous avons répertorié ceux que nous utiliserons dans la version prochaine de GenDR et ceux qui seront temporaires. En ce qui concerne GenDR, nous planifions utiliser les : *semanticon* , *lexicon* et *gpcon*. Le *semanticon* que nous utiliserons sera sensiblement le même que nous utilisions dans la version initiale de GenDR, le *lexicon* retiendra beaucoup d'éléments de sa version initiale, mais sera quelque peu modifié grâce aux ajouts de VerbNet et le *gpcon* est un nouveau dictionnaire qui contiendra la liste des patrons de régime possibles selon VerbNet.

En ce qui concerne les dictionnaires créés avec Python, nous créerons un fichier `lexicon.dict` (contenant la hiérarchie de VerbNet et les classes verbales en découlant), puis un fichier `members.dict` (contenant les verbes de l'anglais, plus de 6 000, avec leur classe verbale correspondante) et le `gpcon.dict` qui contient les patrons de régime. Parmi ces dictionnaires créés par Python, `lexicon.dict` et `members.dict` seront fusionnés pour former deux sections du nouveau *lexicon* de GenDR. De plus, celui-ci sera complémenté par le lexique (environ 1 000 lexèmes non verbaux de la langue anglaise) qui étaient déjà dans

1. https://docs.python.org/3/library/xml.etree.elementtree.html, 01-06-17

la version initiale. Le contenu et l'interaction des dictionnaires de la nouvelle version de GenDR seront expliqués en détail au chapitre suivant 5.

4.1. Préparation du nouveau *lexicon* de GenDR

Le dictionnaire lexical original de GenDR contient des classes abstraites et des entrées lexicales (voir la section 2.1.1). À ces deux sections, nous proposons d'ajouter les classes verbales de VerbNet et les membres verbaux de ces classes, deux nouvelles sections que nous identifions comme `VERBNET CLASSES` et `VERBNET MEMBERS`. Nous décrirons dans les pages suivantes comment les scripts Python que nous avons créés nous ont permis de bâtir ces nouvelles sections du *lexicon*.

D'abord, pour mieux comprendre la manière dont nous avons créé la section `VERBNET CLASSES`, il faut rappeler quelques notions de base du fonctionnement de GenDR. Au chapitre 2, nous avons décrit le mécanisme d'héritage qui façonne l'architecture du lexique (p. 22).

Ce mécanisme permet à une entrée d'hériter des traits (patrons de régime, diathèses, partie du discours, etc.) d'une classe abstraite. Cette transmission s'effectue en faisant pointer une entrée vers une classe abstraite (ex. : `like : verb_dt`). Ainsi, nous n'avons pas à réécrire pour chaque verbe transitif qu'il prend un sujet et objet direct, par exemple. Nous avons donc planifié l'importation des classes verbales de VerbNet en fonction de ce mécanisme.

Nous reprendrons ce mécanisme pour deux objectifs. D'abord, pour imiter l'architecture de VerbNet qui consiste à établir une hiérarchie entre les classes mères et leurs classes filles. Comme nous l'avons vu à la section 3.8.2.1, les classes verbales sont hiérarchisées et les traits des classes dominantes sont transmis aux classes qu'elles gouvernent. Nous voulions donc transposer cette architecture à la section `VERBNET CLASSES`. Le but était de transmettre les GP des classes mères aux classes filles grâce au mécanisme d'héritage. Puis, nous reprenons ce mécanisme pour transmettre les traits de partie du discours de la classe abstraite `VERB` afin d'éviter de répéter cette information pour chacune des classes verbales. Pour ce faire, on reprend le même concept qui lie les classes filles à leurs classes mères, mais cette fois-ci, on lie chaque classe mère à la classe abstraite `VERB` (illustré à la figure 4.1). Ainsi, comme elle contient les traits de partie du discours `dpos` et `spos`, ils seront transmis à tous les lexèmes verbaux du dictionnaire.

Ensuite, nous avons extrait les verbes compris dans toutes les classes verbales de VerbNet et nous les avons aussi soumis à ce mécanisme (figure 4.2). Ainsi, chaque membre pointe vers une classe verbale dont il héritera les traits. Par exemple, les lexèmes absorb, ingest et take in héritent tous les trois des traits spécifiés dans la classe verbale `absorb-39.8`. Cela

```
verb {
  dpos = V
  spos = verb
}
```

FIGURE 4.1. Traits de la classe abstraite `VERB`

nous permet de traiter 6 393 acceptions sans avoir à décrire leur comportement syntaxique complet, simplement en pointant vers la bonne classe, telle qu'indiquée dans VerbNet. L'ajout de nouveaux verbes consisterait essentiellement à trouver à quelle classe ils appartiennent.

```
absorb: "absorb-39.8"
take_in: "absorb-39.8"
ingest_1: "absorb-39.8"
```

FIGURE 4.2. Membres verbaux pointant vers leur classe

4.1.1. Extraction de l'architecture de VerbNet

```
<FRAMES>
        <FRAME>
            <DESCRIPTION descriptionNumber="0.2" primary="NP V NP" secondary="Basic Transitive"/>
            <EXAMPLES>
                <EXAMPLE>Cotton absorbs water.</EXAMPLE>
            </EXAMPLES>
            <SYNTAX>
                <NP value="Goal">
                    <SYNRESTRS/>
                </NP>
                <VERB/>
                <NP value="Theme">
                    <SYNRESTRS/>
                </NP>
            </SYNTAX>
        <FRAME>
         ...
```

FIGURE 4.3. Input du script : cadres syntaxiques imbriqués dans les fichiers VerbNet

```
lexicon {
"absorb-39.8": verb {
  gp = { id=NP_V_NP dia=x } // Cotton absorbs water.
  gp = { id=NP_V_NP_PP_from_source dia=x } // Cattle take in nutrients from their feed.
}
```

FIGURE 4.4. Output du script : propriétés de la classe verbale `absorb-39.8`

Nous avons divisé la description du premier script en trois blocs pour en faciliter la compréhension et nous illustrons à l'aide des figures 4.3 et 4.4 le type de données que le script prend en entrée et ce qu'il produit en sortie.

4.1.1.1. *Bloc 1 : Hiérarchie des classes verbales*

```
# BLOCK 1 : VERBNET HIERARCHY
def supers(t, i):
    ID = t.get('ID') # get the shared syntactic information.
    sc = {ID:i} # simulates the inheritance mechanism.
    subclasses = t.findall('SUBCLASSES/VNSUBCLASS') # gets all the information on the subclasses.
    if len(subclasses) > 0:
        for sub in subclasses: # If there's a subclass for a given VNCLASS,
            sc = {**sc, **supers(sub, ID)} # it'll point towards the class it's being dominated by.
    return sc
```

FIGURE 4.5. Importation de l'architecture des classes verbales

L'objectif de la fonction *supers* (présentée à la figure 4.5) est de recréer l'architecture de VerbNet pour pouvoir l'intégrer à GenDR. Cette fonction récursive récupère l'identifiant d'une classe, puis de toutes les sous-classes imbriquées sous elle. Ensuite, **la fonction** crée un dictionnaire[2] Python dont la clé correspond à l'identifiant d'une classe fille et la valeur est l'identifiant de sa classe mère. Cette configuration correspond à celle qui est nécessaire à GenDR pour l'application du mécanisme qui nous permettra de transmettre les attributs d'une classe mère à une classe fille. Par exemple, cette fonction produit `"begin-55.1-1": "begin-55.1"`, ce qui indique que `begin-55.1-1` hérite des traits de la classe `begin-55.1`.

4.1.1.2. *Bloc 2 : Création du dictionnaire de classes verbales*

L'objectif de la fonction *treeframes* (figure 4.6) est de récupérer, pour chaque classe VerbNet, l'identifiant de la classe verbale, les identifiants des patrons de régime compris sous celle-ci et une phrase exemple. Ainsi, cette fonction retourne un couple dont le deuxième élément est un couple : `(ID classe verbale, (ID GP, exemple))`, ce qui se traduit concrètement par cela `:('spray-9.7', ('NP_V_NP_destination', 'Jessica sprayed the wall'))`. Dans cet exemple, l'étiquette `NP_V_NP_destination` correspond à la fois à la description d'un patron de régime et à son identifiant.

Pour ce faire, la fonction *treeframes* récupère d'abord l'identifiant de la classe `spray-9.7`, puis les identifiants des patrons de régime qui sont encodés dans la section `<FRAME>` de cette

2. Un "dictionnaire" en Python est un type de structure de données, des paires de clé et de valeur non-ordonnées. Il ne faut pas les confondre avec les dictionnaires de langue.

```
# BLOCK 2 : EXTRACTION of GPS identification and EXAMPLES
def treeframes(t):
    ID = t.get('ID') # gets the name of the verbnet class
    z = []
    for frame in t.findall('FRAMES/FRAME'):
        description = re.sub(r"\s*[\s\.\-\ +\\\/\(\)]\s*", '_',
        frame.find('DESCRIPTION').get('primary')) # primary description = identification of a GP
        if description in exclude:
            continue
        description = re.sub('PP', 'PP_{}', description)
        preps = [p.get('value') or
               p.find('SELRESTRS/SELRESTR').get('type').upper()
               for p in frame.findall('SYNTAX/PREP')+frame.findall('SYNTAX/LEX')]
        preps = [sorted(p.split()) for p in preps] # manipulates data to insert the prep. in desc.
        examples = [e.text for e in frame.findall('EXAMPLES/EXAMPLE')] # get ex. for each desc.
        if len(preps)==1:
            description = description.format('_'.join(preps[0]))
        elif len(preps)==2:
            description = description.format('_'.join(preps[0]),
                                             '_'.join(preps[1]))
        elif len(preps)==3:
            description = description.format('_'.join(preps[0]),
                                             '_'.join(preps[1]),
                                             '_'.join(preps[2])) # inserting preps in descriptions
        z.append((description, examples))

    subclasses = t.findall('SUBCLASSES/VNSUBCLASS') # gets the root of each subclasses
    subframes = [treeframes(subclass) for subclass in subclasses] # applies function to subclasses
    subframes = sum(subframes, []) # flatten list of lists
    return [(ID, z)] + subframes # returns list of (sub)class, GP-identification and example
```

FIGURE 4.6. Création du dictionnaire de classes verbales

classe verbale. Ensuite, grâce aux expressions régulières, on manipule les identifiants de GP pour qu'ils correspondent au type de code demandé par GenDR. De plus, nous intégrons les prépositions régies de chaque patron de régime à l'intérieur de l'identifiant de celui-ci afin de les distinguer. En effet, certains identifiants de régime portaient le même nom, mais ne comprenaient pas les mêmes types de prépositions. Ainsi, en mettant les prépositions sélectionnées dans les identifiants des GP, nous pouvons les distinguer et faire en sorte que les classes verbales utilisent les bonnes prépositions. Par exemple, le patron de régime `NP_V_PP_patient` peut se distinguer par la préposition qui accompagne le groupe prépositionnel, donc pour nous assurer que les classes verbales sélectionnent les bonnes prépositions, nous les avons distinguées ainsi : `NP_V_PP_on_patient` et `NP_V_PP_from_patient`. Finalement, on récupère les exemples accompagnant chaque patron de régime pour compléter le triplet.

4.1.1.3. *Bloc 3 : Écriture des données dans le fichier* lexicon.dict

```
# BLOCK 3 : WRITING of the EXTRACTED INFORMATION in a FILE
with open('lexicon.dict','w') as f: # write the output into lexicon.dict
    f.write('lexicon {\n')
    for file in [f for f in os.listdir('verbnet') if f[-4:] == '.xml']: # open VerbNet XML files
        root = ET.parse('verbnet/'+file).getroot() # Applies the Python Element Tree module
        d = dict(treeframes(root)) # create dictionary from results of treeframes
        sc = supers(root, 'verb') # applies supers function to each file
        for c in d.keys():
            f.write('"'+c+'"')
            if sc[c] == 'verb': # all non-dominated classes point to the default verb class
                f.write(': ' +sc[c] + ' {')
            else:
                f.write(': ' +'"'+sc[c]+'"' + ' {') #dominated classes point towards their governor
            [f.write('\n gp = { id=' + gp[0] + (max(len(gp[0]), 30)-len(gp[0]))
                    *' ' + ' dia=x } // ' + ' '.join(gp[1])) for gp in d[c]]
            f.write('\n}\n') # GPs will have attributes: id and dia
    f.write('\n}')
```

FIGURE 4.7. Écriture des données dans le fichier *lexicon.dict*

L'objectif de ce dernier bloc de code (voir la figure 4.7) est de créer un fichier qui contiendra le résultat des deux fonctions sur l'ensemble des données de VerbNet. Pour procéder, nous ouvrons un fichier appellé *lexicon.dict* dans lequel nous écrirons les informations de VerbNet concernant les cadres syntaxiques et les classes verbales de VerbNet manipulés grâces aux fonctions *treeframes* et *supers*. Le produit de ce bloc correspondra à la section `VERBNET CLASSES` du nouveau *lexicon* de GenDR.

4.1.2. Création du dictionnaire des verbes

Maintenant que nous avons complété la section `VERBNET CLASSES` de notre dictionnaire, il ne nous reste qu'à peupler le dictionnaire avec les verbes que VerbNet a décrits. De cette manière notre *lexicon* pourra effectivement couvrir une partie importante du lexique anglais grâce au travail de Schuler (2005), qui a associé plus de 6 000 verbes à leur classe verbale. Lors de l'importation de ces verbes, nous procèderons aussi à leur désambiguïsation. Bien que Schuler ait distingué les différentes acceptions d'un même vocable en leur assignant des classes verbales différentes, la forme des verbes n'est pas désambiguïsée. Pour que GenDR reconnaisse chaque entrée verbale, elles doivent porter des étiquettes différentes.

4.1.2.1. *Récupérer les verbes*

L'objectif de la fonction *treemember* (illustré à la figure 4.10) est de récupérer les membres assignés à chaque classe et sous-classe verbale de VerbNet. Cette fonction récupère d'abord

```
<VNCLASS ID="absorb-39.8" >
    <MEMBERS>
        <MEMBER name="absorb">
        <MEMBER name="ingest" >
        <MEMBER name="take_in">
    </MEMBERS>
```

FIGURE 4.8. Input du script : verbes correspondant aux membres d'une classe verbale

```
abound : "swarm-47.5.1-2-1"
abrade : "other_cos-45.4"
abridge : "other_cos-45.4"
absolve : "free-80-1"
absorb : "absorb-39.8"
```

FIGURE 4.9. Output du script : lexèmes pointant vers une classe verbale

l'identifiant de la classe verbale, puis les membres qui lui sont associés. Finalement, la fonction retourne des paires de classes verbales et de membres. Par exemple, le script récupèrera deux fois le vocable ORDER, une fois comme `{order : "get-13.5.1"}`, puis comme `{order : "order-60-1"}`. À cette étape, nous venons de récupérer la désambiguïsation de VerbNet, mais elle ne s'insère pas dans notre dictionnaire puisque nous devons donner des noms différents à ces entrées. Les entrées et sorties de ce script sont présentés aux figures 4.8 et 4.9.

```
#GET MEMBERS FROM VERBNET CLASSES
def treemember(t):
    ID = t.get('ID') # get ID of the VNCLASS
    members = [m.get('name') for m in t.findall('MEMBERS/MEMBER')] # get members
    subclasses = t.findall('SUBCLASSES/VNSUBCLASS')
    submembers = []
    if len(subclasses) > 0: # if there's a subclass
        for sub in subclasses:
            submembers = submembers + treemember(sub) # get ID of the subclass and members
    return [(ID, members )] + submembers
```

FIGURE 4.10. Récupération des verbes de VerbNet

4.1.2.2. *Désambiguïser les noms des entrées*

L'objectif de ce bloc (figure 4.11) est de désambiguïser les noms des entrées des différentes acceptions d'un même vocable. Pour cela, nous avons extrait toutes les entrées homonymiques, puis nous les avons distinguées en leur assignant un numéro (ex. : `grill_1`, `grill_2`, `grill_3`). Finalement, nous avons identifié les vocables monosémiques et nous les

avons mis à part dans une liste. Nous avons ensuite fusionné les acceptions désambiguïsées et celles qui ne le nécessitaient pas dans une même liste.

```
# DISAMBIGUATE MEMBERS
files = [f for f in os.listdir('verbnet') if f[-4:] == '.xml']
members = dict(sum([treemember(ET.parse('verbnet/'+file).getroot())
 for file in files], [])) # VNCLASS : [member1, member2, etc. ]

values = sum(list(members.values()), []) # just the members of all classes

dups = {m:[ID for ID in members.keys() if m in members[ID]]
 for m in values if values.count(m)>1} # get all the duplicates

lexemes = {d[0]+'_'+str(n+1):d[1][n]
 for d in dups.items() for n in range(len(d[1]))} # enumerate all duplicates: eat_1, eat_2

unique_member = {m:ID for ID in members.keys()
 for m in values if m in members[ID] and values.count(m)==1} # get all unique lexemes

unified_dict = {**unique_member, **lexemes} # fuse both dict. to get all members disambiguated
```

FIGURE 4.11. Désambiguïsation des noms des entrées

4.1.2.3. *Écriture des verbes dans un fichier* .dict

L'objectif de ce bloc (figure 4.12) est de créer le dictionnaire *members.dict*, qui sera ensuite intégré au *lexicon.dict* en tant que section `VERBNET MEMBERS`. Pour ce faire, nous classons alphabétiquement les verbes, puis nous assignons à chaque acception sa classe verbale (ex. : `order_1 : "get-13.5.1"`) grâce à la fonction *treemember*. Finalement, nous insérons le tout dans le fichier *members.dict*

```
# WRITE MEMBERS IN A FILE
with open('members.dict','w') as f: # open a file
    f.write('members {\n')
    for key in sorted(unified_dict.keys()): # sort the members
        f.write(key) # write the members
        f.write(': '),
        f.write('"'+str(unified_dict[key])+'"') # point members towards ID of VNCLASS
        f.write('\n')
    f.write('\n}\n')
```

FIGURE 4.12. Écriture des verbes dans le fichier *members.dict*

4.2. CRÉATION DU DICTIONNAIRE DE PATRONS DE RÉGIME

Lors de la création de la section `VERBNET CLASSES`, nous avons extrait les classes verbales de VerbNet ainsi que les identifiants des patrons de régime qui y sont encodés. Nous

avons ensuite analysé le contenu des patrons de régime prélevés afin de trouver la meilleure manière d'encoder les propriétés lexicales et syntaxiques de chaque patron de régime dans un dictionnaire.

D'abord, nous avons dû nous débarrasser de certaines constructions syntaxiques. Nous avons ainsi filtré les GP que nous voulions importer dans un dictionnaire afin de nous départir de ceux qui pourraient poser problème et ceux que nous jugions inutiles.

Nous avons exclu les constructions purement stylistiques comme `NP_location_V_NP` qui réalisent les phrases du type *All through the mountains raged a fire*. Puis, nous avons aussi exclu les GP qui sélectionnent des constructions de type *déplacement Qu–*, comme la phrase *Heather cabled when to send the package*, car GenDR ne traite pas ces comportements pour l'instant. Il a aussi fallu enlever les constructions contenant des modificateurs adverbiaux ou adjectivaux, par exemple `NP_V_ADVP_Middle_PP_into_to_with`, qu'on retrouve dans la phrase *The computer connected* ***well*** *to the network*. L'emploi d'un groupe adverbial dans un patron de régime n'a pas sa place, car il ne s'agit pas d'un actant syntaxique et il n'est certainement pas sélectionné par le verbe. Environ une centaine de constructions syntaxiques ont ainsi été retirées. Après avoir filtré le tout, nous avons procédé à la création du dictionnaire de patrons de régime.

Nous ne pouvions pas automatiquement extraire des GP de VerbNet toutes les propriétés lexicales nécessaires pour GenDR. Comme nous l'avons vu plus tôt, un patron de régime dans GenDR indique la diathèse et la combinatoire lexicale et syntaxique d'une lexie (la partie du discours de ses actants, les relations de surface et les prépositions régies), mais les cadres syntaxiques de VerbNet ne nous donnent pas toutes ces informations. La nature des actants syntaxique est évidemment absente des cadres syntaxiques de VerbNet, puisque cette ressource utilise les rôles thématiques pour distinguer les actants liés au verbe. Toutefois, nous pouvons accéder à la partie du discours qui est encodée ainsi : `NP` correspond à `dpos=N` et `S_INF/S_ING` pour les verbes qui sélectionnent d'autres verbes correspond à `dpos=V`. VerbNet nous donne aussi accès aux prépositions régies qui sont encodées dans les cadres syntaxiques. Toutefois, les relations de surface ne sont pas explicitées dans les cadres syntaxiques, mais nous pouvons les déduire grâce aux phrases exemples accompagnant les identifiants de patrons de régime que nous avons prélevées plus tôt.

Bref, nous avons extrait le plus d'information possible des identifiants des GP. Ainsi, un identifiant comme `NP_V_NP_destination_PP_with_theme` nous donne les informations suivantes : la partie du discours de chaque actant syntaxique, l'ordre des actants syntaxiques en RSyntP et les prépositions régies par le verbe pour un actant donné. Ensuite, la phrase exemple *Jessica loaded the wagon with boxes* nous permet de déduire, de façon manuelle, les

relations de surface (sujet, objet direct, etc.) de chaque actant syntaxique. C'est grâce à ces informations que nous avons construit le *gpcon*.

À l'aide d'un petit script Python, nous avons établi qu'il y avait 274 GP uniques. Chacun sera une entrée dans le dictionnaire de patrons de régime, que nous avons écrite manuellement. Cependant, pour accélérer ce processus, nous avons défini dans Python des objets qui représentent des actants syntaxiques non-étiquetés incorporant des traits lexicaux et syntaxiques. Ensuite, nous avons défini une fonction permettant d'assigner une étiquette au contenu d'un actant syntaxique, puis la combinaison du contenu et de l'étiquette nous a permis de créer des patrons de régime assez rapidement. Le tout sera expliqué en détail dans la section qui suit.

Pour cette partie, nous avons construit le dictionnaire de patrons de régime à partir de données que nous avons entrées manuellement. Ainsi, il n'y a pas d'input XML provenant de VerbNet. Toutefois, afin de mieux comprendre la section présente, voici à quoi ressemble l'output prévu pour le script (voir figure 4.13).

```
gpcon {
NP_agent_V {
   I={rel=subjective dpos=N}
}
NP_agent_V_NP {
   I={rel=subjective dpos=N}
   II={rel=dir_objective dpos=N}
}
NP_asset_V_NP_PP_from_out_of {
   I={rel=subjective dpos=N}
   II={rel=dir_objective dpos=N}
   III={rel=oblique dpos=N prep=from}
   III={rel=oblique dpos=N prep="out of"}
}
```

FIGURE 4.13. Output prévu par le script gpcon

4.2.1. Contenu lexical et syntaxique d'un actant non-étiqueté

Dans le premier bloc, nous définissons des objets comme `subj` ou `dir_N` qui correspondent à des propriétés syntaxiques qui seraient normalement associées à un actant syntaxique. Par exemple, l'objet `subj` renvoie aux traits `'rel=subjective dpos=N'`. Nous avons ainsi défini toutes les propriétés lexico-syntaxiques possibles des actants que nous avons relevés parmi les identifiants des GP. Autrement dit, ces objets correspondent aux éléments constitutifs des GP. Ensuite, nous avons créé une fonction pour prendre ces éléments et leur assigner une étiquette selon leur position dans le GP. Cela nous évite de décrire les propriétés syntaxiques

des actants à chaque patron de régime. Finalement, nous avons décrit tous les patrons de régime en combinant ces éléments : `{'NP_agent_V_NP': [subj, dir_N],}`. Le tout est illustré à la figure 4.14.

```
# BLOCK 1 SYNTACTIC ACTANTS PROPERTIES
#subjective
subj = 'rel=subjective dpos=N'

#direct obj
dir_N = 'rel=dir_objective dpos=N'
dir_V_ING = 'rel=dir_objective dpos=V finiteness=GER'
dir_V_INF = 'rel=dir_objective dpos=V finiteness=INF'

#indirect obj
to_N = 'rel=indir_objective dpos=N prep=to'
indir_N = 'rel = indir_objective dpos = N'

#oblic
on_V = 'rel=oblique dpos=V prep=on'
to_obl_N = 'rel=oblique dpos=N prep=to'
for_obl_N = 'rel=oblique dpos=N prep=for'
against_N = 'rel=oblique dpos=N prep=against'
...
```

FIGURE 4.14. Contenu lexical et syntaxique d'un actant non-étiqueté

4.2.2. Assignation des relations syntaxiques profondes aux actants

L'objectif de la combinaison des fonctions *roman* et *gp* est d'assigner les relations syntaxiques profondes aux actants selon leur position dans un GP (voir la figure 4.15). Par exemple, dans `NP_agent_V_NP: [subj, dir_N]`, puisque `dir_N` est à la deuxième position, les propriétés qu'il renferme seront celles de l'actant syntaxique `II`. Cela permet de générer le patron de régime `NP_agent_V_NP { I={rel=subjective dpos=N} II={rel=dir_objective dpos=N} }`, qui est le format attendu dans GenDR. De plus, on dédouble le même actant syntaxique pour un GP dont l'un des actants syntaxiques se réalise à l'aide de deux prépositions différentes. Cela est illustré par le GP `NP_asset_V_NP_PP_from_out_of` qu'on a décrit à l'aide des objets `[subj, dir_N, [from_N, out_of_N]]`. Cela nous donne `III={rel=oblique dpos=N prep=from}` et `III={rel=oblique dpos=N prep="out of"}` pour le troisième actant syntaxique.

```
# BLOCK 2.1 ASSIGN SYNTACTIC LABEL to ACTANT
def roman(n):
    return ['I', 'II', 'III', 'IV', 'V', 'VI'][n-1] # transforms arabic numbers in roman numbers
def gp(name, real_actant):
    s = name + ' {\n'
    i=0
    for actant in real_actant:
        i = i+1 # starts to enumerate at 1
        if type(actant) == list: # if not actant but list, apply function to actants in list
            for y in actant:
                s = s + " " + roman(i) + "={" + y + "}\n"
        else:
            s = s + " " + roman(i) + "={" + actant + "}\n" # apply function to actant
    s = s + '}\n'
    return s

# BLOCK 2.2 CONSTRUCTION OF A GP ENTRY
descriptions = {
'NP_agent_V': [subj],
'NP_agent_V_NP': [subj, dir_N],
'NP_asset_V_NP_PP_from_out_of': [subj, dir_N, [from_N, out_of_N]],
...
```

FIGURE 4.15. Assignation des relations syntaxiques profondes aux actants et construction des patrons de régime

4.2.3. Création du *gpcon*

Finalement, il ne nous reste qu'à créer le fichier *gpcon.dict* (figure 4.16) qui renfermera tous les GP. Donc, nous appliquons la fonction *gp* à chaque description (`ex: 'NP_agent_V_NP'`) pour que le contenu des objets (`[subj, dir_N]`) se fassent assigner une étiquette syntaxique (`I ='rel=subjective dpos=N'`, `II = dir_N = 'rel=dir_objective dpos=N'`). Un exemple du contenu final de ce dictionnaire est illustré à la figure 5.5.

```
# BLOCK 3 GPCON CREATION
with open('gpcon.dict','w') as f:
    f.write('gpcon {\n')
    for d in descriptions.keys(): # for each gps descriptions,
        f.write(gp(d, descriptions[d])) # write them with the correct syntactic label
    f.write('}')
```

FIGURE 4.16. Création du dictionnaire de patrons de régime

Chapitre 5

ADAPTATION DE GENDR

Au chapitre précédent, nous avons expliqué comment nous avons extrait les informations de VerbNet à l'aide de scripts Python pour les importer dans GenDR. Ce chapitre couvrira l'adaptation de ce réalisateur pour lui faire exploiter ces nouvelles données. D'abord, nous expliquerons comment nos dictionnaires fonctionnent après l'importation de VerbNet et comment ils communiquent entre eux. Puis, nous montrerons comment fonctionnent les nouvelles règles de grammaire qui activent ces connaissances lexicales.

5.1. Adaptation des dictionnaires

Au chapitre 2, nous avons montré comment le lexique s'encodait dans la version initiale de GenDR. Nous avions deux dictionnaires : un dictionnaire de sémantèmes (*semanticon*) et un dictionnaire de lexèmes (*lexicon*). Comme nous venons de l'exposer, la nouvelle version de GenDR sera complétée par l'ajout d'un troisième dictionnaire : le *gpcon*, un dictionnaire de patrons de régime. Ce dictionnaire de GP était une conséquence de l'intégration de VerbNet à notre système.

Pour l'implémentation de ces nouvelles données lexicales dans GenDR, il y avait plusieurs avenues possibles. Nous aurions pu intégrer le tout dans le *lexicon* directement, mais il semblait logique de stocker les patrons de régime dans un autre fichier puisqu'il s'agit de données lexicales de nature différente.

Dans la version initiale de GenDR (Lambrey, 2017; Lareau *et al.*, 2018), quelques patrons de régime plus ou moins *ad hoc* étaient listés dans le *lexicon*. Comme nous voulons intégrer 278 patrons de régime, il fallait revoir la composition de ce dictionnaire et l'interaction entre les règles de correspondance sémantiques et les dictionnaires. Toutes nos règles de grammaire étaient conçues en fonction de l'emplacement des GP dans le *lexicon*. C'est pourquoi nous

avons dû refaire une partie de notre *lexicon* et nous avons dû ajouter et modifier des règles sémantiques de notre grammaire pour adpater le système à la venue de VerbNet.

5.1.1. Une nouvelle architecture pour le *lexicon*

Le *lexicon* de GenDR est maintenant séparé en quatre sections : les classes abstraites, les verbes de VerbNet, les classes verbales de VerbNet et le reste du lexique.

La première section, **`ABSTRACT CLASSES`**, décrit les classes générales de GenDR. Elle donne les propriétés génériques des verbes, noms, adverbes, adjectifs, prépositions, etc.

Dans la version originale de GenDR, les comportements syntaxiques **des verbes** étaient décrits dans cette section. Il y avait une hiérarchie rudimentaire de classes de verbes (intransitif, transitif direct/indirect et ditransitif) comprenant toute l'information syntaxique (les patrons de régime) pour chaque sous-classe. Maintenant, la classe générale `VERB` ne contient que les informations partagées par tous les verbes du dictionnaire : soit, la partie du discours profonde et celle de surface.

Les classes abstraites correspondant aux autres parties du discours (noms, adjectifs, adverbes, etc.) contiennent plus d'information que la classe abstraite `VERB`, puisque leurs comportements sont prévisibles (contrairement aux verbes). Les classes abstraites de ces catégories contiennent : la partie du discours, un patron de régime par défaut et des traits morpho-syntaxiques. Ainsi chaque nom, chaque adjectif, etc. possèdent sans exception les traits transmis par une classe abstraite décrite au début du *lexicon*. De plus, leurs informations concernant leurs comportements syntaxiques sont aussi imbriquées dans le dictionnaire de patrons de régime afin d'uniformiser l'encodage des patrons, peu importe la catégorie syntaxique. C'est pourquoi dans la figure 5.1 nous voyons que la classe abstraite `NOUN` ne possède pas d'information syntaxique explicite décrivant ses comportements syntaxiques, mais elle possède un identifiant de patron de régime `id=NP`.

La deuxième section, **`VERBNET MEMBERS`**, contient les membres des classes verbales de VerbNet que nous avons extraits (voir section 4.1.2). Sont ici listés tous les 6 393 verbes avec la classe de VerbNet (ou la sous-classe) à laquelle ils appartiennent. La figure 5.2 présente un échantillon de cette section.

La troisième section, **`VERBNET CLASSES`**, liste les types de GP que sélectionne chaque classe verbale (voir la figure 5.3). Cela se modélise par un trait `gp` dont la valeur est une structure qui contient deux attributs : la diathèse (`dia`) et l'identifiant du patron de régime (`id`). La diathèse d'un GP s'encode ainsi différement dans le nouveau *lexicon*.

```
verb {
  dpos = V
  spos = verb
}

noun {
  dpos = N
  spos = noun
  countable = yes
  gp = { id=NP dia=1}
}
...
```

FIGURE 5.1. Extrait du *lexicon* : attributs par défaut des classes génériques

```
"open up" : "establish-55.5-1"
operate : "other_cos-45.4"
oppose : "amalgamate-22.2-3"
ordain : "appoint-29.1"
order_1 : "get-13.5.1"
order_2 : "order-60-1"
organize_1 : "create-26.4"
organize_2 : "establish-55.5-1"
organize_3 : "force-59-1"
originate : "establish-55.5-1"
ornament_1 : "butter-9.9"
ornament_2 : "fill-9.8"
ornament_3 : "illustrate-25.3"
...
```

FIGURE 5.2. Extrait du *lexicon* : unités lexicales verbales

Dans la version initiale de GenDR (Lareau *et al.*, 2018), la diathèse était explicitée dans l'entrée `predicate`, qui donnait une diathèse triviale par défaut à tous les prédicats du dictionnaire, mais qu'on pouvait court-circuiter pour un lexème donné. Maintenant, on spécifie pour chaque GP la diathèse qui lui est associée de cette manière : `dia=132`, qui implique : `I:1 II:3 III:2`. Autrement dit, l'ordre de présentation des actants sémantiques (les chiffres arabes 1, 3, 2) correspondra à la réalisation syntaxique de ces actants (I, II, III). L'architecture de VerbNet ne nous permettait pas d'en extraire la diathèse, car cette ressource identifie les actants syntaxiques à l'aide des rôles thématiques. Ainsi, nous ne pouvions pas déduire la diathèse à partir des rôles, donc nous avons dû coder manuellement chaque diathèse de chaque GP. Nous avons ajouté la valeur `dia` après l'importation de VerbNet dans GenDR. Puis, chaque classe verbale est dotée d'un trait `id`, l'identifiant du patron de régime, qui permettra au système de récupérer les propriétés syntaxiques du patron de régime correspondant.

Toutefois, nous avons relevé un problème important après avoir manuellement encodé chaque diathèse. En faisant quelques tests préliminaires pour tester notre système nous en sommes venu à la conclusion que le mécanisme d'héritage ne transmettait pas toutes les informations que nous souhaitions. Il se trouve que la partie du discours se transmet sans problème, mais les patrons de régime ne percolent pas entre les classes verbales.

En effet, si un verbe pointe vers une sous-classe X, il héritera de ses patrons de régime, et il héritera de la partie du discours verbale qui est encodée dans la classe abstraite `VERB`, mais il n'héritera pas des patrons de régime encodés dans la classe mère de la sous-classe X. Le problème provient du fait que l'architecture que nous avons instaurée bloque l'héritage du trait `gp`, puisqu'on redéfinit complètement cet attribut dans la sous-classe. Ainsi, le système est incapable de récupérer les GP encodés dans la classe dominante. Quant au trait `dpos`, il a été transmis à partir de la classe abstraite `VERB` sans problème puisque le trait `dpos` n'est jamais redéfini lors de son passage entre une classe verbale puis une sous-classe. Ce problème découle ainsi d'un choix de design dans l'implémentation du logiciel MATE. Cela a eu pour conséquence que nous avons dû manuellement rajouter chaque GP de classes dominantes à classes dominées, puisque les traits `dia` avaient déjà été manuellement encodés et qu'il aurait été plus coûteux de regénérer la structure automatiquement, perdant du coup ces diathèses. Bien que toute l'architecture des classes verbales soit générée automatiquement (voir chapitre 4), nous avons tout de même décidé d'utiliser le mécanisme d'héritage pour construire notre lexique. L'avantage premier de cette méthode est qu'on garde un certain isomorphisme entre VerbNet et GenDR, puisque VerbNet est un système hiérarchisé où les classes filles héritent des traits des classes mères. Par le fait même, cela rendait notre *lexicon* moins chargé et plus facilement navigable pour un humain. Finalement, comme c'était un mécanisme que nous utilisions dans la version précédente de GenDR, nous avons cru bon de continuer à l'utiliser car cela fonctionnait sans problème initialement.

Passons ensuite à la section `NON-VERBAL LEXICAL ENTRIES` (figure 5.4) qui regroupe **le reste du lexique** : noms, adjectifs, adverbes, prépositions, déterminants, etc. Ces entrées proviennent de la version originale de GenDR (Lareau *et al.*, 2018) et le contenu de cette section a été enrichi par les lexèmes qu'on retrouvait dans les phrases exemples de VerbNet. Bref, les entrées de cette section pointent vers les classes (`NOUNS`, `PREPOSITIONS`, ...) abstraites leur correspondant, ce qui leur permet d'hériter des attributs suivants : `dpos`, `spos`, `id`, et `dia`.

```
"escape-51.1": verb {
  gp = { id=NP_V dia=1 } // The prisoners advanced.
  gp = { id=NP_V_PP_PATH_initial_loc dia=1 } // He came from France.
  gp = { id=NP_V_PP_PATH_destination dia=13 } // He came to Colorado.
  gp = { id=NP_V_PP_PATH_trajectory dia=14 } // He came through the door.
  gp = { id=NP_V_PP_PATH_initial_loc_PP_PATH_destination dia=123 } // He came from France to Colorado.
  }
"escape-51.1-1": "escape-51.1" {
  gp = { id=NP_V dia=1 } // The prisoners advanced.
  gp = { id=NP_V_PP_PATH_initial_loc dia=1 } // He came from France.
  gp = { id=NP_V_PP_PATH_destination dia=13 } // He came to Colorado.
  gp = { id=NP_V_PP_PATH_trajectory dia=14 } // He came through the door.
  gp = { id=NP_V_PP_PATH_initial_loc_PP_PATH_destination dia=123 } // He came from France to Colorado.
  gp = { id=NP_V_NP dia=12 } // The convict escaped the prison.
}
}...
```

FIGURE 5.3. Extrait du *lexicon* : classes de VerbNet

```
accountant : noun
acorn : noun
acquaintance : noun
across : preposition
...
```

FIGURE 5.4. Extrait du *lexicon* : unités lexicales non-verbales

5.1.2. Le dictionnaire des patrons de régime : *gpcon*

Le *gpcon* est un dictionnaire de patrons de régime qui contient 278 identifiants uniques de GP et leurs propriétés syntaxiques. Considérant que les classes verbales du lexicon partagent ces patrons de régime, l'option de les encoder dans un dictionnaire à part nous apparaît évidente. Cela allège grandement le contenu du lexicon et ça nous permet d'entretenir les données plus facilement. Ainsi, la modification d'un GP dans le *gpcon* est donc appliquée à toutes les classes verbales utilisant ce GP.

Une entrée typique dans ce dictionnaire (voir la figure 5.5) contient : l'identifiant du GP et ses propriétés syntaxiques qui sont explicitées par l'énumération des actants syntaxiques compris pour ce régime ainsi que les contraintes des actants (comme la partie du discours nécessaire à la lexicalisation). Nous avons aussi instauré un mécanisme pour tenir compte du fait que certains patrons de régime permettent deux prépositions en compétition pour le même actant syntaxique. C'est le cas par exemple pour le GP `NP_asset_V_NP_PP_from_out_of`, qui contient `III={rel=oblique dpos=N prep=from}` et `III={rel=oblique dpos=N prep="out of"}`. Ce mécanisme nous permet de mieux exploiter la richesse de combinatoire des verbes pour générer plus de paraphrases.

Cependant, ce dictionnaire n'est pas sans failles. Nous nous sommes rendu compte qu'il existait des doublons de GP en termes de contenu. L'origine de ces doublons provient de VerbNet, qui utilise des rôles thématiques pour identifier les actants syntaxiques, ce qui permet à deux GP d'avoir des propriétés identiques mais des identifiants différents. Par exemple, les patrons `NP_agent_V` et `NP_attribute_V` couvrent la même construction, mais la différence provient du fait que l'actant syntaxique I est de type agentif dans un cas et attributif dans l'autre. Nous avons gardé ces doublons puisqu'ils proviennent de l'architecture de VerbNet et ils pourraient nous être utiles si nous voulions faire le pont avec une autre ressource qui se sert des rôles thématiques pour identifier la diathèse.

```
NP_agent_V {
   I={rel=subjective dpos=N}
}
NP_agent_V_NP {
   I={rel=subjective dpos=N}
   II={rel=dir_objective dpos=N}
}
NP_asset_V_NP_PP_from_out_of {
   I={rel=subjective dpos=N}
   II={rel=dir_objective dpos=N}
   III={rel=oblique dpos=N prep=from}
   III={rel=oblique dpos=N prep="out of"}
}
NP_attribute_V {
   I={rel=subjective dpos=N}
}
...
```

FIGURE 5.5. Extrait du *gpcon* : liste des patrons de régime

5.2. ADAPTATION DE LA GRAMMAIRE

Les modifications apportées aux dictionnaires entraînent des modifications dans les règles de la grammaire. Nous les présenterons dans la section suivante. En ce qui concerne le module sémantique (RSem–RSyntP), l'essentiel de la version originale de GenDR a été transposé, mais nous avons dû façonner quelques nouvelles règles pour tenir compte de l'interaction entre les identifiants des patrons de régime dans le *lexicon* et leur description syntaxique dans le *gpcon*.

5.2.1. Module sémantique

Dans le chapitre 2, nous avons montré comment fonctionnait l'arborisation et la lexicalisation dans GenDR, nous montrerons maintenant comme ceux-ci interagissent dans la

nouvelle version. Pour effectuer **l'arborisation** et la **lexicalisation** (cf. 2.2.1, 2.2.2), nous reprenons des règles de la version originale et en ajoutons de nouvelles pour tenir compte des changements apportés à l'architecture de GenDR.

1. **Création de la racine**. L'application de la règle *root_standard* (que nous avons expliqué à la section 2.3) n'a pas changé, donc on construit la racine de l'arbre syntaxique à partir du nœud dominant de la structure sémantique.
2. **Lexicalisation de la racine**. Ensuite, toujours comme dans la version originale, on applique une règle de lexicalisation pour trouver dans le dictionnaire sémantique un lexème qui correspondra au sens demandé tout en respectant les contraintes imposées. Toutefois, malgré que cette étape reste identique, la nouvelle version de GenDR sollicitera beaucoup plus souvent une règle de lexicalisation de secours (cf. section 2.2.2) pour réaliser la racine puisque les 6 393 verbes de GenDR ne figurent pas tous dans le *semanticon*. En revanche, comme ils sont désambiguïsés, il s'agit généralement d'un ratio *1 pour 1* entre un lexème désambiguïsé et son sens correspondant. Bref, la règle de lexicalisation de secours s'applique puisque le sémantème correpondant au ND ne figure pas dans le *semanticon*, mais il existe dans le lexicon sous sa forme lexémique. Ainsi, le système supposera que cette entrée dans le *lexicon* exprime le sémantème dans la structure d'input.
3. **Sélection du patron de régime**. Une fois que le nœud de la racine est lexicalisé, la règle *actant_gp_selection* est déclenchée, ce qui permet à GenDR de récupérer les informations encodées dans l'entrée lexicale correspondant à la racine. Par exemple, si la racine est lexicalisée comme ORDER_1, alors le système parcourt le dictionnaire pour trouver ce membre verbal : `order_1 : "get-13.5.1"`. Ensuite, le mécanisme d'héritage permet au lexème ORDER_1 d'hériter des traits `gp` qui possèdent les informations `id` et `dia` de la classe `get-13.5.1`. Celles-ci sont ainsi récupérées par la règle et apposées sur la racine lexicalisée. Cette règle peut mener à la construction de plusieurs arbres profonds puisque le système génère autant de racines qu'il y a de traits `gp`, mais seuls les patrons de régime qui respecteront les contraintes de l'input génèreront des arbres profonds complets, les autres resteront incomplets.
4. **Application d'une règle actancielle**. À l'étape précédente, le nœud racine de l'arbre profond était enrichi des traits `id` et `dia` qui sont essentiels à l'application des règles actancielles. Pour créer les branches qui poursuivront l'arborisation du graphe sémantique, nous avons prévu six règles actancielles différentes. Chaque règle modélise un nombre d'actants syntaxiques à réaliser pour un prédicat donné. Nous les avons nommées ainsi : *actant_i* (un actant), *actant_ij* (deux actants), et ainsi de suite

jusqu'à six actants. Bien que le patron de régime avec le plus d'actants que nous avons importé de VerbNet soit de quatre, nous voulions créer un système où tous les prédicats puissent être réalisés en syntaxe.

Bref, une règle actancielle est appliquée en fonction de la diathèse qui est précisée par le trait `dia`. Autrement dit, pour que l'arborisation se fasse correctement, il faut que les actants sémantiques se trouvant dans la diathèse existent aussi dans la structure sémantique donnée. Cette règle va donc permettre uniquement l'application des GP satisfaisant les contraintes de diathèse.

Ce mécanisme est nouveau puisque dans la version originale de GenDR, le système traitait chaque relation actancielle sémantique individuellement, puis la faisait correspondre à une relation syntaxique. Cela se traduisait formellement par la création d'un arc entre la racine et un nœud vide (qui se faisait imposer des contraintes par son gouverneur, cf. section 2.3.1.3). La version actuelle de GenDR ne fonctionne plus ainsi ; maintenant le GP sélectionné déclenche une règle actancielle en fonction du nombre d'actants compris dans sa diathèse, puis la règle crée tous les arcs néssaires en fontion de la diathèse. La règle *actant_ijk* créera ainsi 3 arcs en syntaxe profonde dont les nœuds sont vides et non-contraints. Ainsi, on a changé la manière de procéder en ce qui concerne les règles actancielles, car il fallait tenir compte du fait que la diathèse était maintenant encodée en "paquets" (ex : `dia=123`), il nous fallait donc une règle de correspondance qui fasse la transition entre des "paquets" d'actants sémantiques vers des "paquets" d'actants syntaxiques.

5. **Application des contraintes sur les nœuds** Notre avons créé une règle qui s'occupait strictement d'imposer les contraintes aux noeuds nouvellement créés par les règles actancielles. Cette règle récupère les restrictions sur les nœuds qui sont encodés comme propriétés syntaxiques des GP dans le *gpcon* (voir la figure 5.8). La règle d'application de contraintes s'applique autant de fois qu'il y a de nœuds à contraindre.

6. **Lexicalisation des nœuds contraints** Ensuite, on répète la phase de lexicalisation pour récupérer les lexèmes qui pourront satisfaire les nœuds nouvellement contraints. Il s'agit de parcourir le dictionnaire lexical à la recherche du lexème correspondant à la structure sémantique d'input pour un nœud donné et dont les propriétés lexicales et morpho-syntaxiques correspondent aux contraintes imposées sur le nœud syntaxique.

   Finalement, ces lexèmes nouvellement lexicalisés déclencheront l'application de la règle 2, puis successivement jusqu'à la règle 6, et ainsi de suite jusqu'à ce que le parcours de la structure sémantique soit complété afin qu'on obtienne l'arbre syntaxique profond désiré.

5.3. Exemple

Pour illustrer le fonctionnement de la nouvelle version de GenDR, nous présenterons une phrase exemple provenant du corpus de VerbNet. Dans la présente section, nous verrons comment se réalise la phrase *The teacher talked about history to the students*. La figure 5.6 représente la structure sémantique que nous avons donnée en input au système. Le nœud dominant est 'talk_3' et il lie trois actants sémantiques : 'teacher', 'student' et 'history'. Dans cet exemple, chaque nœud reçoit les traits grammaticaux de temps, nombre et définitude appropriés.

```
structure Sem S {
  S:1{
    talk_3:1{
      tense=PAST
      1-> teacher:1
      2-> student:1
      3-> history:1
    }
    teacher:1{number=SG definiteness=DEF}
    history:1{number=SG definiteness=NO}
    student:1{number=PL definiteness=DEF}
    main-> talk_3:1
  }
}
```

Figure 5.6. Structure sémantique de *The teacher talked about history to the students*

Cet input permet de générer huit structures syntaxiques profondes, qui correspondent aux phrases suivantes :

1. *The teacher talked.*
2. *The teacher talked to the students.*
3. *The teacher talked with the students.*
4. *The teacher talked to the students about history.*
5. *The teacher talked with the students about history.*
6. *The teacher talked about history to the students.*
7. *The teacher talked about history with the students.*
8. *Te teacher talked about history.*

Bien que nous n'en visions qu'une en particulier, huit réalisations ont découlé de l'application de cet input parce que talk_3 appartient à la classe `"talk-37.5"` et il hérite

```
"talk-37.5": verb {
  gp = { id=NP_V
          dia=1 } // Susan talked.
  gp = { id=NP_V_PP_to_co_agent
          dia=12 } // Susan talked to Rachel.
  gp = { id=NP_V_PP_with_co_agent
          dia=12 } // Susan talked with Rachel.
  gp = { id=NP_V_PP_to_co_agent_PP_about_topic
          dia=123 } // Susan talked to Rachel about the problem.
  gp = { id=NP_V_PP_with_co_agent_PP_about_topic
          dia=123 } // Susan talked with Rachel about the problem.
  gp = { id=NP_V_PP_about_topic_PP_to_co_agent
          dia=132 } // Susan talked about the problem to Rachel.
  gp = { id=NP_V_PP_about_topic_PP_with_co_agent
          dia=132 } // Susan talked about the problem with Rachel.
  gp = { id=NP_V_PP_about_topic
          dia=13 } // Susan talked about the problems of modern America.
}
```

FIGURE 5.7. Classe `talk-37.5` dans le *lexicon*

donc de tous ses GP (ce qui est illustré à la figure 5.7). Toutes ces constructions ont pu être réalisées parce que les GP satisfaisaient chacune des contraintes demandées par l'input.

Afin de bien comprendre l'exemple que nous décortiquons, nous présentons à la figure 5.8 à quoi ressemble la structure interne du patron de régime que nous voulons réaliser avec GenDR. Il s'agit de celui en jeu dans la phrase *The teacher talked about history to the students*.

```

NP_V_PP_about_topic_PP_to_co_agent {
   I={rel=subjective dpos=N}
   II={rel=oblique dpos=N prep=about}
   III={rel=indir_objective dpos=N prep=to}
    }
```

FIGURE 5.8. Propriétés syntaxiques de `NP_V_PP_about_topic_PP_to_co_agent`

5.3.1. Création et lexicalisation de la racine

Application de la règle *root_standard* pour créer la racine et y imposer un verbe. Le sémantème 'talk_3' n'existe pas dans le *semanticon* donc une règle de lexicalisation de secours s'occupe de vérifier dans le *lexicon* s'il existe un lexème correspondant, puis le système trouve TALK_3 et en fait la lexicalisation de la racine, comme c'est un verbe (voir figure 5.9).

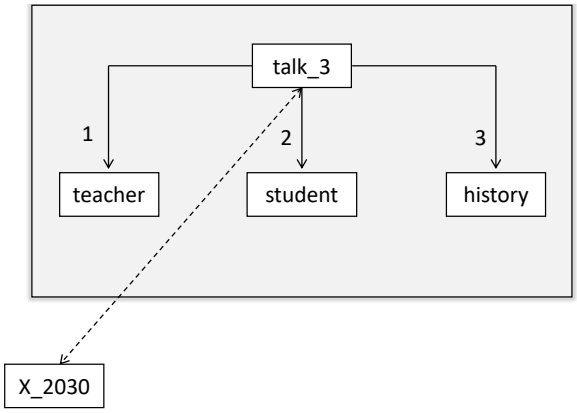


FIGURE 5.9. Création de la racine à partir du nœud dominant

5.3.2. Sélection du patron de régime dans le lexicon

Ensuite, la règle *actant_gp_selection* (figure 5.10) est déclenchée par l'apparition de TALK_3 pour satisfaire la racine. GenDR récupère les traits encodés pour chaque attribut `gp` de la classe verbale `talk-37.5`. Parmi ces `gp`, GenDR récupère ensuite le trait `id` dont la valeur est `NP_V_PP_about_topic_PP_to_co_agent` et le trait `dia` dont la valeur est `132`. Puis, il appose ces traits sur la racine.

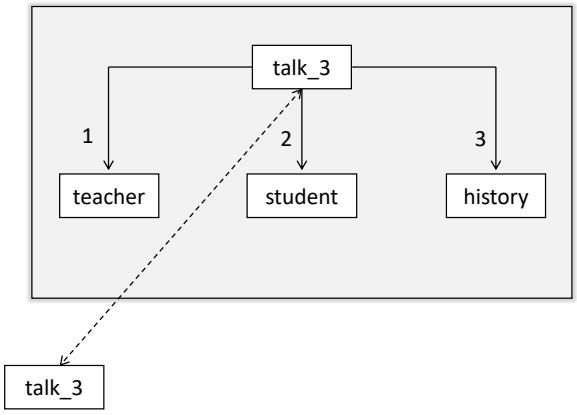


FIGURE 5.10. Sélection du patron de régime : *actant_gp_selection*

5.3.3. Application de la règle actancielle : *actant_gp_ijk*

La règle *actant_gp_ijk* est sélectionnée puisque le trait de diathèse apposé sur le nœud précise qu'il y a trois actants sémantiques (figure 5.11). À ce stade-ci, pour que l'arborisation se fasse correctement, il faut que les actants sémantiques imposés par la diathèse correspondent à ceux donnés par la structure sémantique. Comme notre exemple d'input en

contient trois, alors la règle s'applique correctement et trois arcs en partance de la racine sont créés.

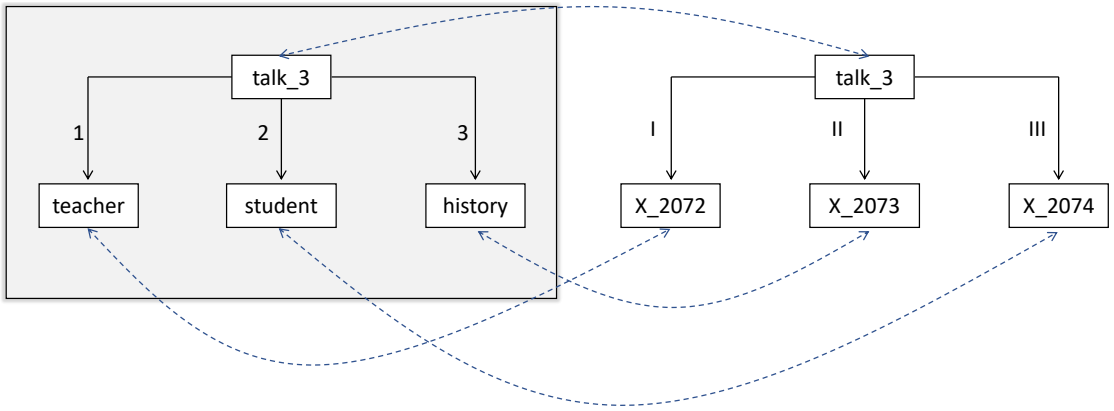


FIGURE 5.11. Application d'une règle actancielle : *actant_gp_ijk*

5.3.4. Application des contraintes sur les nœuds

La règle de contrainte récupère les restrictions sur les actants syntaxiques que sélectionne `talk-3-1.5` grâce au `id` d'un de ses `gp` qui renvoient aux propriétés du patron de régime dans le *gpcon* : `NP_V_PP_about_topic_PP_to_co_agent { I={rel=subjective dpos=N}, II=...` (voir figure 5.8). Dans ce cas, la règle d'application de contraintes s'applique trois fois puisqu'il y a trois nœuds vides.

5.3.5. Lexicalisation des nœuds contraints

On lexicalise tous les nœuds puisqu'ils en respectent les contraintes. Le système utilisera une règle de lexicalisation simple dans ce cas parce que chacun de ces sémantèmes existe dans le *semanticon* (voir la figure 5.12).

5.3.6. Application de la règle *actant_gp_selection*

Finalement, la règle *actant_gp_selection* est sollicitée pour les trois nœuds nouvellement lexicalisés TEACHER, STUDENT et HISTORY (bien que l'arbre profond semble complet à cette étape). Comme toutes les catégories syntaxiques du lexique ont des patrons de régime, le système récupère systématiquement les traits `id` et `dia` de chaque lexème dans l'arbre au cas où ceux-ci sélectionneraient des actants à leur tour. Ce qui n'est pas le cas dans cet exemple (voir la figure 5.13).

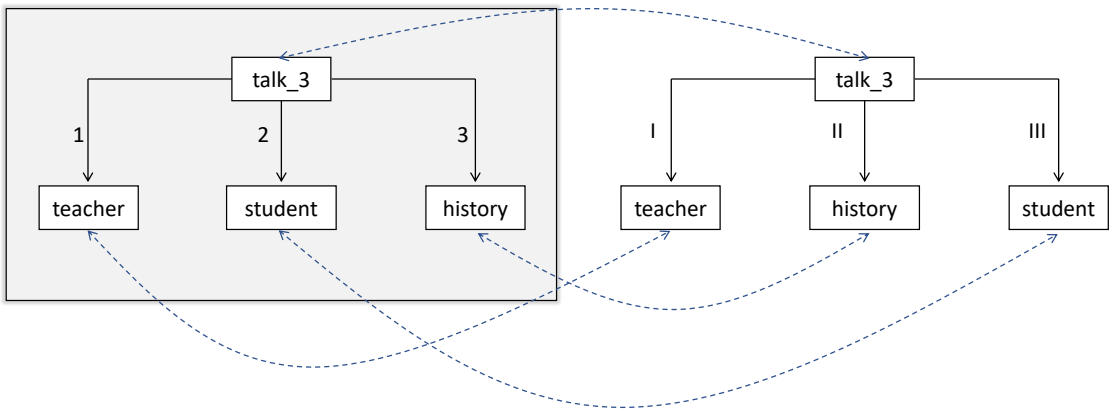


FIGURE 5.12. Lexicalisation des nœuds contraints : *lex_standard*

En ce qui concerne **l'arborisation de surface**, nous n'élaborerons pas sur cette partie puisqu'elle est quasi identique à celle que nous avons vue dans la section 2.3. Les lexicalisations de surface sont effectuées par la même règle que nous avons expliquée, de même que la règle qui introduit les déterminants. Seules les règles actancielles de surface ont quelque peu changé, car elles puisent dorénavant l'information sur la correspondance entre les actants syntaxiques et les relations de surface qu'elles imposent (et le choix des prépositions) dans le *gpcon*, alors que dans la version originale, le tout était encodé à même le *lexicon*.

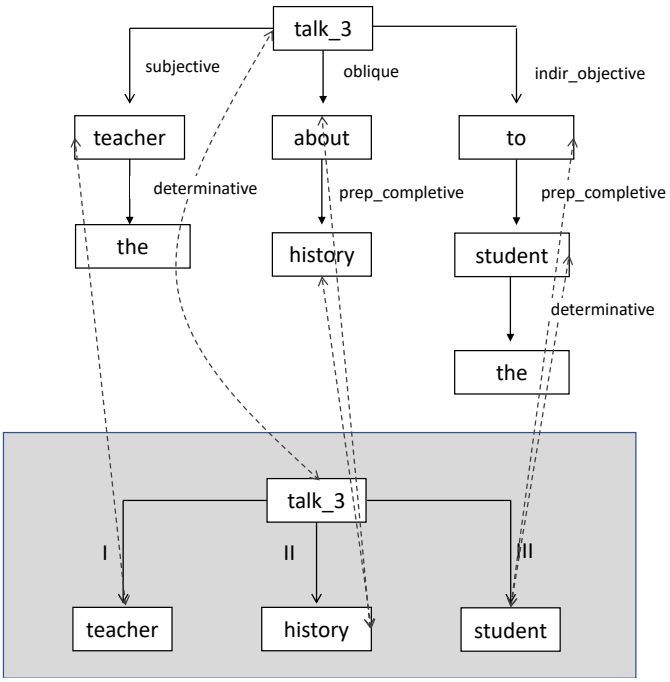


FIGURE 5.13. Applications des règles actancielles et réalisation des lexies fonctionnelles

Cela met fin à notre chapitre sur l'adaptation de GenDR. Nous passerons maintenant à la phase d'évaluation.

Chapitre 6

ÉVALUATION

Reiter et Belz (2009) expliquent qu'il existe trois types classiques d'évaluation en GAT : l'évaluation basée sur l'exécution d'une tâche, l'évaluation automatique, puis l'évaluation humaine. Nous ferons un bref survol de ces trois types pour justifier nos choix.

La méthode d'évaluation basée sur l'exécution de tâches consiste à trouver des évaluateurs qui exécuteront une tâche donnée en se basant sur des rapports rédigés automatiquement. Ainsi, il faut trouver des participants qui seront prêts à donner de leur temps pour exécuter la tâche donnée. Reiter et Belz (2009) estiment qu'il s'agit de la méthode qui évalue le mieux le contenu des réalisations d'un système de GAT puisqu'on mesure concrètement si le texte généré est de bonne qualité. Par exemple, un texte bien composé permettra aux participants de compléter la tâche rapidement, tandis qu'un texte ilisible ou très mal écrit nuira grandement à la rapidité d'exécution de la tâche et mènera potentiellement à des erreurs. Toutefois comme les auteurs le rappellent, c'est la méthode la plus coûteuse en termes de temps et de ressources puisqu'il faut trouver des participants prêts à faire cette évaluation et il faut préparer une tâche en conséquence.

Parmi les méthodes automatiques, la méthode BLEU, qui a été conçue à la base pour les systèmes de traduction automatique, est relativement commune en GAT. Par exemple, Langkilde-geary et Knight (2000) et Habash (2003) ont utilisé cette méthode pour évaluer leurs systèmes. Elle consiste à comparer le résultat à une phrase étalon préparée manuellement (ou tirée d'un corpus). Toutefois, cette méthode d'évaluation est souvent critiquée. Par exemple Mille *et al.* (2017b) soulignent que BLEU évalue bien la couverture, mais ils critiquent sa valeur dans l'évaluation de la qualité des outputs. Dans cet article, ils argumentent que leur réalisateur, FORGe, obtient un score au-dessus de la moyenne pour l'évaluation humaine, mais un score plus faible avec BLEU, et expliquent ce décalage par le fait que leur réalisateur favorise la précision par rapport au rappel.

L'évaluation humaine consiste à noter les phrases produites selons divers paramètres. Par exemple, Belz *et al.* (2011) proposent comme critères d'évaluation la clarté (qu'on juge en fonction de la facilité à lire le texte), la lisibilité (qu'on juge par la fluidité du texte : construction syntaxique étranges, erreurs grammaticales, etc.) et la similarité de sens (qui teste le paraphrasage). Cette méthode d'évaluation est plus simple à mettre en place que la méthode à base de tâches, mais elle demande plus de temps à exécuter que la méthode automatique.

Pour évaluer GenDR, nous comparerons les outputs de notre système avec les phrases du corpus de VerbNet pour vérifier si nous retrouvons exactement l'arbre syntaxique de surface correspondant à la phrase originale. Ensuite, comme GenDR incorpore un système de paraphrasage puissant, les phrases fournies en input produisent généralement plusieurs outputs à la fois. Donc, nous évaluerons chaque phrase générée, peu importe si on l'attendait ou pas, pour en vérifier la grammaticalité.

6.1. Méthodologie d'évaluation

Pour faciliter l'encodage des graphes sémantiques qui serviront d'input à nos tests, nous avons créé des scripts Python qui nous ont permis d'extraire les phrases exemples de VerbNet et de les intégrer à des fichiers *.str*. Nous avons ainsi obtenu 978 structures vides d'information sémantique, mais contenant les phrases à encoder afin d'accélérer le processus d'encodage de celles-ci. Nous devions ensuite encoder manuellement les graphes sémantiques pour qu'ils représentent les phrases que nous tenterons de générer. Pour ce faire, trois annotateurs se sont divisé la tâche, et les structures ont toutes été contre-validées par un autre annotateur.

Nous étions maintenant prêts à procéder à l'évaluation. Toutefois, comme nous étions limité dans le temps, nous avons pris 75 structures aléatoirement, dont 25 ont servi lors d'une phase de développement précédant la phase d'évaluation à proprement parler.

La phase de développement nous a permis de constater que certains noms d'entrées lexicales, appartenant à des parties du discours différentes, apparaissaient en double dans notre dictionnaire. Par exemple, les entrées pour le verbe WORK et le nom WORK avaient le même nom, donc le système n'accédait qu'à la dernière des entrées et ignorait la première. Nous avons conséquemment procédé à un filtrage pour remédier à la situation en donnant un nom différent aux entrées homonymiques. Ensuite, après la phase de développement terminée, nous avons procédé à l'évaluation sur les 50 structures restantes.

	RSyntS
Nombre de structures attendues	50
Nombre de structures générées	45
Silences dus à GenDR	2
Silences dus à VerbNet	2
Silences dus à une incompatibilité TST/VerbNet	1
Pourcentage global	90%
Pourcentage GenDR	100%

TABLEAU 6. I. Évaluation du rappel

6.1.1. Rappel

Nous avons mesuré le rappel en calculant le ratio entre le nombre de structures attendues et le nombre effectivement généré. Ainsi, chaque structure manquante à l'appel est considérée comme une erreur et le calcul se fait sur 50 puisque nous avons testé sur 50 phrases provenant du corpus de VerbNet. Les résultats sont présentés au tableau 6. I.

$$\text{Rappel} = \frac{\text{nombre de structures attendues générées}}{\text{nombre de structures attendues}}$$

Les **silences dûs à VerbNet** proviennent de deux problèmes distincts. D'abord, c'est à cause d'un patron de régime non existant pour la classe verbale `"escape-51.1"` qui correspond à la classe verbale attitrée au lexème GO_2. La structure 974, qui représente la phrase *He backed out of going on the trip* contient deux verbes : BACK OUT et GO_2, mais VerbNet n'a pas le patron de régime nécessaire au second verbe pour pouvoir réaliser *X goes on a trip*. Ainsi, GenDR était dans l'impossibilité de réaliser la structure attendue puisqu'il nous manquait des informations sur la combinatoire de GO_2. Il nous rendait des arbres de surface incomplets s'arrêtant à la lexicalisation de GO_2, sans actants. Puis, le deuxième problème hérité de VerbNet concerne la structure 267 représentant la phrase *He converted to believing in Buddha*. L'erreur vient du fait qu'il n'existe aucun patron de régime dans VerbNet pour les unités lexicales BELIEVE_1, BELIEVE_2 ou BELIEVE_3 qui puisse réaliser la forme *X believes in Y*, ce qui fait que la phrase ne peut pas être générée.

Le silence dû à une incompatibilité entre la TST et VerbNet provient de la modélisation sémantique de certains phénomènes langagiers par la TST. Une seule structure n'a pas pu être générée : 630–*She laughed in embarrassment*. L'incompatibilité est due au fait que nous avons représenté ces phrases en graphes sémantiques à la Mel'čuk (2012). Pour la structure 630, le problème provient du fait que *in embarrassment* est sémantiquement un modificateur du sens de 'laugh'. On l'a donc représenté ainsi dans notre graphe sémantique

(voir la structure 630 dans l'annexe A). Toutefois, VerbNet considère que EMBARASSMENT correspond au deuxième actant syntaxique du verbe, et que c'est un objet sélectionné par celui-ci, qui requiert une préposition lors de sa réalisation de surface. Ils ont donc créé un patron de régime pour tenir compte de cette construction, mais ce GP est inutile selon nous, puisque EMBARASSMENT n'est pas sélectionné par LAUGH selon nous. Il s'agit en fait d'une collocation réalisée grâce à la fonction lexicale $\texttt{Adv}_1$. Nous n'avons pas implémenté les fonctions lexicales dans ce projet, mais il est possible de le faire avec GenDR comme l'a démontrée Lambrey (2017) [1]. Pour cette raison nous ne l'avons pas modélisé dans le graphe d'input comme étant un actant sémantique de 'laugh' mais plutôt comme un modificateur de celui-ci. Puisque nous n'avons pas encodé de règles de correspondance permettant de réaliser la forme *in embarassment* en tant que modificateur, notre système n'a pas pu générer la phrase attendue. En revanche, nous avons généré *She laughed embarassingly*, qui est une paraphrase de la phrase attendue et que notre système permettait de réaliser.

Les silences dus à GenDR proviennent du fait que GenDR ne gère pas la coordination qui permet de créer un actant syntaxique à partir de deux actants sémantiques. Les phrases que nous n'avons pas pu générer à cause de cela sont 036–*Plays and ballets alternate* et 330–*This flyer and that flyer differ*. En effet, nous n'avons pas de mécanisme dans GenDR pouvant recréer ce type de structure en surface, donc le fait d'encoder 'play' et 'ballet' comme deux actants sélectionnés par le verbe menait nécessairement à un échec de réalisation. Par contre, nous avons pu générer *Plays alternate with ballets* et *This flyer differs from that flyer*, qui sont des paraphrases des phrases attendues. Cela démontre par le fait même la capacité de paraphrasage de GenDR.

Finalement, en ce qui concerne l'évaluation du rappel, nous n'avons pas comptabilisé comme une erreur l'impossibilité de générer des compléments du nom comme *oil's price*. GenDR ne permettait pas de réaliser ce type de forme et c'est un problème dont nous avons hérité, mais ce n'était pas l'objet de notre recherche, puisque nous cherchions à intégrer correctement les patrons de régime des verbes, alors que ce type de construction possessive relève des patrons de régime des noms. Nous les avons néanmoins notés dans l'annexe pour montrer ce que nous attendions comme sortie (section A).

1. Pour en savoir plus sur les fonctions lexicales et les collocations, nous renvoyons le lecteur à (Mel'čuk, 1996)

	DSynt	SSynt
Nombre de structures générées	106	116
Nombre de structures correctes	90	93
Erreurs dues à GenDR	0	0
Erreurs dues à VerbNet	16	23
Erreurs dues à une incompatibilité TST/VerbNet	0	0
Pourcentage global	85%	80%
Pourcentage GenDR	100%	100%

TABLEAU 6. II. Évaluation de la précision

6.1.2. Précision

Pour mesurer la précision, nous avons mesuré le ratio entre le nombre de structures grammaticalement correctes générées et le nombre total de structures générées :

$$\text{Précision} = \frac{\text{nombre de structures correctes}}{\text{nombre de structures générées}}$$

Pour l'évaluation de la précision (voir la tableau 6. II), nous avons séparé les réalisations de la RSyntP de celles de la RSyntS car les résultats différaient concernant les erreurs dues à VerbNet. Cet écart provient du fait qu'il existe certaines **prépositions régies par le verbe** qui ne sont pas encore réalisées en syntaxe profonde, mais que lorsqu'elles sont réalisées en syntaxe de surface, elles **mènent à des incohérences**. Cela est une conséquence des GP qui permettent à certains de leurs actants de se réaliser par plus d'une préposition. Le système génèrera alors autant d'arbres différents qu'il y a de choix de prépositions pour un actant donné. Mais très souvent, seule l'une d'entre elles produit une phrase grammaticale. Par exemple, en fournissant en input la structure 177, GenDR génère les phrases *The doctor cured pat of pneumonia* et **The doctor cured pat out of pneumonia*. La deuxième étant incohérente à cause de la présence de la préposition OUT OF qui rend la phrase agrammaticale. Nous constatons que sept erreurs en surface sont dues à ce phénomène.

Les erreurs que nous vous présenterons maintenant concernent les deux niveaux de représentations (RSem–RSyntP et RSyntP–RSyntS). D'abord, un type d'erreur courant de VerbNet provient de la **sélection d'un GP** qui respecte les contraintes imposées par la structure sémantique (les actants sémantiques), mais dont **le produit génère une phrase agrammaticale**. Par exemple, l'input de la structure 968–*Barry Cryer erased at the writing* comprenait les actants sémantiques `1` et `2`, ce qui a permis au patron de régime `NP_V_PP_at_Conative` de s'appliquer, générant la forme **Barry Cryer erased at the writing*. En réalité, le type de phrase que ce GP illustre normalement pour la classe verbale

`"wipe_manner-10.4.1-1"` est *Brian wiped at the counter*. De cette façon, la présence du GP (`NP_V_PP_at_Conative`), dont la diathèse correspond aux actants sémantiques demandés par l'input, a permis au GP d'être sélectionné et de réaliser un arbre syntaxique profond agrammatical.

Finalement, le dernier type d'erreur rencontré dans l'échantillon provient de **l'absence d'un verbe**. En effet, le verbe DO ne figure pas dans VerbNet. Cela n'a pas été un problème pour le rappel puisque GenDR est doté de règles de secours permettant de traiter des sémantèmes qu'il ne connaît pas. Ainsi, dans la phrase *I begged her to do it*, on a pu générer la phrase attendue car GenDR a opéré diverses réalisations dont, une correspondait à celle que nous attendions. Le système a supposé que c'était un verbe puisque le patron de régime de BEG avait mis comme contrainte sur le nœud du second actant syntaxique que ce soit un verbe. Donc, il assume que c'est bel et bien un verbe et il récupère un patron de régime assigné par défaut aux verbes, soit `NP_V_NP` (un patron de verbe transitif direct), ce qui permet à DO de sélectionner les deux actants sémantiques 'me' et 'it', le tout résultant en une bonne réalisation. Mais pour tenir compte de toutes les possibilités, GenDR suppose ensuite que c'est possiblement un nom (`dpos=N`), et cela crée un nouvel arbre profond où DO est un nom. Le résultat de cette supposition erronnée est **I begged her for the do*. Nous n'avons relevé que très peu de scénarios similaires, mais ils étaient tous liés au fait que DO ne figure pas parmi les verbes répertoriés par VerbNet. Cela est probablement dû au fait que DO est généralement utilisé comme auxiliaire en anglais, donc ils l'ont exclu de leur ressource, tout comme ils ont exclu les verbes modaux.

6.2. Synthèse

L'évaluation du rappel démontre que l'importation de VerbNet dans GenDR a été un succès (90%) et que la couverture de GenDR est dorénavant beaucoup plus grande qu'elle l'était (6 393 verbes, 274 classes verbales et 278 patrons de régime).

L'évaluation de la précision nous montre qu'il y a quelques ajustements à faire puisque l'encodage des informations dans VerbNet mène à un taux d'erreur faible, mais non-négligeable (15–20%). La raison pour laquelle les réalisateurs profonds à base de règles sont encore utiles est leur capacité à produire des phrases grammaticales, toutefois, nous démontrons ici qu'une partie des phrases réalisées par notre système sont incorrectes. Cependant, ces erreurs proviennent de VerbNet, donc notre réalisateur fonctionne bien, il n'a fait qu'appliquer les règles de grammaires et récupérer les informations dans ses dictionnaires. Pour remédier à la situation, nous pourrions corroborer les patrons de régime encodés dans VerbNet avec les patrons de régime de verbes encodés dans d'autres ressources similaires. De cette

manière nous pourrions adapter VerbNet pour qu'il n'engendre pas de structures agrammaticales. Bref, nous pensons qu'il est possible d'améliorer la précision de l'implémentation de VerbNet, mais cela demandera du temps et des ressources pour analyser les patrons potentiellement problématiques et retirer ou ajouter les patrons de régime nécessaires dans les classes verbales.

CONCLUSION

L'un des enjeux fondamentaux en GAT est de générer du texte qui paraisse aussi naturel que possible. Généralement, il faut pour cela que le système ait accès à des connaissances linguistiques fines. Par exemple, les réalisateurs à base de règles nécessiteront une grammaire qui modélise des phénomènes complexes et des dictionnaires riches encodant les propriétés de combinatoires des lexies. Si on veut couvrir le plus de constructions permises par une langue, il nous faut avoir accès aux différents comportements des lexèmes de cette langue. Certaines parties du discours présentent des comportements plus prévisibles, mais les comportements syntaxiques des verbes sont extrêmement variés et très imprévisibles. Il faut donc stocker ces données dans un dictionnaire pour pouvoir produire du texte représentant toute la richesse des langues naturelles.

C'est pour cette raison que plusieurs chercheurs ont cherché à créer des ressources lexicales décrivant les comportements syntaxiques des unités lexicales de l'anglais. Ces ressources ont pour but de servir toutes les branches du TAL, dont la GAT. L'objet de ce mémoire était de pourvoir GenDR d'une couverture linguistique beaucoup plus grande que celle qu'il a présentement en y intégrant VerbNet, qui est une base de données lexicales décrivant les comportements syntaxiques de 6 393 verbes. Ainsi, ce mémoire répond à ces deux questions :

1. *Comment intégrer une telle ressource dans un réalisateur profond à base de règles ?*
2. *Est-ce que cette intégration donne de bons résultats ?*

Dans le premier chapitre, nous avons introduit la GAT et le pipeline classique qui la compose. Nous nous sommes arrêtés sur la réalisation linguistique, qui est la dernière étape du pipeline. Ensuite nous avons souligné qu'il existait différents types d'approches pour effectuer la réalisation linguistique : à base de patrons, à base de règles et à base de modèles statistiques. Ensuite, nous avons exploré les différents réalisateurs de surface et profonds. Notamment, nous avons parlé de SimpleNLG (Gatt et Reiter, 2009), JSrealB (Molins et Lapalme, 2015b) et RealPro (Lavoie et Rambow, 1997) qui sont des réalisateurs de surface.

Puis, nous avons décrit des réalisateurs profonds : KMPL Bateman (1997), SURGE (Elhadad et Robin, 1998), MARQUIS (Wanner *et al.*, 2010), FORGe(Mille et Wanner, 2017).

Dans le deuxième chapitre, nous avons décrit en détail le réalisateur profond GenDR (Lareau *et al.*, 2018), un héritier de MARQUIS. Nous avons dépeint l'environnement dans lequel GenDR s'insère : le logiciel MATE (conçu pour la TST), qui offre un éditeur de graphes, de dictionnaire et de règles pour développer et tester une grammaire computationnelle. Ensuite nous avons expliqué quelques notions de base de la TST dont l'interface sémantique-syntaxe. Puisque GenDR opère au niveau de cette interface en modélisant l'arborisation et la lexicalisation. Finalement, nous avons démontré le fonctionnement du réalisateur GenDR à l'aide d'un exemple décrivant l'interaction des règles et dictionnaires pour générer des arbres syntaxiques de surface à partir d'un input sémantique.

Dans le troisième chapitre, nous avons fait un survol des ressources lexicales potentielles que nous envisagions pour augmenter la couverture de GenDR et sa capacité à traiter le plus grand nombre de constructions possibles. Nous avons exploré notamment WordNet, FrameNet, XTAG, LCS, Comlex, Valex,et le VDE et finalement VerbNet, le grand gagnant, qui est basé sur les travaux de Levin (1993). Ce qui nous a plu dans cette ressource est l'organisation des classes verbales ainsi que sa large couverture (plus de 6 000 acceptions).

Dans le quatrième chapitre, nous avons procédé à l'extraction des données lexicales de VerbNet. Nous avons d'abord extrait les informations syntaxiques des classes verbales en conservant la hiérarchie pensée pour VerbNet. Puis nous avons extrait les verbes associés à chaque classe verbale et nous les avons désambiguïsés avant de les ajouter à notre dictionnaire. Ensuite, nous avons procédé à la création du dictionnaire de patrons de régime à partir des données extraites. Le tout a été fait par l'entremise de Python, qui nous a permis de manipuler les fichiers de VerbNet encodés en *XML*.

Dans le cinquième chapitre, nous avons démontré comment nous avons adapté GenDR à l'utilisation d'un dictionnaire de patrons de régime. Le nombre de verbes décrit par le réalisateur est ainsi passé de 500 à 6 393. Puis, nous avons complètement changé l'utilisation des patrons de régime avec la venue du dictionnaire de GP. Cela a permis à GenDR d'avoir une couverture beaucoup plus large en termes de lexèmes verbaux et de couvrir un grand nombre de constructions syntaxiques possibles en anglais. Par le fait même, on règle le problème des GP multiples.

Dans le sixième chapitre, nous avons montré comment nous avons créé les scripts permettant de générer les structures de base pour évaluer GenDR. Nous avons sélectionné 50 structures sémantiques aléatoirement parmi les 978 encodées pour faire l'évaluation. Nous avons ainsi testé notre système en calculant le rappel et la précision. D'abord, le rappel a

été calculé en vérifiant qu'on arrivait à générer les structures syntaxiques des 50 phrases attendues. Ensuite, la précision a été calculée en vérifiant combien des structures générées étaient grammaticales. Les résultats nous montrent que les règles de GenDR fonctionnent généralement bien et que la plupart des erreurs viennent de VerbNet, notamment les erreurs affectant la précision.

Le travail que nous avons fait apporte plusieurs contributions importantes au développement de GenDR. Nous avons démontré comment s'implémente une ressource lexicale comme VerbNet dans un réalisateur profond à base de règles, de façon surtout automatique. Nous avons montré qu'il est possible de prendre une ressource relativement neutre d'un point de vue théorique et d'adapter les données pour une utilisation dans un autre cadre théorique sans que ce ne soit trop encombrant. C'est ainsi que nous avons créé un dictionnaire de patrons de régime selon le formalisme de la TST. Cela pourrait être utile à d'autres réalisateurs qui voudraient s'inspirer de cette théorie. De plus, nous avons montré que l'implémentation de VerbNet sans trop de retouches donnait un score naturel de 80% de précision, ce qui veut dire que l'intégration de cette ressource dans le cadre de la GAT nécessite quelques modifications pour éviter des phrases agrammaticales. Il reste que la couverture de GenDR s'est vue grandir d'un coup grâce à l'implémentation massive de VerbNet, ainsi il est possible de rapidement augmenter la couverture d'un réalisateur, sans trop modifier la source.

Cette recherche contribue également à évaluer la pertinence de VerbNet comme dictionnaire verbal pour un réalisateur profond en GAT. Nous en concluons que c'est une ressource que se prête très bien à cette branche du TAL, mais qu'il faudrait faire quelques modifications à la ressource post-intégration pour tenir compte des erreurs que nous avons soulignées.

De plus, cette recherche contribue à démontrer que la GAT sert à tester des ressources linguistiques (comme Danlos (1983) le soulignait). À la section 6.1.2, nous avons mentionné que les erreurs de précision provenaient de VerbNet. Plusieurs étaient dues au fait que VerbNet propose plus d'une préposition pour un même actant syntaxique. Cela a eu pour conséquence la génération de phrases comme **The doctor cured Pat* ***out of*** *pneumonia*. Nous pensons que cette surgénération est passée inaperçue jusqu'ici puisque les concepteurs de VerbNet testaient probablement leur ressource en analysant des phrases tirées de corpus. Ainsi, ce type d'erreur n'était donc pas relevé, puisque les formes agrammaticales ne se retrouvent généralement pas dans les corpus. En testant VerbNet avec la GAT, nous prouvons que les quasi-synonymes proposés dans VerbNet n'en sont pas toujours. Ainsi, l'intégration de cette ressource à un système de GAT démontre que cette branche du TAL est un bon outil pour la vérification de dictionnaires informatiques.

Ce projet de recherche démontre aussi que GenDR pourrait bénéficier d'autres ressources lexicales pour complémenter sa couverture. Notamment, nous pourrions implémenter les régimes des noms, grâce aux autres ressources (comme FrameNet Fillmore *et al.* (2003)). Nous pourrions aussi aller chercher les GP manquants dans d'autres ressources que nous avons répertoriées. De plus, comme il existe des VerbNet dans d'autres langues (francais (Danlos, Nakamura et Pradet, 2015), portugais (Scarton et Alu, 2012), italien (Busso et Lenci, 2016), espagnol (Taulé, 2010), tchèque (Pala et Horák, 2008), mandarin (Liu *et al.*, 2008)), on aurait avantage à les acquérir. La tâche serait relativement rapide puisqu'on sait déjà comment implémenter les données de ce format. Comme GenDR est un réalisateur multilingue, son architecture interne est déjà établie pour accueillir d'autres langues. Finalement, une recherche récente (Majewska *et al.*, 2017) démontre que l'exportation des classes et des membres de VerbNet (Schuler, 2005) à d'autres langues est valide et donne généralement de bons résultats. Ils soulignent qu'il faudrait évidemment faire des ajustements spécifiques pour chaque langue, mais que l'architecture pensée pour l'anglais se traduit bien à d'autres langues.

Finalement, pour que GenDR soit utilisé à pleine capacité, il faudrait lui intégrer le module de fonctions lexicales de sa version originale (Lambrey, 2017; Lareau *et al.*, 2018), ce qui enrichirait nettement sa nouvelle version grâce à la modélisation des règles permettant le traitement des collocations.

Bibliographie

Abend, O., R. Reichart et A. Rappoport. 2008, «A Supervised Algorithm for Verb Disambiguation into VerbNet Classes», dans *Proceedings of the 22Nd International Conference on Computational Linguistics - Volume 1*, COLING '08, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 9–16.

Ayan, N. F. et B. J. Dorr. 2002, «Generating A Parsing Lexicon from an LCS-Based Lexicon», dans *In : Proceedings of the LREC-2002 Workshop on Linguistic Knowledge Acquisition and Representation, Las Palmas, Canary Islands*, p. 4352.

Baker, C. F., C. J. Fillmore et J. B. Lowe. 1998, «The Berkeley FrameNet Project», dans *Proceedings of the 17th International Conference on Computational Linguistics - Volume 1*, COLING '98, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 86–90.

Bateman, J. A. 1997, «Enabling Technology for Multilingual Natural Language Generation : The KPML Development Environment», *Natural Language Engineering*, vol. 3, n° 1, p. 15–55.

Belz, A. et E. Kow. 2009, «System Building Cost vs. Output Quality in Data-to-text Generation», dans *Proceedings of the 12th European Workshop on Natural Language Generation*, ENLG '09, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 16–24.

Belz, A., M. White, D. Espinosa, E. Kow, D. Hogan et A. Stent. 2011, «The First Surface Realisation Shared Task : Overview and Evaluation Results», dans *Proceedings of the 13th European Workshop on Natural Language Generation*, ENLG '11, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 217–226.

Bohnet, B., A. Langjahr et L. Wanner. 2000, «A Development Environment for an MTT-based Sentence Generator», dans *Proceedings of the First International Conference on Natural Language Generation - Volume 14*, INLG '00, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 260–263.

Bohnet, B., F. Lareau et L. Wanner. 2007, «Automatic Production of Multilingual Environmental Information», dans *Proceedings of EnviroInfo 2007*, vol. 2, Warsaw, p. 59–66.

Bohnet, B. et L. Wanner. 2010a, «Open Soucre Graph Transducer Interpreter and Grammar Development Environment», dans *Proceedings of the Seventh Conference on International Language Resources and Evaluation (LREC'10)*, European Language Resources Association (ELRA), Valletta, Malta.

Bohnet, B. et L. Wanner. 2010b, «Open source graph transducer interpreter and grammar development environment», dans *Proceedings of the Seventh Conference on International Language Resources and Evaluation (LREC'10)*, p. 8.

Bollmann, M. 2011, «Adapting SimpleNLG to German», dans *Proceedings of the 13th European Workshop on Natural Language Generation*, Association for Computational Linguistics, Nancy, France, p. 133–138.

Briscoe, T., J. Carroll et R. Watson. 2006, «The Second Release of the RASP System», dans *Proceedings of the COLING/ACL on Interactive Presentation Sessions*, COLING-ACL '06, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 77–80.

Busso, L. et A. Lenci. 2016, «Italian VerbNet : A Construction-based Approach to Italian Verb Classification.», dans *In Proceedings of Language Resources and Evaluation Conference*, p. 10.

Cheng, H. et C. Mellish. 2000, «Capturing the Interaction Between Aggregation and Text Planning in Two Generation Systems», dans *Proceedings of the First International Conference on Natural Language Generation - Volume 14*, INLG '00, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 186–193.

CLEAR. 2005, «VerbNet Annotation Guidelines», Guideline, University of Colorado Boulder.

Copestake, A. 1992, «The ACQUILEX LKB : Representation issues in semi-automatic acquisition of large lexicons», dans *Proceedings of the 3rd Conference on Applied Natural Language Processing (ANLP-92*, p. 88–96.

Danlos, L. 1983, «Présentation d'un modèle de génération automatique», *Revue québécoise de linguistique*, vol. 13, n° 1, p. 203–228.

Danlos, L., T. Nakamura et Q. Pradet. 2015, «Traduction de VerbNet vers le français», dans *Congrès ACFAS*, ACFAS, Rimouski, Canada, p. 1.

Danlos, L., Q. Pradet, L. Barque, T. Nakamura et M. Constant. 2016, «Un Verbenet du français», *Traitement Automatique des Langues*, vol. 57, n° 1, p. 25.

Daoust, N. et G. Lapalme. 2015, «JSREAL : A Text Realizer for Web Programming», dans *Language Production, Cognition, and the Lexicon*, vol. 48, édité par N. Gala, R. Rapp et G. Bel-Enguix, Springer International Publishing, Cham, p. 361–376.

Doran, C., D. Egedi, B. A. Hockey, B. Srinivas et M. Zaidel. 1994, «XTAG System : A Wide Coverage Grammar for English», dans *Proceedings of the 15th Conference on Computational Linguistics - Volume 2*, COLING '94, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 922–928.

Dorr, B. J. 1992, «The Use of Lexical Semantics in Interlingual Machine Translation», *Machine Translation*, vol. 7, n° 3, p. 135–193.

Dorr, B. J. 2001, «LCS Verb Database», dans *Online Software Database of Lexical Conceptual Structures and Documentation*, University of Maryland.

Dras, M., F. Lareau, B. Börschinger, R. Dale, Y. Motazedi, O. Rambow, M. Turpin et M. Ulinski. 2012, «Complex predicates in Arrernte», dans *Proceedings of LFG12*, Denpasar, Indonesia, p. 177–197.

Dubinskaite, I. 2017, *Développement de Ressources Lituaniennes Pour Un Générateur Automatique de Texte Multilingue*, mémoire de maîtrise, Université Grenoble Alpes, Grenoble.

Elhadad, M. et J. Robin. 1998, «SURGE : A Comprehensive Plug-in Syntactic Realization Component for Text Generation», cahier de recherche, Ben-Gurion University, Computer Science Dept.

Elhadad, M., J. Robin et K. McKeown. 1997, «Floating Constraints in Lexical Choice», *Comput. Linguist.*, vol. 23, n° 2, p. 195–239.

Fellbaum, C., éd.. 1998, «WordNet : An Electronic Lexical Database», Language, Speech, and Communication, MIT Press, Cambridge, MA.

Fellbaum, C. 2014, «Large-scale Lexicography in the Digital Age», *International Journal of Lexicography*, vol. 27, n° 4, p. 378–395.

Fillmore, C. J. 1968, «The Case for Case», dans *Universals in Linguistic Theory*, édité par E. Bach et R. T. Harms, Holt, Rinehart and Winston, New York, p. 0–88.

Fillmore, C. J., C. R. Johnson et M. R. L. Petruck. 2003, «Background to Framenet», *International Journal of Lexicography*, vol. 16, n° 3, p. 235–250.

Gatt, A. et E. Krahmer. 2018, «Survey of the State of the Art in Natural Language Generation : Core tasks, applications and evaluation», *Journal of Artificial Intelligence Research*, vol. 61, p. 65–170.

Gatt, A. et E. Reiter. 2009, «SimpleNLG : A Realisation Engine for Practical Applications», dans *Proceedings of the 12th European Workshop on Natural Language Generation*, ENLG '09, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 90–93.

Grishman, R., C. Macleod et A. Meyers. 1994, «Comlex Syntax : Building a Computational Lexicon», dans *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*, COLING '94, Association for Computational Linguistics, Kyoto, Japan, p. 268–272.

Habash, N. 2003, «Matador : A Large-Scale Spanish-English GHMT System», dans *In Proceedings of the Ninth Machine Translation Summit*, New Orleans, USA, p. 8.

Hensman, S. et J. Dunnion. 2004, «Automatically Building Conceptual Graphs Using VerbNet and WordNet», dans *Proceedings of the 2004 International Symposium on Information and Communication Technologies*, ISICT '04, Trinity College Dublin, Las Vegas, Nevada, USA, p. 115–120.

Herbst, T., D. Heath, I. F. Roe et D. Götz. 2004, «A Valency Dictionary of English, A Corpus-Based Analysis of the Complementation Patterns of English Verbs, Nouns and Adjectives», De Gruyter Mouton, Berlin, Boston.

Herbst, T. et P. Uhrig. 2009, «Erlangen Valency Patternbank. A corpus-based research tool for work on valency and argument structure constructions.», cahier de recherche.

Iordanskaja, L., R. Kittredge et A. Polguère. 1988, «Implementing a Meaning-Text model for language generation», dans *Proceedings of the 12th International Conference on Computational Linguistics (COLING'88)*.

Jackendoff, R. 1972, «Semantic Interpretation in Generative Grammar», Cambridge : Mass., Mit Press.

Jackendoff, R. 1992, «Semantic Structures», MIT Press, Cambridge, MA.

Järvinen, T. 1994, «Annotating 200 Million Words : The Bank of English Project», dans *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*, COLING '94, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 565–568, doi :10.3115/991886.991985.

Jolkovsky, A. et I. Mel'čuk. 1967, «Essai d'une théorie sémantique applicable au traitement de langage», dans *Seconde Conférence Internationale Sur Le Traitement Automatique Des Langues, COLING 1967, Grenoble, France, August 1967*, p. 8.

Joshi, A. K. et Y. Schabes. 1997, «Tree-Adjoining Grammars», dans *Handbook of Formal Languages*, Springer, Berlin, Heidelberg, p. 69–123.

Kay, M. 1984, «Functional Unification Grammar : A Formalism for Machine Translation», dans *Proceedings of the 10th International Conference on Computational Linguistics*, COLING '84, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 75–78.

Kipper, K., H. T. Dang et M. Palmer. 2000, «Class-Based Construction of a Verb Lexicon», dans *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, AAAI Press, p. 691–696.

Kipper, K., A. Korhonen, N. Ryant et M. Palmer. 2006, «Extending VerbNet with Novel Verb Classes», dans *Proceedings of the Fifth International Conference on Language Resources and Evaluation – LREC'06 (http ://verbs.colorado.edu/ Mpalmer/Projects/Verbnet.Html)*.

Korhonen, A., Y. Krymolowski et T. Briscoe. 2006, «A large subcategorization lexicon for natural language processing applications», dans *In Proceedings of LREC*.

Lambrey, F. 2017, *Implémentation Des Collocations Pour La Réalisation de Texte Multilingue*, mémoire de maîtrise, Université de Montréal.

Lambrey, F. et F. Lareau. 2015, «Le traitement des collocations en génération de texte multilingue», dans *Actes de TALN 2015*, Caen, p. 579–585.

Lambrey, F. et F. Lareau. 2016, «GÉCO v1.0 : User Manual», Université de Montréal, Département de linguistique et de traduction, p. 26.

Langkilde, I. 2000, «Forest-based Statistical Sentence Generation», dans *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, NAACL 2000, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 170–177.

Langkilde-geary, I. et K. Knight. 2000, «Forest-based statistical sentence generation», dans *In Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, Morgan Kaufmann Publishers Inc, p. 170–177.

Lareau, F., M. Dras et R. Dale. 2011, «Detecting Interesting Event Sequences for Sports Reporting», dans *Proceedings of ENLG 2011*, Nancy, p. 200–205.

Lareau, F., F. Lambrey, I. Dubinskaite, D. Galarreta-Piquette et M. Nejat. 2018, «GenDR : A Generic Deep Realizer with Complex Lexicalization», dans *Proceedings of 11th Edition of the Language Resources and Evaluation Conference (LREC)*, Miyazaki.

Lareau, F., L. Wanner, P. Fabra, T. Holloway et E. M. Bender. 2007, «Towards a Generic Multilingual Dependency Grammar for Text Generation», .

Lavoie, B. et O. Rambow. 1997, «A Fast and Portable Realizer for Text Generation Systems», dans *Proceedings of the Fifth Conference on Applied Natural Language Processing*, ANLC '97, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 265–268.

Levin, B. 1993, «English verb classes and alternations : A preliminary investigation», The University of Chicago Press, Chicago.

Liu, M.-c., T.-y. Chiang et others. 2008, «The construction of Mandarin VerbNet : A frame-based study of statement verbs», *Language and Linguistics*, vol. 9, n° 2, p. 239–270.

Mahamood, S. et E. B. Reiter. 2011, «Generating Affective Natural Language for Parents of Neonatal Infants», dans *Proceedings of ENLG-2011*, Association for Computational Linguistics, p. 12–21.

Majewska, O., I. Vulić, D. McCarthy, Y. Huang, A. Murakami, V. Laippala et A. Korhonen. 2017, «Investigating the cross-lingual translatability of VerbNet-style classification», *Language Resources and Evaluation*, doi :10.1007/s10579-017-9403-x, ISSN 1574-0218.

Mann, W. C. 1983, «An overview of the PENMAN text generation system», dans *Proceedings of the National Conference on Artificial Intelligence*, AAAI, p. 261–265. Also appears as USC/Information Sciences Institute, RR-83-114.

Matthiessen, C. M. I. M. et M. A. K. Halliday. 1997, «Systemic functional grammar : A first step into the theory», .

Mazzei, A., C. Battaglino et C. Bosco. 2016, «SimpleNLG-IT : Adapting SimpleNLG to Italian», dans *Proceedings of the 9th International Natural Language Generation Conference*, Association for Computational Linguistics, Edinburgh, UK, p. 184–192.

McRoy, S. W., S. Channarukul et S. S. Ali. 2003, «An augmented template-based approach to text realization», *Natural Language Engineering*, vol. 9, n° 4, p. 381–420.

Mel'čuk, I. 1988, «Dependency Syntax : Theory and Practice», State University of New York Press.

Mel'čuk, I. 1997, «Vers une linguistique Sens-Texte», Leçon inaugurale, Collège de France.

Mel'čuk, I. 2012, «Semantics : From Meaning to Text», vol. 1, John Benjamins Publishing Company.

Mel'čuk, I. 1996, «Lexical functions : A tool for the description of lexical relations in a lexicon», dans *Lexical Functions in Lexicography and Natural Language Processing*, John Benjamins Publishing, p. 37–102.

Milićević, J. 2007, «A Short Guide to the Meaning-text Linguistic Theory», *Journal of Koralex*, vol. 8, p. 187–233.

Milićević, J. 2009, «Schéma de régime : Le pont entre le lexique et la grammaire», *Langages*, vol. 176, nº 4, p. 94–116.

Mille, S., B. Bohnet, L. Wanner et A. Belz. 2017a, «Shared Task Proposal : Multilingual Surface Realization Using Universal Dependency Trees», dans *Proceedings of the 10th International Conference on Natural Language Generation*, Association for Computational Linguistics, Santiago de Compostela, Spain, p. 120–123.

Mille, S., R. Carlini, A. Burga et L. Wanner. 2017b, «FORGe at SemEval-2017 Task 9 : Deep sentence generation based on a sequence of graph transducers», dans *Proceedings of the 11th International Workshop on Semantic Evaluation, SemEval@ACL 2017, Vancouver, Canada, August 3-4, 2017*, p. 920–923.

Mille, S. et L. Wanner. 2015, «Towards Large Coverage Detailed Lexical Resources for Data-to-Text Generation.», dans *Proceedings of the First International Workshop on D2T Generation*, Edinburgh, Scotland.

Mille, S. et L. Wanner. 2017, «A demo of FORGe : The Pompeu Fabra Open Rule-based Generator», *Proceedings of the 10th International Conference on Natural Language Generation*, p. 245–246.

Miller, G. A., R. Beckwith, C. Fellbaum, D. Gross et K. Miller. 1990, «WordNet : An on-line lexical database», *International Journal of Lexicography*, vol. 3, p. 235–244.

Molins, P. et G. Lapalme. 2015a, «JSrealB : A Bilingual Text Realizer for Web Programming», dans *ENLG 2015 - Proceedings of the 15th European Workshop on Natural Language Generation, 10-11 September 2015, University of Brighton, Brighton, UK*, p. 109–111.

Molins, P. et G. Lapalme. 2015b, «JSrealB : A Bilingual Text Realizer for Web Programming», Association for Computational Linguistics, p. 109–111.

Nivre, J., M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajic, C. D. Manning, R. McDonald, S. Petrov, S. Pyysalo, N. Silveira, R. Tsarfaty et D. Zeman. 2016, «Universal Dependencies v1 : A Multilingual Treebank Collection», dans *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, édité par N. C. C. Chair), K. Choukri, T. Declerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk et S. Piperidis, European Language Resources Association (ELRA), Portorož, Slovenia, ISBN 978-2-9517408-9-1.

de Oliveira, R. et S. Sripada. 2014, «Adapting SimpleNLG for Brazilian Portuguese realisation», dans *Proceedings of the 8th International Natural Language Generation Conference (INLG)*, Association for Computational Linguistics, Philadelphia, Pennsylvania, U.S.A., p. 93–94.

Pala, K. et A. Horák. 2008, «Can complex valency frames be universal», *RASLAN 2008 Recent Advances in Slavonic Natural Language Processing*, p. 41.

Palmer, M., D. Gildea et P. Kingsbury. 2005, «The Proposition Bank : An Annotated Corpus of Semantic Roles», *Comput. Linguist.*, vol. 31, n° 1, p. 71–106.

Paroubek, P., Y. Schabes et A. K. Joshi. 1992, «XTAG : A Graphical Workbench for Developing Tree-adjoining Grammars», dans *Proceedings of the Third Conference on Applied Natural Language Processing*, ANLC '92, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 223–230.

Pfeil, J. W. 2016, «Algorithms and Resources for Scalable Natural Language Generation», thèse de doctorat, Case Western Reserve University.

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. 1998a, «La théorie Sens-Texte», *Dialangue*, vol. 8-9, p. 9–30.

Polguère, A. 1998b, «Pour un modèle stratifié de la lexicalisation en génération de texte», *Traitement Automatique des Langues (TAL)*, vol. 39, n° 2, p. 57–76.

Ramos Soto, A., J. Janeiro Gallardo et A. Bugarín Diz. 2017, «Adapting SimpleNLG to Spanish», dans *Proceedings of the 10th International Conference on Natural Language Generation*, Association for Computational Linguistics, Santiago de Compostela, Spain, p. 144–148.

Reiter, E. et A. Belz. 2009, «An Investigation into the Validity of Some Metrics for Automatically Evaluating Natural Language Generation Systems», *Computational Linguistics*, vol. 35, n° 4, p. 529–558.

Reiter, E. et R. Dale. 2000, «Building Natural Language Generation Systems», Cambridge University Press, New York, NY, USA.

Scarton, C. et S. Alu. 2012, «Towards a cross-linguistic VerbNet-style lexicon for Brazilian Portuguese», *Workshop on Creating Cross-language Resources for Disconnected Languages and Styles Workshop Programme*.

Schuler, K. K. 2005, «Verbnet : A Broad-coverage, Comprehensive Verb Lexicon», PhD Thesis, University of Pennsylvania, Philadelphia, PA, USA.

Shi, L. et R. Mihalcea. 2005, «Putting Pieces Together : Combining FrameNet, VerbNet and WordNet for Robust Semantic Parsing», dans *Proceedings of the 6th International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing'05, Springer-Verlag, Mexico City, Mexico, p. 100–111.

Taulé, M. 2010, «AnCora-Net : Mapping the Spanish AnCora-Verb lexicon to VerbNet», dans *Proceedings of Interdisciplinary Workshop on Verbs.*, Pisa, Italia, p. 76–81.

Tesnière, L. 1965, «Eléments de syntaxe structurale», Klincksieck, Paris. OCLC : 489764849.

Thomason, J., S. Venugopalan, S. Guadarrama, K. Saenko et R. Mooney. 2014, «Integrating Language and Vision to Generate Natural Language Descriptions of Videos in the Wild», dans *Proceedings of the 25th International Conference on Computational Linguistics (COLING 2014)*, Dublin, Ireland, p. 1218–1227.

Traum, D. et N. Habash. 2000, «Generation from Lexical Conceptual Structures», dans *Proceedings of the 2000 NAACL-ANLP Workshop on Applied Interlinguas : Practical Applications of Interlingual Approaches to NLP - Volume 2*, NAACL-ANLP-Interlinguas '00, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 52–59.

Van Der Lee, C., E. Krahmer et S. Wubben. 2017, «PASS : A Dutch data-to-text system for soccer, targeted towards specific audiences», dans *Proceedings of the 10th International Conference on Natural Language Generation*, Association for Computational Linguistics, Santiago de Compostela, Spain, p. 95–104.

Vaudry, P.-L. et G. Lapalme. 2013, «Adapting SimpleNLG for Bilingual English-French Realisation», dans *Proceedings of the 14th European Workshop on Natural Language Generation*, Association for Computational Linguistics, Sofia, Bulgaria, p. 183–187.

Vicente, M., C. Barros, F. Peregrino Torregrosa, F. Agulló Antolín et E. Lloret. 2015, «La generación de lenguaje natural : Análisis del estado actual», , p. 721–756.

Wanner, L., B. Bohnet, N. Bouayad-Agha, F. Lareau et D. Nicklass. 2010, «MARQUIS : generation of user-tailored multilingual air quality bulletins», *Appl. Artif. Intell.*, vol. 24, n° 10, p. 914–952.

Wen, D., S. Jiang et Y. He. 2008, «A question answering system based on VerbNet frames», dans *Proceedings of the 4th International Conference on Natural Language Processing and Knowledge Engineering, NLPKE 2008, Beijing, China, October 19-22, 2008*, p. 1–8.

White, M. et R. Rajkumar. 2012, «Minimal Dependency Length in Realization Ranking», dans *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 244–255.

XTAG Research Group. 2001, «A Lexicalized Tree Adjoining Grammar for English», cahier de recherche IRCS-01-03, University of Pennsylvania.

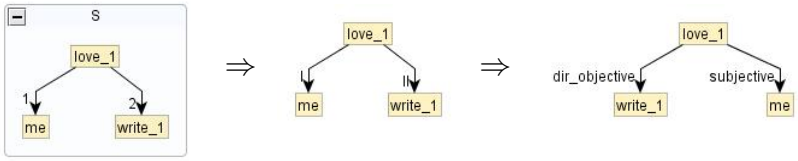
Annexe A

CORPUS D'ÉVALUATION

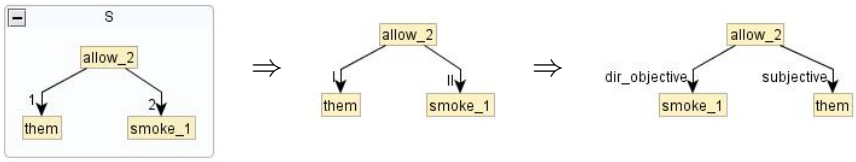
Voici l'échantillon utilisé pour l'évaluation de notre implémentation (voir le chapitre 6). Les structures d'input ont été créées à la main et contre-validées par trois annotateurs. Sauf indication contraire, les structures d'output sont à la fois les structures attendues et celles générées par le système. Dans les cas où nous n'avons pas pu réaliser la structure attendue, nous indiquons par un cadre vert la structure attendue et les structures fautives générées sont indiquées par un cadre rouge quand l'erreur est pertinente pour notre étude, et en orange quand il s'agit d'une erreur qui ne relève pas de notre travail (par exemple, le patron de régime des noms, la génération des pronoms, etc.). Finalement, nous précisons que la relation `main` qui marque le nœud saillant d'un réseau sémantique est occultée dans les figures quand la structure sémantique est déjà hiérarchique (il s'agit alors de la racine).

Parmi cet échantillon, certaines structures contiennent un nœud étiqueté d'un X suivi d'un chiffre. Ces nœuds représentent un sujet implicite en syntaxe profonde. Ces "nœuds fantômes" facilitent la création de la structure syntaxique profonde mais disparaissent en syntaxe de surface. Finalement, notez que les auxiliaires ne sont pas réalisés en syntaxe de surface. Comme ce n'était pas l'objet de notre projet, nous n'avons pas développé une règle de grammaire réalisant les auxiliaires en surface.

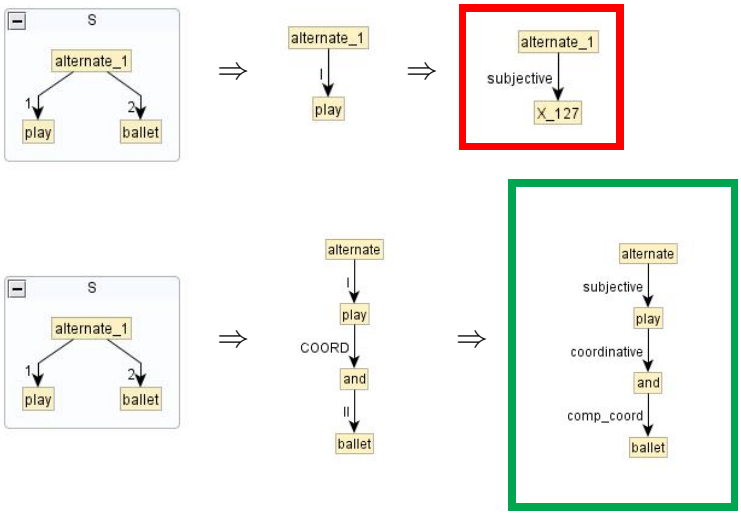
(17) *I loved writing.*



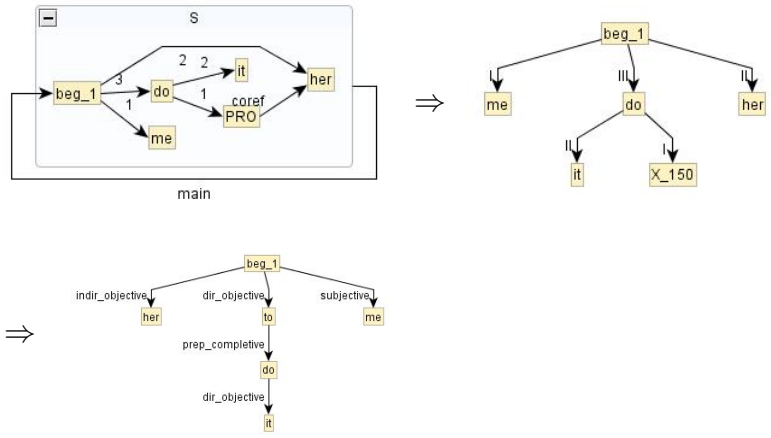
(27) *They allow smoking.*



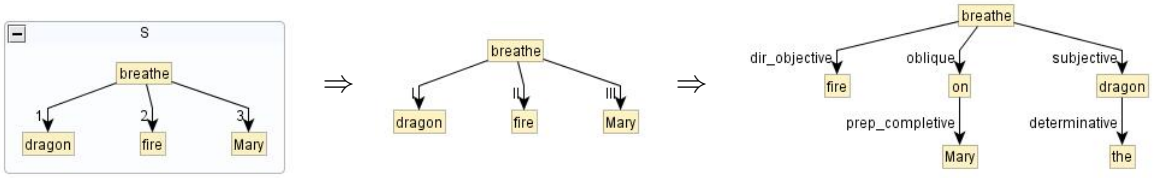
(36) *Plays and ballets alternate.*



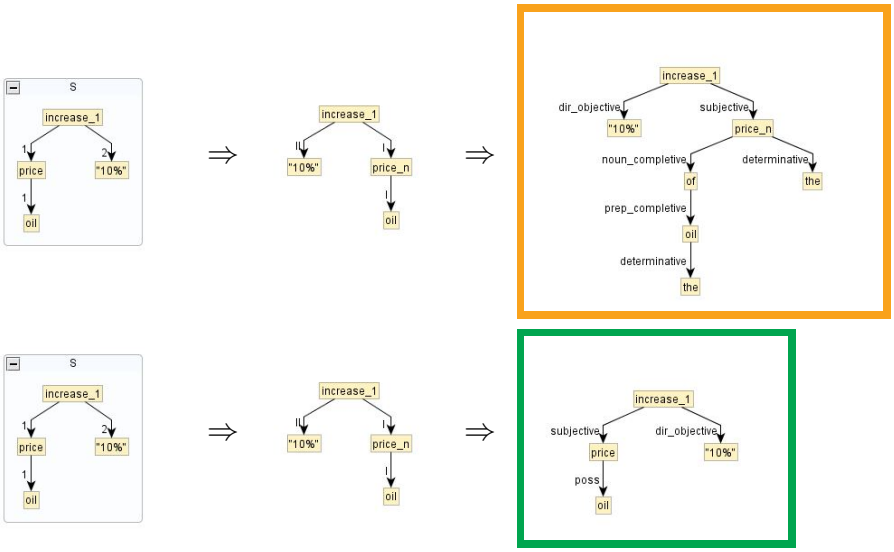
(77) *I begged her to do it.*



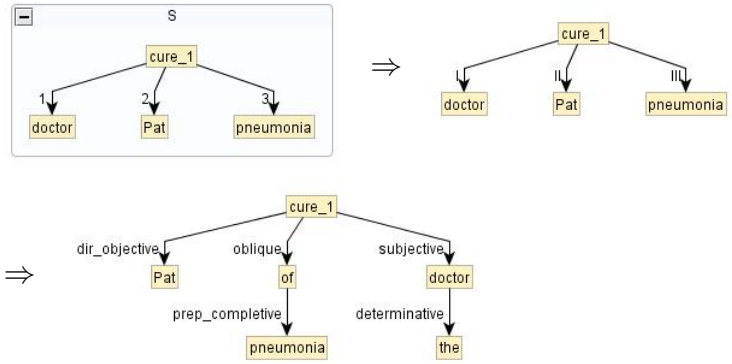
(111) *The dragon breathed fire on Mary.*



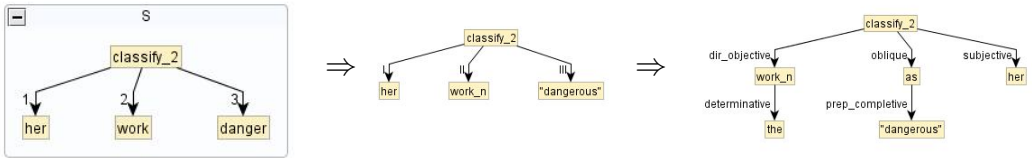
(142) *Oils's price increased ten percent.*



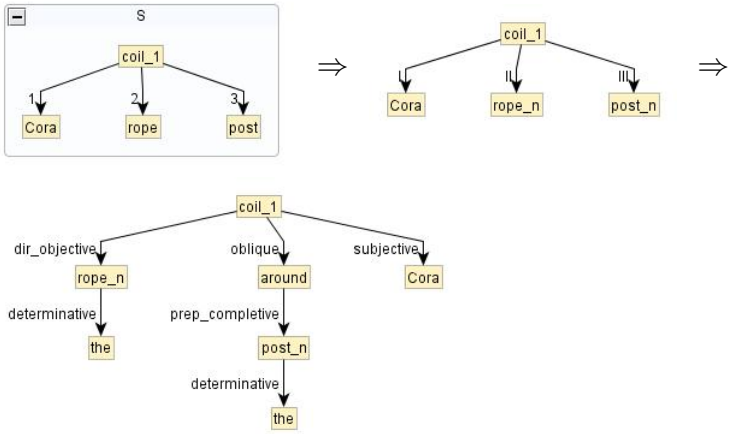
(177) *The doctor cured Pat of pneumonia.*



(198) *She classified the works as 'dangerous'.*

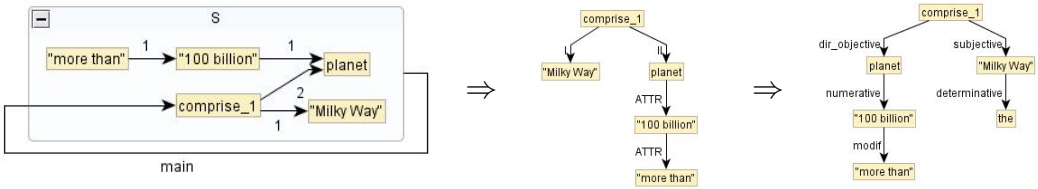


(206) *Cora coiled the rope around the post.*

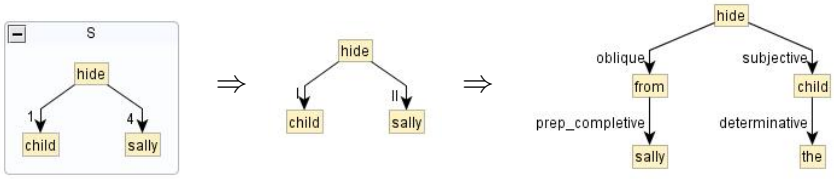


(213) (à la fin de l'annexe)

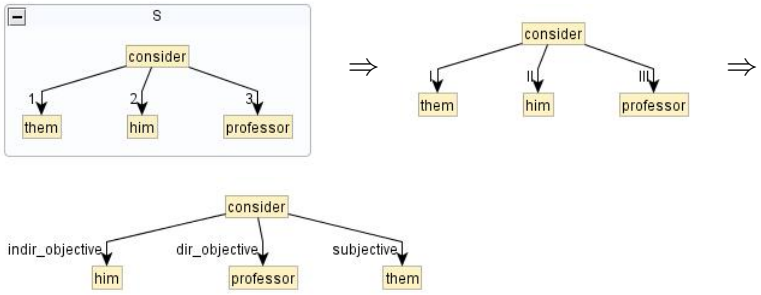
(222) *The Milky Way comprises more than 100 billion planets.*



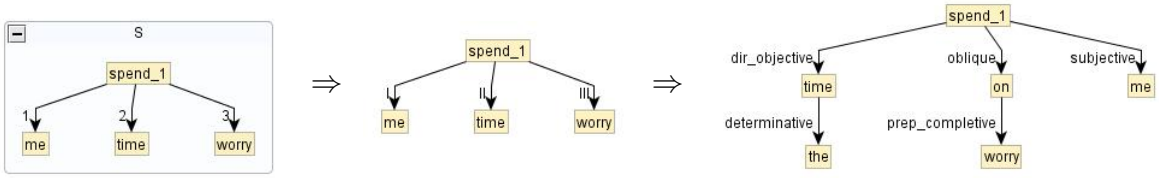
(227) *The children hid from Sally.*



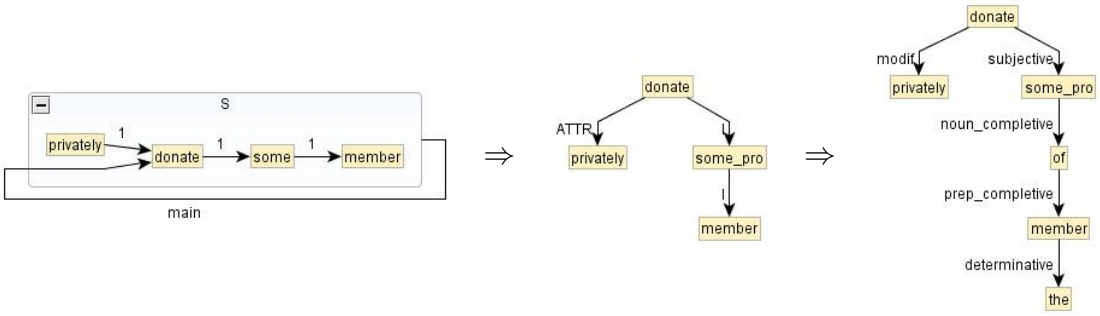
(244) *They considered him professor.*



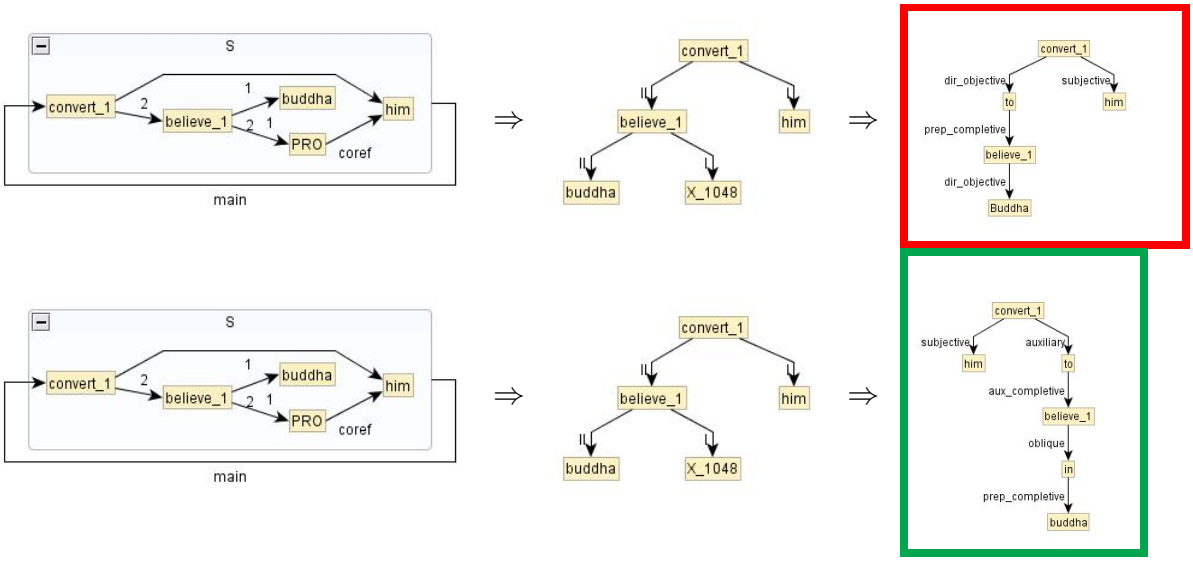
(250) *I spent the time on worries.*



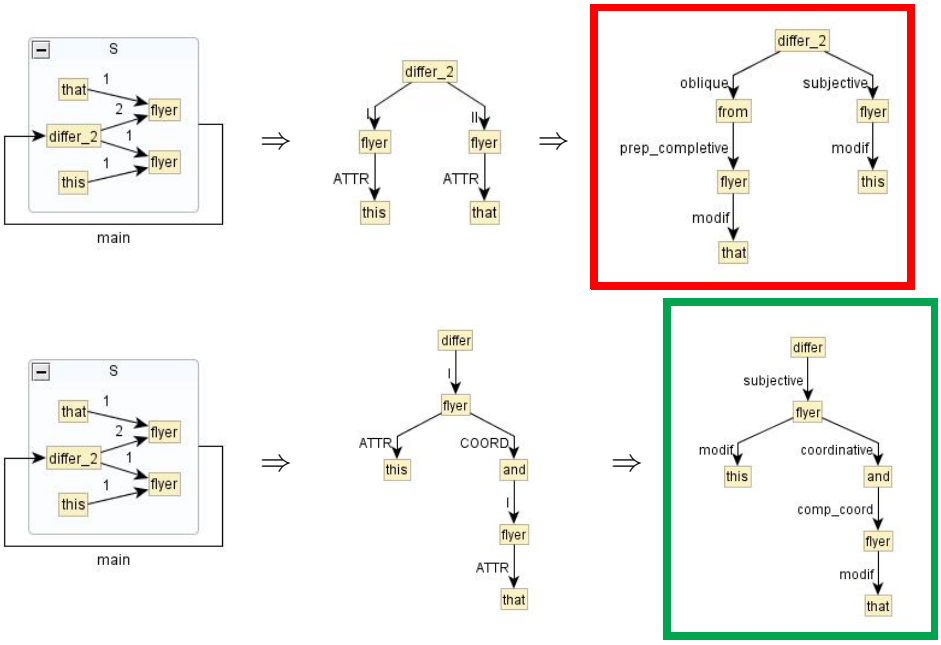
(264) *Some of the members may donate privately.*



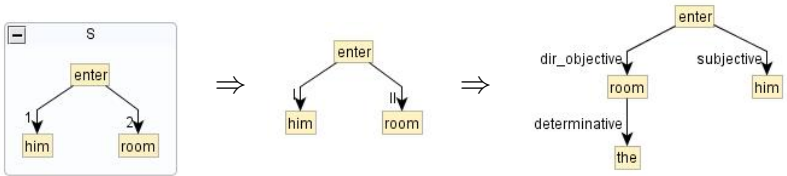
(267) *He converted to believing in Buddha.*



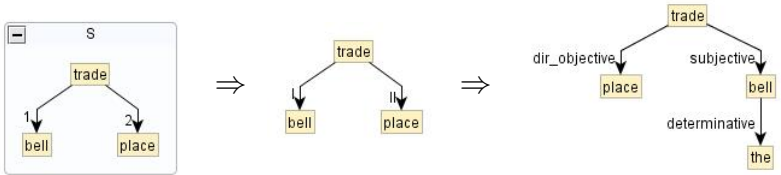
(330) *This flyer and that flyer differ.*



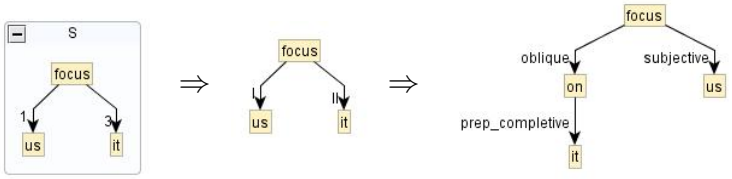
(384) *He entered the room.*



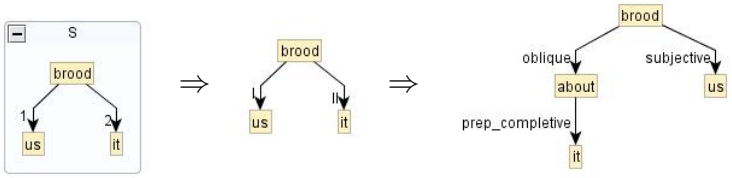
(399) *The bells traded places.*



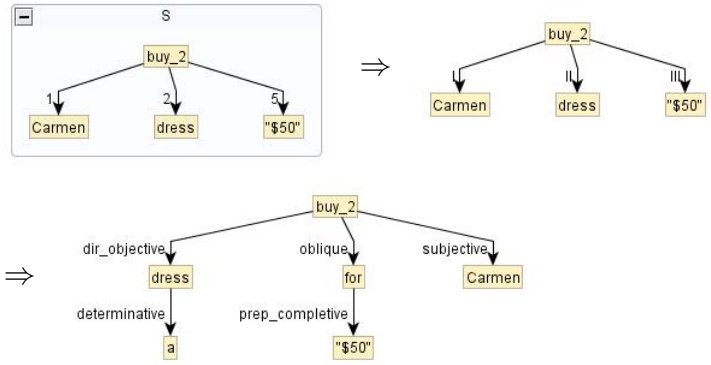
(426) *We focused on it.*



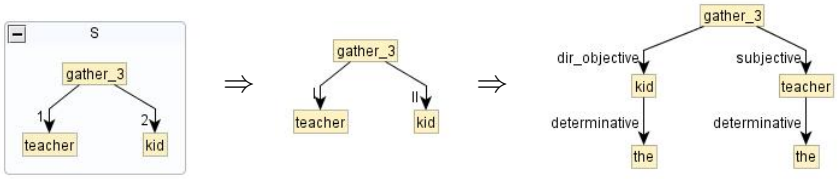
(428) *We brooded about it.*



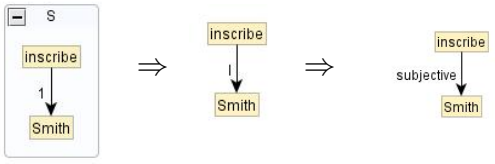
(452) *Carmen bought a dress for 50$.*



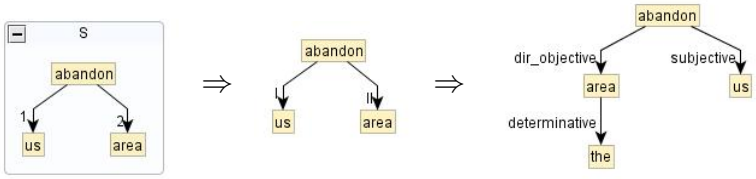
(476) *The teacher gathered the kids.*



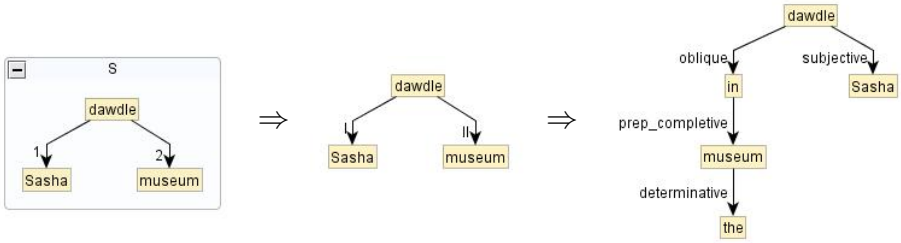
(503) *Smith was inscribing.*



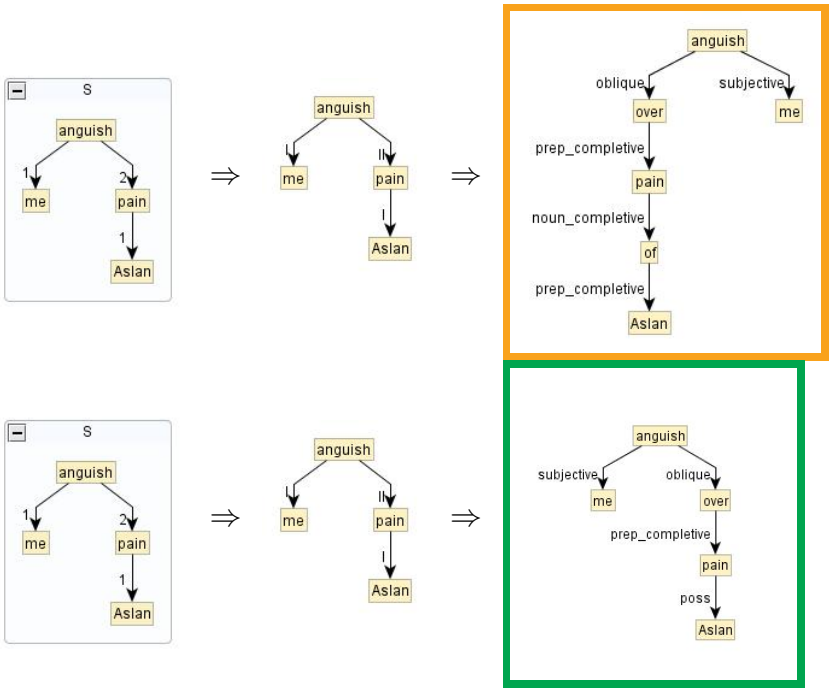
(540) *We abandoned the area.*



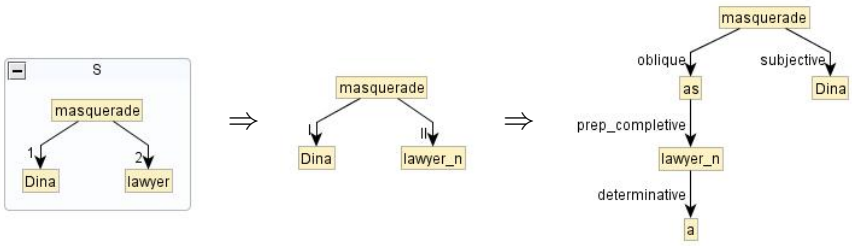
(554) *Sasha dawdled in the museum.*



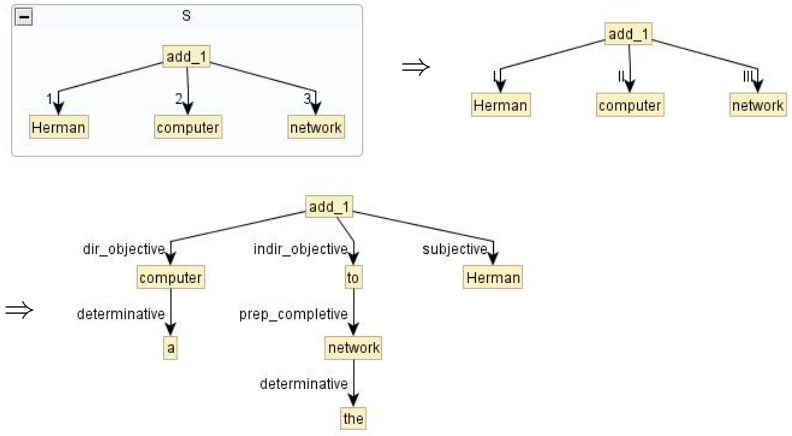
(581) *I anguished over Aslan's pain.*



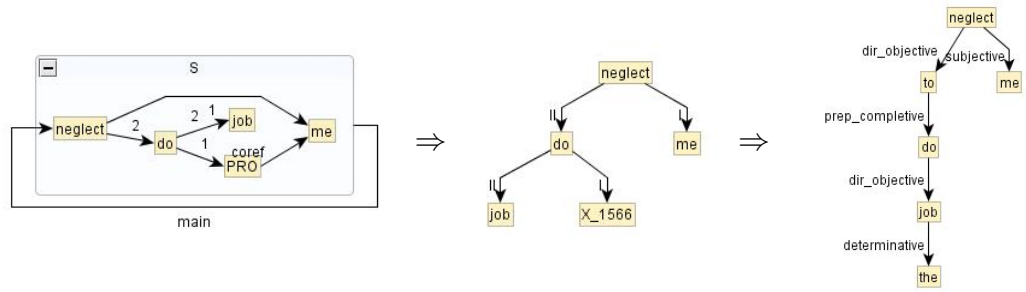
(583) *Dina masqueraded as a lawyer.*



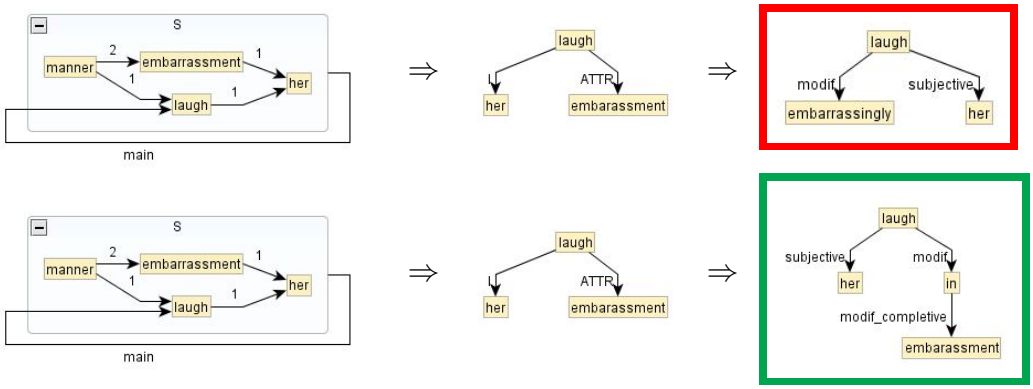
(604) *Herman added a computer to the network.*



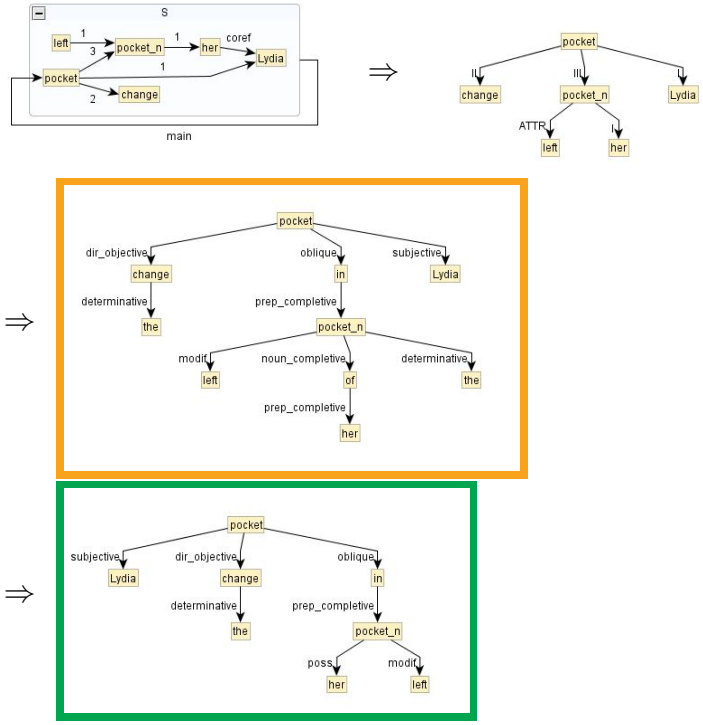
(618) *I neglected to do the job.*



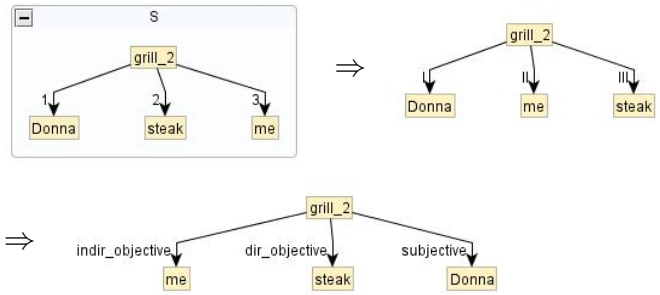
(630) *She laughed in embarrassment.*



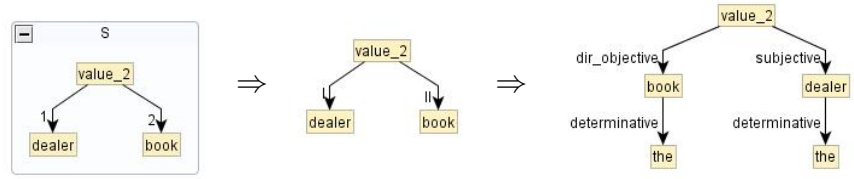
(667) *Lydia pocketed the change in her left pocket.*



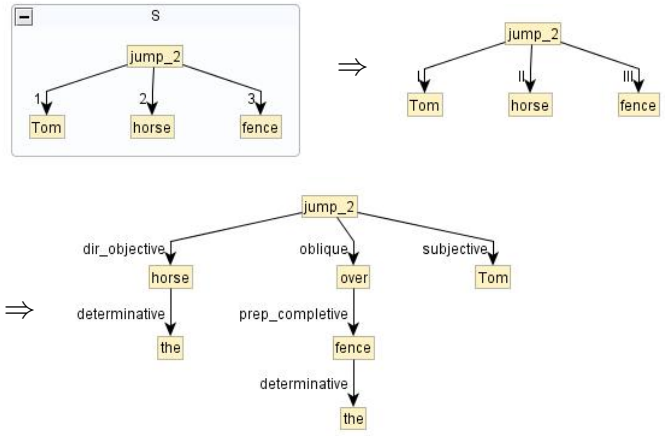
(686) *Donna grilled me steaks.*



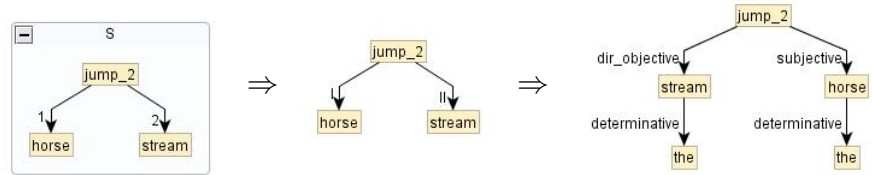
(688) *The dealer valued the book.*



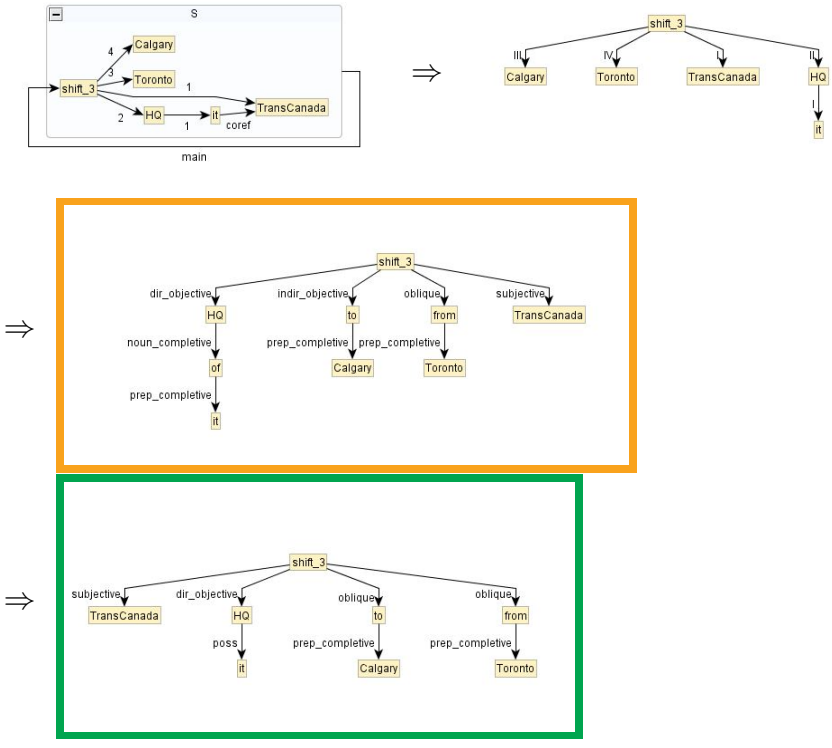
(746) *Tom jumped the horse over the fence.*



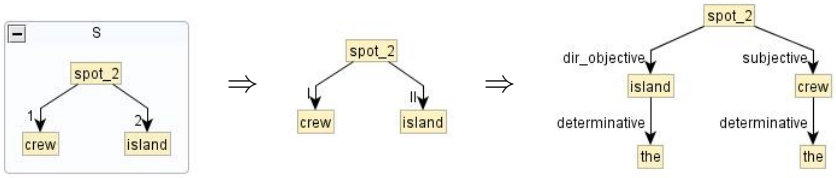
(750) *The horse jumped the stream.*



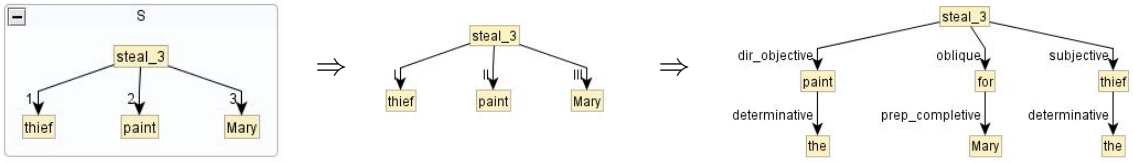
(777) *TransCanada is shifting its HQ to Calgary from Toronto.*



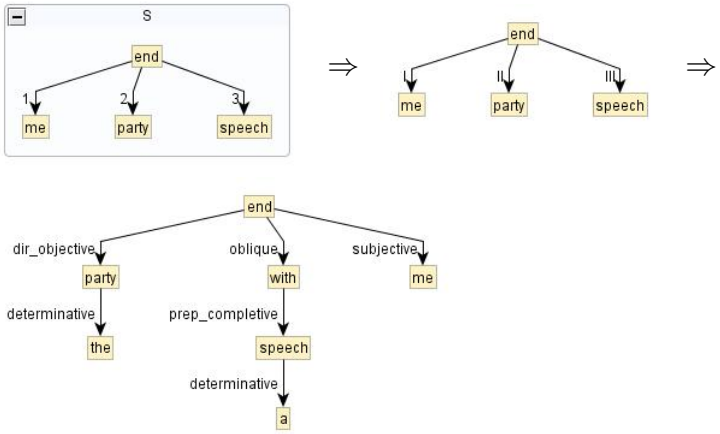
(789) *The crew spotted the island.*



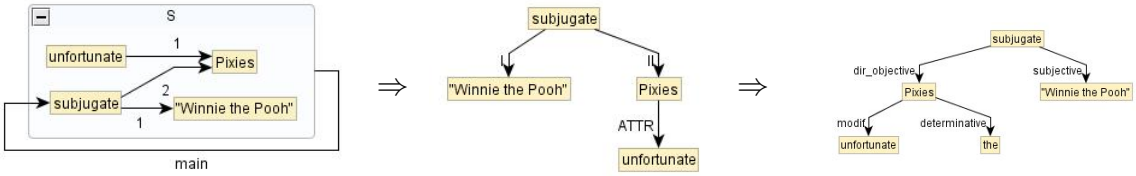
(836) *The thief stole the paint for Mary.*



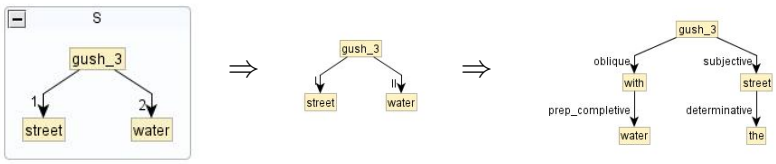
(843) *I ended the party with a speech.*



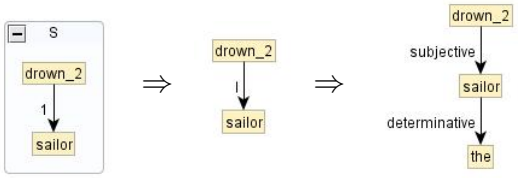
(845) *Winnie the Pooh subjugated the unfortunate Pixies.*



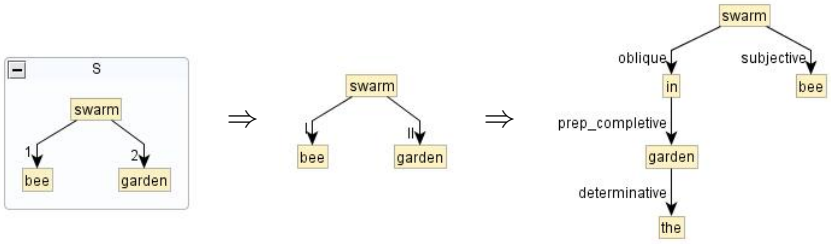
(851) *The streets gushed with water.*



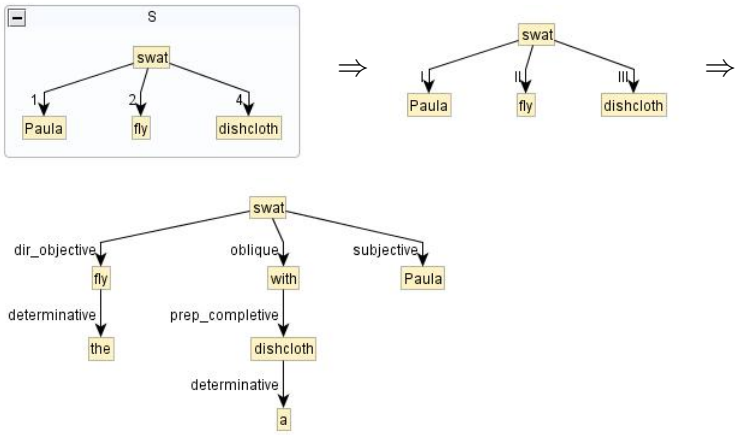
(862) *The sailor drowned.*



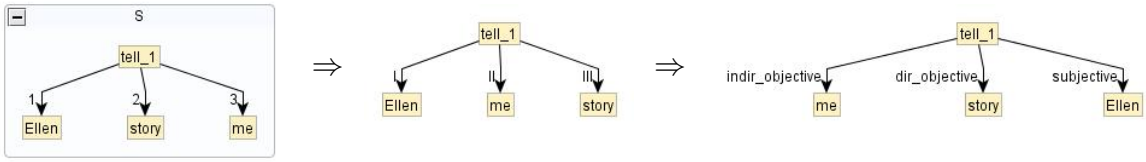
(868) *Bees are swarming in the garden.*



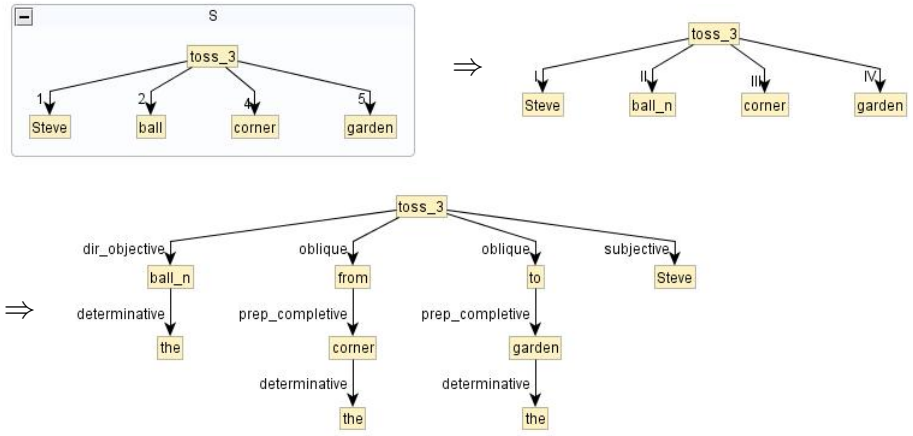
(873) *Paula swatted the fly with a cloth.*



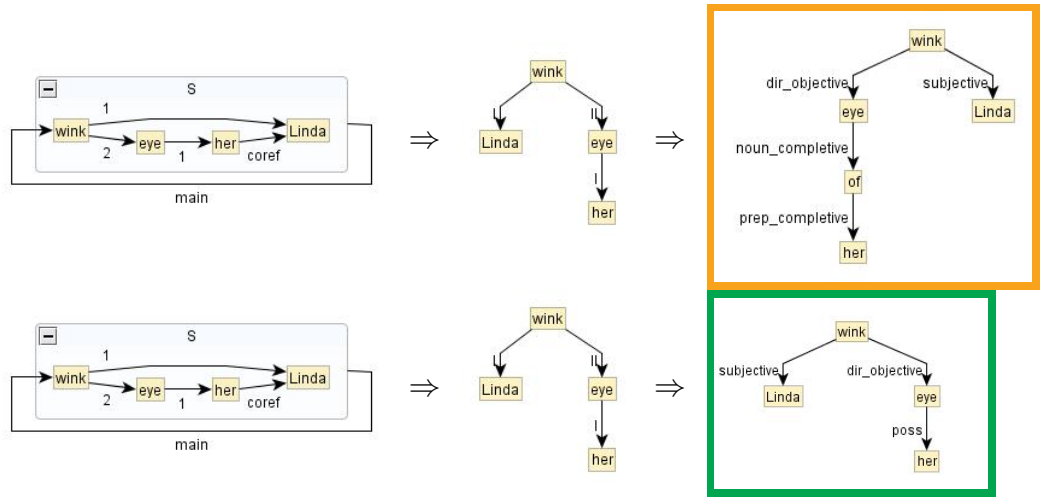
(894) *Ellen told me a story.*



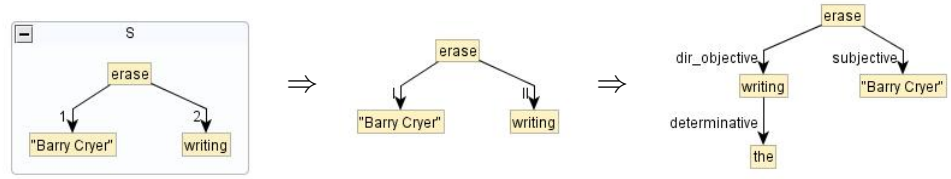
(900) *Steve tossed the ball from the corner to the garden.*



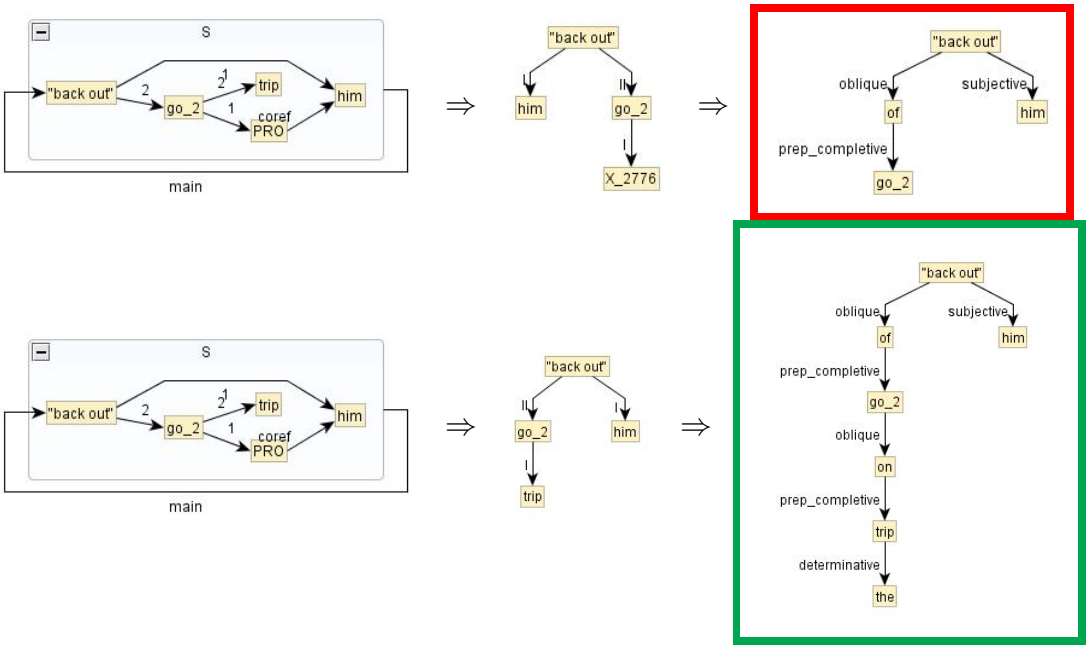
(955) *Linda winked her eye.*



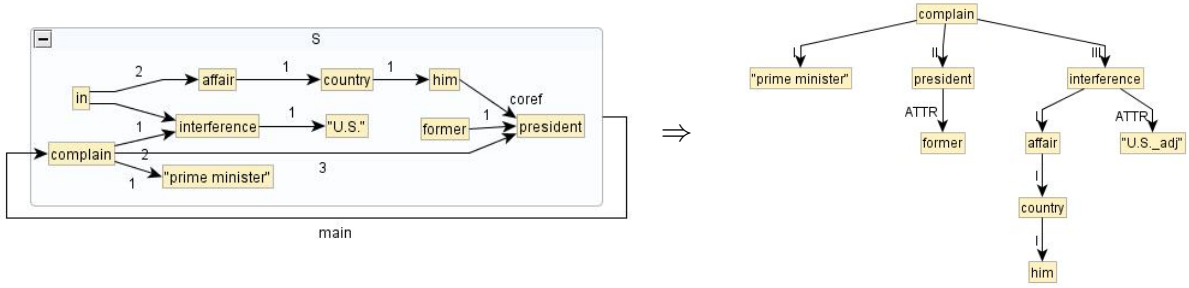
(968) *Barry Cryer erased the writing.*



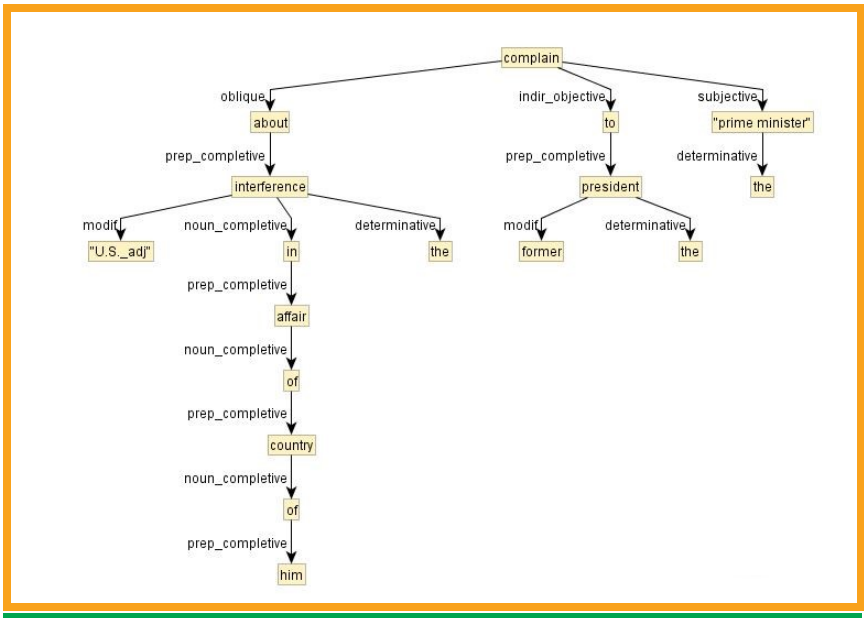
(974) *He backed out of going on the trip.*



(213) *The prime minister complained to the former president about U.S. interference in his country's affairs.*



⇒



⇒