Université de Montréal

Le traitement des verbes à montée et à contrôle dans un réalisateur profond

Par

Yunpeng Qiao

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

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

Août 2024

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

---

Ce mémoire intitulé

Le traitement des verbes à montée et à contrôle dans un réalisateur profond

Présenté par

Yunpeng Qiao

A été évalué par un jury composé des personnes suivantes

Antoine Venant
Président-rapporteur

François Lareau
Directeur de recherche

Justin Royer
Membre du jury

Résumé

Le verbe joue souvent un rôle central dans une phrase. Pour générer des phrases aussi naturelles que possible, un générateur symbolique requiert des dictionnaires riches, particulièrement pour les verbes. Cependant, certains verbes, tels que les verbes à montée et à contrôle utilisés dans le langage courant, ne reçoivent malheureusement pas autant d'attention et d'études qu'ils méritent dans le domaine de la génération automatique de texte (GAT). Ce mémoire a pour objectif de modéliser les verbes à montée et à contrôle dans GenDR, un réalisateur de texte profond. Pour y parvenir, nous nécessitons deux éléments essentiels : des dictionnaires pertinents et des règles spécifiques.

En l'absence d'une ressource lexicale pour les verbes à montée et à contrôle, nous avons utilisé l'outil Graph Rewriting (Grew) pour automatiquement extraire les phrases comportant ces verbes à partir des corpus, entre autres, `French-GSD` et `English-GUM`. Ensuite, nous avons traité les données extraites pour identifier le patron de régime des verbes pertinents et en construire les dictionnaires adaptés à GenDR.

En outre, en nous appuyant sur la syntaxe de dépendance, nous avons élaboré des règles spécifiques afin de permettre à GenDR de générer des phrases avec les verbes en question.

Grâce à cette approche, nous avons ainsi réussi à modéliser les verbes à montée et à contrôle dans le cadre de GenDR.

Mots-clés: génération automatique de texte, réalisateur profond, verbes, Théorie Sens-Texte

Abstract

Verbs play a central role in a sentence. To generate sentences as natural as possible, a symbolic generator requires rich dictionaries, particularly for verbs. However, certain verbs, such as raising and control verbs frequently used in everyday language, unfortunately do not receive as much attention and study as they deserve in the field of automatic text generation. This thesis aims to model raising and control verbs in GenDR, a deep realizer. To accomplish this aim, we require two essential elements: pertinent dictionaries and specific rules.

Since no lexical resource exists for raising and control verbs, we used Graph Rewriting (Grew) to automatically extract sentences containing these verbs from corpora, including `French-GSD` and `English-GUM`. Then, we processed the extracted data to identify the government pattern of relevant verbs and build GenDR-compatible dictionaries.

Building on dependency syntax principles, we developed specific grammatical rules that enable GenDR to properly handle these verb constructions.

Through this approach, we have successfully modeled raising and control verbs within GenDR.

Keywords: automatic text generation, deep realizer, verbs, Meaning-Text Theory

Table des matières

Liste des tableaux

Liste des figures

Liste des listings

Liste des sigles et abréviations

AMR	Abstract Meaning Representation
DMRS	Dependency Minimal Recursion Semantics
GAT	génération automatique de texte
GP	Patron de régime (Government Pattern)
Grew	Graph Rewriting
JSON	JavaScript Object Notation
RSém	représentation sémantique
RSyntP	représentation syntaxique profonde
RSyntS	représentation syntaxique de surface
SSyntP	structure syntaxique profonde
SUD	Surface-Syntactic Universal Dependencies
TAL	traitement du langage naturel
TST	théorie Sens-Texte
UD	Universal Dependencies

Remerciements

Je tiens à remercier infiniment mon directeur de recherche, François Lareau !

Chapitre 1

Introduction

1.1 Problématique

Le verbe, représentant le cœur d'une phrase, véhicule généralement son message principal. Syntaxiquement, il gouverne ses actants au sommet de la structure arborescente. Ces caractéristiques lui confèrent une place importante dans la génération automatique de texte (GAT). Galarreta-Piquette (2018) a intégré les régimes verbaux du dictionnaire VerbNet (Levin, 1993; Schuler, 2005; Kipper *et al.*, 2007) dans GenDR. GenDR, développé par Lareau *et al.* (2018), est un réalisateur profond multilingue, permettant de générer des représentations syntaxiques dans diverses langues. Grâce aux efforts de divers chercheurs, GenDR peut actuellement supporter plusieurs langues : l'anglais (Galarreta-Piquette, 2018), le chinois mandarin (He, 2020), le français (Lareau *et al.*, 2018) et le lituanien (Dubinskaite, 2017).

Cependant, ces implémentations précédentes ne permettent pas de générer des phrases comportant des verbes à montée et à contrôle. Par exemple, les régimes verbaux de VerbNet intégrés ne prennent pas en compte ces verbes. Bien que les verbes à montée et à contrôle posent des défis de modélisation en raison des phénomènes syntaxiques complexes qu'ils impliquent, tels que le partage ou la montée d'actants, ils sont fréquemment utilisés dans le langage courant. Par conséquent, leur implémentation dans GenDR s'avère primordiale, car cela permettra d'améliorer sa capacité de génération. Donc, notre objectif est de modéliser les verbes à montée et à contrôle dans le cadre de GenDR.

La réalisation de l'objectif nécessite deux éléments essentiels : des dictionnaires pertinents et des règles spécifiques, tous deux adaptés à l'utilisation de GenDR. Leur élaboration consiste en un travail complexe qui se divise en quatre étapes :

1. Extraire automatiquement les phrases comportant les verbes à montée et à contrôle à

partir de corpus ;

2. Identifier automatiquement les patrons de régime des verbes pertinents, en basant sur les données extraites ;
3. Construire un dictionnaire lexical et un dictionnaire des partons de régime à partir de patrons de régime identifiés ;
4. Ajouter un module intégré des règles spécifiques à la version actuelle de GenDR basée sur le *gpcon* (Galarreta-Piquette, 2018) pour traiter les phénomènes de contrôle et de montée.

La nouvelle version de GenDR équipée du nouveau module sera capable de générer des phrases contenant des verbe à montée et à contrôle dans diverses langues, à condition que les dictionnaires pertinents l'accompagnent. En raison du volume de travail, le présent mémoire se concentre seulement sur le français et l'anglais.

1.2 Organisation du mémoire

1. Nous faisons une revue de la littérature dans les sections suivantes du présent chapitre.
2. Le chapitre 2 présente la méthode d'extraction de données et de construction des dictionnaires adaptés au réalisateur profond.
3. Le chapitre 3 consiste à concevoir et implémenter le nouveau module chargé du traitement des verbes à montée et à contrôle dans GenDR.
4. Finalement, nous revenons à l'objectif de la recherche et proposons des perspectives de futurs travaux dans le chapitre 4.

1.3 GenDR

GenDR (Lareau *et al.*, 2018; Lambrey, 2016) est un réalisateur de texte profond multilingue basé sur la théorie Sens-Texte (Žolkovskij et Mel'čuk, 1967; Mel'čuk, 1997; Polguère, 1998). Il a hérité de MARQUIS (Lareau et Wanner, 2007), un système météorologique multilingue de données vers texte. Il fonctionne sur un transducteur de graphes appelé MATE (Bohnet *et al.*, 2000; Bohnet et Wanner, 2010).

En plus du transducteur, GenDR requiert trois composantes essentielles pour travailler : les règles, les dictionnaires et les graphes d'entrée. GenDR prend en entrée la représentation sémantique (RSém) et retourne la représentation syntaxique de surface (RSyntS), en passant

par la représentation syntaxique profonde (RSyntP). Les modules contenant les règles orchestrent ces transformations successives, comme illustré dans l'organigramme de programmation référencé à la figure 1.1 [1].

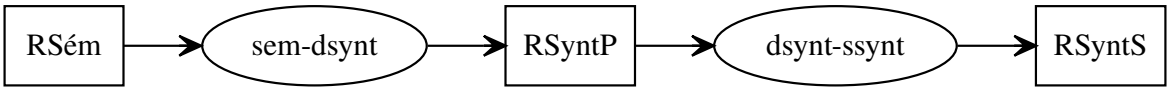


FIGURE 1.1 – Processus de GenDR : de la RSém à la RSyntS

1.3.1 Graphes

Représentation sémantique (RSém)

La RSém sert de point de départ pour GenDR. Elle se présente comme un graphe, composé de nœuds et d'arcs. Les nœuds représentent un sémantème et les arcs représentent les relations entre un sémantème prédicatif et ses arguments. La relation entre un prédicat et son argument est orientée, un arc pointant toujours du prédicat vers son argument. Les arcs sont étiquetés avec numéros actanciels qui sont sémantiquement vides. Ils ne servent qu'à distinguer les actants d'un sémantème prédicat. Parmi les sémantèmes dans la RSém, le sémantème le plus saillant est étiqueté *main*, comme nœud dominant de toute la phrase (Melčuk et Wanner, 2001; Polguère, 1990).

Dans son utilisation, MATE accepte une RSém sous forme de texte facile à éditer. MATE peut aussi convertir cette représentation en un graphe pour une meilleure visualisation. Les figures 1.2 et 1.3 présentent ces deux modes. De plus, la RSém est l'entrée des règles du module `sem-dsynt`, comme l'organigramme de programmation en 1.1 le montre.

Représentation syntaxique profonde (RSyntP)

La RSyntP se présente comme un arbre de dépendances dans lequel les nœuds sont, après la lexicalisation, étiquetés avec des lexies profondes et munies de grammèmes profonds. Les nœuds sont liés par les arcs étiquetés avec des relations syntaxiques profondes (Mel'čuk, 1988). La figure 1.4 illustre la RSyntP transformée de la RSém en figure 1.3.

La lexie profonde est une lexie sémantiquement pleine qui correspond à un sémantème, le nom d'une fonction lexicale.

1. Module `sem-dsynt` : semantics→ deep syntax ; module `dsynt-ssynt` : deep syntax→ surface syntax. Les modules seront présentés en §1.3.3.

```
structure Sem S {
  S:1{
    Paul:1{class=proper_noun}
    donner:1{
      1-> Paul:1
      2-> livre:1
      3-> Lucie:1
    }
    livre:1{}
    Lucie:1{class=proper_noun}
    main-> donner:1
  }
}
```

FIGURE 1.2 – RSém en mode textuel

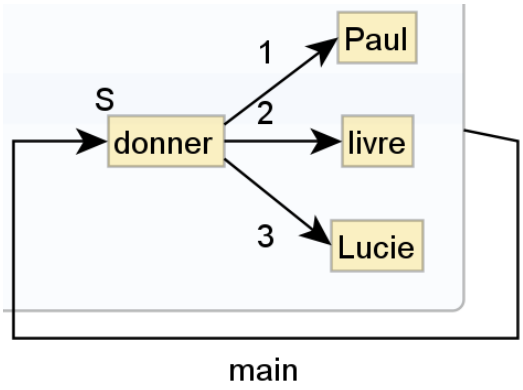


FIGURE 1.3 – RSém en mode visuel

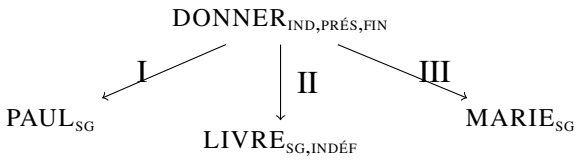


FIGURE 1.4 – Lexies profondes accompagnées de leurs grammèmes dans la RSyntP

Les grammèmes sont des significations flexionnelles, telle que PLURIEL, SUBJONCTIF, FUTUR, etc. Ils accompagnent les lexies profondes dans la RSyntP. Des grammèmes spécifiques sont obligatoire pour les lexies d'une partie du discours donnée. Par exemple, les lexies nominales portent des grammèmes indiquant le nombre et la définitude (défini/indéfini) ; les lexies verbales incluent des grammèmes pour le mode, le temps et la voix.

Les relations syntaxiques profondes diffèrent des relations sémantiques. Une relation sémantique représente la dépendance entre le prédicat et un de ses arguments. Une relation syntaxique profonde s'établit entre le gouverneur syntaxique et son dépendant syntaxique au sein d'un syntagme. Il y a deux types majeurs de relations syntaxiques profondes : les relations actancielles et les relations de modification (Mel'čuk, 2004). Les relations actancielles sont notées par des chiffres romains, de **I** à **VI**. Celles représentant la modification sont notées par **ATTR**. La modification est l'inversion de la relation sémantique, car une lexie L_1 peut être le prédicat sémantique d'une autre lexie L_2, tout en étant son dépendant syntaxique, comme le montre la figure 1.5 (Mel'čuk et Milićević, 2013).

Un exemple concret explique mieux cette particularité. Dans la RSém de la figure 1.6, le prédicat 'plein' domine son argument, le sémantème 'verre'. En revanche, l'adjectif PLEIN

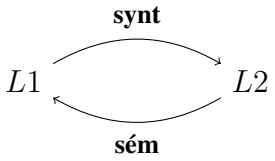


FIGURE 1.5 – Relations syntaxique et sémantique entre un modificateur et son modifié (tiré de Mel'čuk et Milićević (2013, p. 187))

FIGURE 1.6 – Transduction de la RSém à la RSyntP pour le syntagme *verre plein*

comme modificateur dépend du nom VERRE dans la RSyntP.

La construction de la RSyntP est une étape lourde de conséquences : c'est là que s'effectuent les choix lexicaux et syntaxiques, c'est-à-dire la lexicalisation et l'arborisation, permettant à la fois d'engendrer les paraphrases. Donc, la transition entre la RSém et la RSyntP permet de multiplier le paraphrasage et montre la flexibilité de GenDR.

Représentation syntaxique de surface (RSyntS)

La RSyntS est aussi un arbre de dépendances. Elle hérite les nœuds et leurs grammèmes de la RSyntP. En outre, dans la RSyntS s'expriment les mots-outils, que la RSém et la RSyntP n'incluent pas. Les mots-outils comprennent non seulement les lexèmes sémantiquement vides, tels que les prépositions et les conjonctions, mais aussi les auxiliaires et les articles qui sont sémantiquement pleins. Nous devons y trouver tous les lexèmes de la phrase, ce qui fait en sorte que la RSyntS est la plus proche de la forme finale de la phrase parmi les trois représentations.

Les lexèmes sont liés par des relations syntaxiques de surface. Différentes des relations syntaxiques profondes qui sont universelles, les relations syntaxiques de surface sont spécifiques à chaque langue.

La figure 1.7 présente la RSyntS correspondant à la RSyntP illustrée par la figure 1.4.

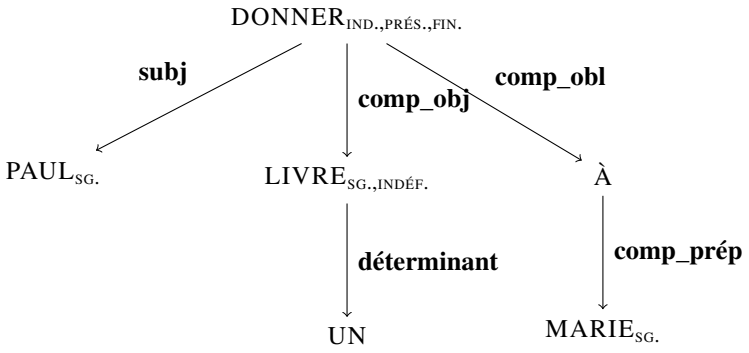


FIGURE 1.7 – RSyntS de la RSyntP en 1.4

1.3.2 Dictionnaires

Les dictionnaires de GenDR constituent les ressources lexicales. Bien que leur contenu varie selon la langue, leur structure et leurs fonctions restent constants. L'approche décrite par Galarreta-Piquette (2018) emploie trois dictionnaires : le dictionnaire sémantique, le dictionnaire lexical et le dictionnaire de patrons de régime.

Dictionnaire sémantique

Le dictionnaire sémantique est aussi connu sous le nom de *semanticon*. Chaque entrée de *semanticon* est un sémantème associé à ses diverses lexies profondes possibles. Le listing 1.1 montre l'extrait de ce dictionnaire. GenDR consulte le *semanticon* lors de la transition de la RSém à la RSyntP. Il sélectionne, pour chaque nœud, l'une parmi les lexies pour le lexicaliser, en respectant les contraintes sur la partie du discours du nœud d'accueil. Ce processus est répété jusqu'à ce que toutes les lexies aient été parcourues, aboutissant à la création de multiples structure syntaxique profonde (SSyntP) dans la RSyntP.

Listing 1.1 – Extrait de *semanticon*

```
admirer { lex=admirer lex=admiration }
arrêter_3 { lex=arrêter_3 lex=arrestation }
aimer_2 { lex=aimer_2 lex=plaire }
avion { lex=avion }
chat { lex=chat }
sembler { lex=sembler lex=paraître }
```

Dictionnaire de patrons de régime

Le dictionnaire de patrons de régime, aussi appelé *gpcon*, regroupe l'ensemble des patrons de régime d'une langue donnée. D'après Milićević (2009), les patrons de régime décrivent les cooccurrences d'une lexie prédicative et ses actants régis. Dans le cadre de GenDR et son dictionnaire *gpcon*, un patron de régime décrit en détail les comportement syntaxiques d'une lexie en fournissant des informations sur le nombre d'actants et leurs attributs, tels que la relation, la finitude, le mode, etc. Le listing 1.2 montre un simple patron de régime utilisé par les verbes transitifs.

Le patron de régime `N_V_N` illustré en 1.2 représente une entrée simple dans ce dictionnaire. Voici une explication de ses composantes :

— `N_V_N` est l'identifiant du GP ;

— `I={rel=subj dpos=N}` correspondant au premier actant syntaxique qui doit être un nom (`dpos=N`). La relation syntaxique de surface entre le gouverneur et l'actant est étiquetée **subj** (sujet).

— `II={rel=comp_obj dpos=N}` correspondant au second actant syntaxique qui doit être un nom (`dpos=N`). La relation syntaxique de surface entre le gouverneur et l'actant est étiquetée **comp_obj** (complément d'objet direct).

Listing 1.2 – Un patron de régime de *gpcon*

```
N_V_N {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=N}
}
```

Dictionnaire lexical

Lexicon, autre nom du dictionnaire lexical, contient l'information sur les lexies, telle que la partie du discours profonde (`dpos`), la partie du discours de surface (`spos`), le patron de régime (GP), la diathèse (`dia`), éventuellement la fonction lexicale (*lf*).

Parmi ces propriétés, la diathèse établit la correspondance entre les actants sémantiques et syntaxiques profonds d'une lexie. Par exemple, une `dia=21` implique une inversion : le premier actant sémantique devient le second actant syntaxique et le second actant sémantique devient le premier actant syntaxique lors de la transition de la RSém à la RSyntP. En bref, la diathèse détermine l'ordre de la réalisation syntaxique des actants sémantiques.

Ce dictionnaire utilise un mécanisme d'héritage qui permet de catégoriser les lexies et

transmettre les attributs entre elles. Par exemple, la lexie MARRY_2 appartient à la classe `marry-36.2` qui hérite des traits de sa classe parente `verb`, tels que `dpos=V` et `spos=verb`. Le listing 1.3 montre bien leurs relations hiérarchiques. Pour une meilleure lisibilité, *lexicon* est composée de deux parties :

1. Dans la première partie, chaque entrée est une lexie associée à une classe syntaxique. Il se peut que plusieurs lexies soient associées à la même classe. Par exemple, la classe `"marry-36.2` régit non seulement la lexie MARRY_2 mais aussi la lexie DIVORCE_1, etc. En revanche, il n'y a aucun cas où une lexie serait associée à plus d'une classe.
2. La deuxième partie comporte les propriétés syntaxiques des classes, comme celles que nous avons mentionnées en haut. Lors de la construction de la RSyntP, la lexie MARRY_2 permet deux patrons de régime : `NP_V` et `NP_V_NP` accompagnés de la diathèse (`dia=12`). Ils forment deux phrases distinctes, comme les exemples fournis dans le listing 1.3.

Listing 1.3 – *marry_2* et ses classes mères dans *lexicon*

```
marry_2 : "marry-36.2"
...
"marry-36.2": verb {
  gp = { id=NP_V    dia=12 } // They married in 2022.
  gp = { id=NP_V_NP dia=12 } // Bill married Kathy in 2022.
}
...
verb {
  dpos = V
  spos = verb
}
```

VerbNet : Le mécanisme d'héritage utilisé dans le dictionnaire lexical est inspiré de VerbNet, un lexique anglais développé par Schuler (2005), basé sur la taxonomie de Levin (1993). VerbNet se distingue par sa structure hiérarchique, organisant environ 5800 verbes. Ces verbes sont regroupés en classes, qui peuvent elles-mêmes contenir des sous-classes, formant une hiérarchie allant jusqu'à trois niveaux de profondeur. Chaque classe comprend des membres (verbes), des rôles thématiques associés aux restrictions de sélection pour les prédicats et leurs arguments et des cadres sémantico-syntaxiques. La hiérarchie permet aux sous-classes d'hériter toute l'information de leurs classes mères, tout en spécifiant des membres de comportement différent. Cette organisation tisse un réseau de classes, sous-classes et

membres, justifiant le nom “VerbNet”.

Dictionnaire des fonctions lexicales

Il existe un quatrième dictionnaire dénommé *lf* (acronyme de *lexical function*), qui n’avait pas été utilisé dans la version initiale de GenDR, *gpcon*. En revanche, la présente recherche exploite ce dictionnaire pour implémenter des fonctions lexicales verbales qui sont cruciales dans le traitement des verbes à montée.

Il convient de noter que les fonctions lexicales trouvent leur origine dans la théorie Sens-Texte. Pour une définition approfondie de ces fonctions, on peut se référer à Mel’čuk (1996); Mel’čuk et Milićević (2013, p. 210) :

> Une fonction lexicale **f** est une fonction qui associe à une lexie L un ensemble d’expressions linguistiques {L_1,...,L_n} qui ont le sens ‘f’ portant sur le sens de L (= sur ‘L’) et qui sont sélectionnées en fonction de L.

Les fonctions lexicales sont un outil important permettant de décrire systématiquement les collocations. Les collocations sont des unités lexicales présentes souvent ensemble (Cruse, 1986), par exemple, *faire l’arrestation*, *effectuer une évaluation*, *peur bleue*, etc. Une collocation est composée d’une base et un collocatif. La fonction lexicale prend en argument la base et renvoie comme valeur son collocatif (Mel’cuk, 2013). Cette valeur n’est pas toujours unique.

L’éventail des fonctions lexicales est vaste et ne peut être exhaustivement présenté ici. Cependant, dans le cadre de ce mémoire, nous nous concentrerons sur trois fonctions lexicales verbales particulièrement pertinentes : $\texttt{Func}_i$, $\texttt{Oper}_i$ et $\texttt{Labor}_{ij}$. Les valeurs des fonctions lexicales verbales sont des verbes sémantiquement vides, également appelés verbes supports. Leur vocation est seulement syntaxique : les verbes supports servent de prédicat syntaxique reliant L et son actant.

1. $\texttt{Func}_i$ prend L comme sujet ; le i-ème actant est un complément d’objet de $\texttt{Func}_i$.
   $\texttt{Func}_1$(tonnerre) = gronder
   $\texttt{Func}_2$(article) = porter (sur)
   $\texttt{Func}_3$(ordre) = concerner
2. $\texttt{Oper}_i$ prend L comme complément d’objet principal ; le i-ème actant est le sujet de $\texttt{Oper}_i$.
   $\texttt{Oper}_1$(ordre) = donner

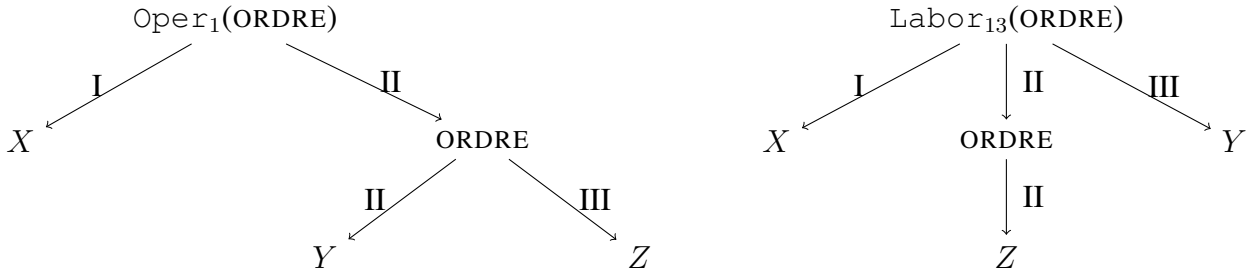
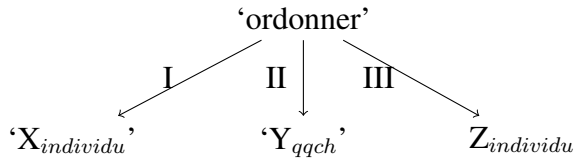


FIGURE 1.8 – Constructions à $\texttt{Oper}_1$ et $\texttt{Labor}_{13}$ avec le nom ORDRE

3. $\texttt{Labor}_{ij}$ prend L comme complément d'objet non principal ; le i-ème actant est le sujet de $\texttt{Labor}_{ij}$ et le j-ème actant est son complément d'objet principal.

   $\texttt{Labor}_{13}$(ordre) = soumettre

Pour déterminer la bonne fonction lexicale dans la description d'une collocation contenant un nom L, il est souvent utile de convertir ce nom en sa forme verbale. Cette approche exploite notre compréhension intuitive de la structure actancielle des verbes. Par exemple, pour utiliser correctement $\texttt{Oper}_1$(ORDRE) et $\texttt{Labor}_{13}$(ORDRE), nous devons connaître la structure actancielle du verbe ORDONNER :



La structure contient 3 actants et signifie qu'un individu X ordonne quelque chose Y à un autre individu Z. $\texttt{Oper}_1$(ORDRE) prend ORDRE comme complément d'objet principal et encore le premier actant X comme sujet, comme le montre la structure gauche dans la figure 1.8 ; on peut alors en déduire la valeur de $\texttt{Oper}_1$, DONNER. Il va de même pour $\texttt{Labor}_{13}$(ORDRE) qui prend ORDRE comme complément d'objet non principal, le premier actant X comme sujet et le troisième actant Z comme complément d'objet principal, comme l'illustre la structure droite dans la même figure. Alors, SOUMETTRE qui correspond bien à cette structure peut être la valeur de $\texttt{Labor}_{13}$(ORDRE).

1.3.3 Règles

Les règles représentent l'âme de GenDR. En tant qu'héritier de MARQUIS (Lareau et Wanner, 2007), GenDR lui emprunte le noyau des règles et développe ses propres règles.

Puisque GenDR est un réalisateur profond multilingue, le noyau de ses règles est universel. Il est chargé de modéliser la lexicalisation, l'arborisation et d'autres phénomènes langagiers fondamentaux. Également, des règles spécifiques à la langue s'occupent d'auxiliaires, déterminants, etc. Elles toutes permettent de générer le texte de toutes les langues, étant données les ressources lexicales appropriées.

Les processus de transduction entre les différentes représentations sont gérés par deux modules distincts qui encapsulent les règles :

1. Le module `sem-dsynt` assure la conversion de la RSém vers la RSyntP.
2. Le module `dsynt-ssynt` prend en charge la transformation de la RSyntP en RSyntS.

Ces modules permettent une transition étape par étape entre les différents niveaux de représentation syntaxique et sémantique.

Dans GenDR, les règles ont conceptuellement une partie gauche, une droite, et des conditions. Cette organisation reflète le principal de fonctionnement « entrée-sortie ». La partie gauche correspond à la structure d'entrée, tandis que la partie droite correspond à la structure de sortie.

Il convient de noter que la présente recherche s'appuie sur l'approche décrite par Galarreta-Piquette (2018) pour effectuer le processus de transduction que nous résumons ci-dessous.

Arborisation

Pour la première transition, GenDR prend en entrée une RSém et cherche à construire un arbre syntaxique profond. Le processus se déroule comme suit :

1. **Initialisation** : GenDR identifie le nœud principal étiqueté *main* dans la RSém et le convertit en racine de l'arbre RSyntP. Puis, le nœud racine est lexicalisé.
2. **Sélection du patron de régime** : GenDR consulte le dictionnaire *lexicon* et sélectionne pour le nœud racine un patron de régime (Nous allons présenter les dictionnaires dans §1.3.2). Si plusieurs patrons de régime sont applicables pour la racine, chacune construira une RSyntP distincte comme paraphrase syntaxique.
3. **Création des nœuds actants** : Il crée en même temps les nœuds représentant ses actants dont l'ordre est défini dans le trait `dia` du patron de régime choisi.
4. **Contrainte des nœuds** : Les nœuds actants sont contraints par les attributs inscrits dans son GP du dictionnaire *gpcon* et par les grammèmes, tels qu'on a mentionnés plus haut (Mel'čuk, 1995).

5. **Lexicalisation profonde et expansion** : GenDR lexicalise les nœuds contraints et sélectionne pour chacun d'eux un patron de régime. Il va répéter les étapes de 2 à 5 jusqu'au moment où le patron de régime des nœuds ne permet plus de créer de nouvel actant.
6. **Finalisation** : Le processus se termine lorsqu'aucun des patrons de régime sélectionnés ne génère de nouveaux actants. On a alors obtenu la RSyntP.

La deuxième transition, quant à elle, permet de convertir la RSyntP en RSyntS. Elle est moins compliquée que la première. Voici son déroulé :

1. **Reproduction et lexicalisation de surface des nœuds** : GenDR reproduit les nœuds de la RSyntP d'entrée en les créant dans la nouvelle structure et en les lexicalisant.
2. **Établissement des relations syntaxiques** : GenDR établit les relations syntaxiques entre les nœuds en se basant sur les arcs d'entrée.
3. **Ajout des déterminants et des mots-outils** : En effet, cette étape fait partie de la lexicalisation. Voir le point « Lexicalisation » pour plus de détails. On garde ici une trace pour raison d'intégralité du processus.

Lexicalisation

La lexicalisation est le processus qui consiste à spécifier toutes les lexies pouvant exprimer un sens donné (Polguère, 1990; Wanner, 1992). Ce processus est essentiel dans les trois niveaux de représentation utilisés par GenDR, chacun impliquant un module de lexicalisation distinct.

Au niveau de la RSém, la lexicalisation vise à sélectionner les Sens qui seront directement exprimés par des lexèmes au niveau syntaxique profond (Polguère, 1990).

La transition entre la RSém et la RSyntP modélise la lexicalisation profonde. Cette étape consiste à choisir une lexie profonde pour exprimer un sémantème donné. Lorsque plusieurs lexies profondes sont valides pour un même sémantème, GenDR renvoie alors autant de RSyntP distinctes, qui sont des paraphrases lexicales. La correspondance entre la lexie profonde et le sémantème est encodée dans le dictionnaire *semanticon* (voir §1.3.2).

La transition entre la RSyntP et la RSyntS implique la lexicalisation de surface. Ce processus consiste à exprimer chaque lexie profonde par un lexème de surface dans la structure de sortie ; il comporte concrètement 5 types de lexicalisation (Lareau *et al.*, 2018) :

1. La lexicalisation simple pour les lexèmes. Elle est la forme de lexicalisation la plus commune.

2. La lexicalisation par patron pour les locutions (Dubé et Lareau, 2022).
3. La lexicalisation liée pour les collocations (Lambrey et Lareau, 2015).
4. La lexicalisation par classe. Par exemple, les noms propres portant l'attribut `class= proper_noun` font l'objet de ce type de lexicalisation.
5. La lexicalisation de secours pour les noms inconnus. Il s'agit des mots qui n'existe pas dans le dictionnaire *semanticon*.

En outre, un aspect important de la lexicalisation est d'ajouter les mots fonctionnels, ces éléments grammaticaux qui ne portent pas de sens lexical mais qui sont indispensables à la structure de la phrase. Plus précisément, GenDR associe le déterminant approprié à chaque nom, à l'exception des noms propres. Lorsque le patron de régime du verbe l'exige, GenDR insère dans la RSyntS la préposition ou la conjonction définie dans le GP. Par exemple, la transformation de la RSyntP (fig. 1.4) en RSyntS (fig. 1.7) présentée précédemment met en évidence l'insertion de la préposition À définie dans le patron de régime de DONNER, laquelle n'apparaît pas en syntaxe profonde, entre le gouverneur DONNER et son dépendant MARIE. Lareau *et al.* (2018) considère cette opération comme le 6ème type de lexicalisation :

6. La lexicalisation grammaticale pour les mots fonctionnels.

1.4 Verbes à montée et à contrôle

Puisque nous avons pour but de modéliser les verbes à montée et à contrôle dans le cadre de GenDR, il est indispensable de présenter leurs caractéristiques. Les verbes à montée et les verbes à contrôle peuvent prendre un complément infinitif. Cependant, selon l'analyse syntaxique traditionnelle, leurs analyses syntaxiques diffèrent significativement. Dans cette section, nous examinerons en détail les particularités de chacune de ces constructions.

1.4.1 Verbes à contrôle

Dans la construction des verbes à contrôle, le sujet de l'infinitive est un pronom phonétiquement nul (PRO) qui fait référence à un antécédent dans la phrase matrice (Martin, 2001). L'analyse traditionnelle des verbes à contrôle propose la présence du PRO[2] dans la position du sujet de l'infinitive. PRO porte les mêmes traits sémantiques et grammaticaux que

2. Il est nécessaire de noter que certaines nouvelles approches analysent les verbes à contrôle avec un déplacement (Hornstein, 1999).

son antécédent en raison de la coindexation. Donc, l'interprétation de PRO dépend de son antécédent.

PRO peut être contrôlé par le sujet, comme en (1), le complément direct comme en (2) ou même par le complément indirect de la phrase matrice, comme en (3). Ce sont les phrases construites autour des verbes à contrôle qui déterminent le choix de l'antécédent. Il se peut que deux constituants nominaux remplissent en même temps la fonction d'antécédent. Dans ces situations, le choix de l'antécédent est laissé à l'interlocuteur, car les phrases construites selon la coindexation entre PRO et l'antécédent issues des interprétations distinctes sont grammaticales. Par exemple, en (4), la relecture du mémoire peut être effectuée par *Christophe* seul, par *François* seul ou par eux ensemble. Dans la dernière interprétation, PRO porte les indices i et j, ce qui indique qu'il a acquis le trait pluriel.

(1) *Christophe$_i$ ne sait pas PRO$_i$ rédiger le mémoire.*

(2) *François a encouragé Christophe$_i$ à PRO$_i$ apprendre la rédaction.*

(3) *Christophe a demandé à François$_i$ de PRO$_i$ lire son mémoire.*

(4) *Christophe$_i$ a proposé à François$_j$ de PRO$_{i/j/i+j}$ relire son mémoire.*

1.4.2 Verbes à montée

Les verbes à montée peuvent aussi avoir pour complément une infinitive. Pourtant, selon l'analyse traditionnelle, PRO n'est pas présent dans leur construction. Le critère thématique, mis de l'avant dans le cadre de la théorie du gouvernement et du liage (Chomsky, 1981), nous permettait d'exclure facilement sa présence. Le critère thématique, aussi appelé critère thêta, assure la grammaticalité d'une phrase. Il repose sur deux conditions (Tellier, 2016) :

1. Tout argument doit être associé à un rôle thématique approprié et à un seul.
2. Tout rôle thématique doit être associé à un argument approprié et à un seul.

Les verbes à montée se caractérisent par l'incapacité d'assigner le rôle thématique externe, comme l'a noté Radford (1988). Si PRO occupait la position sujet de l'infinitive, le sujet de la phrase matrice ne recevrait pas de rôle thématique, ce qui contreviendrait à la première condition du critère thématique. Cette situation problématique est illustrée par l'exemple (5).

Le verbe *écrire* a assigné le rôle thématique d'Agent au sujet de l'infinitive PRO ; le verbe *sembler* a attribué son rôle thématique unique, celui de Thème à TP. Comme il n'y a plus de rôles thématiques à assigner, le sujet de la phrase matrice *Christophe* ne reçoit aucun rôle

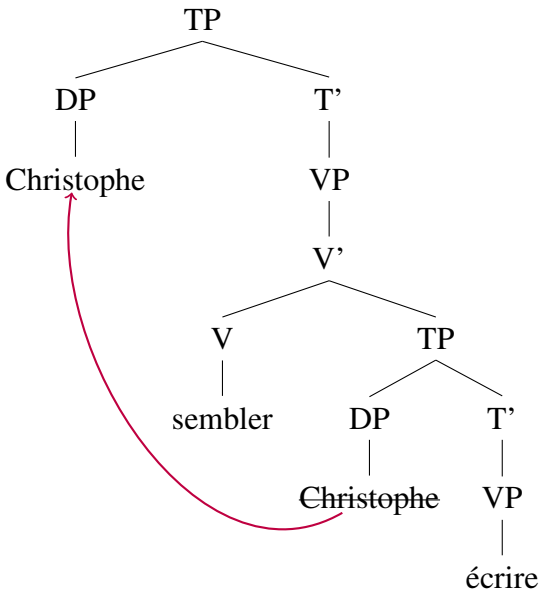


FIGURE 1.9 – L’arbre syntaxique de la phrase (6)

thématique. Cette situation exclurait, à tort, la phrase (5). Donc, cet arbre syntaxique abrégé n’est pas bon ; la présence de PRO dans cette construction est exclue.

(5) Christophe semble TP PRO écrire mal.
Thème Agent

Les verbes à montée présentent une structure syntaxique particulière. Initialement, le verbe à montée ne prend pas de sujet. Le sujet de la phrase matrice provient en effet de la phrase enchâssée. Il se déplace vers la position sujet de la phrase matrice, comme une montée. Cette opération est connue sous le nom de déplacement du DP (Chomsky, 1973), comme l’illustre la figure 1.9. Puisque le sujet de surface des verbes comme *sembler* monte à partir d’une position enchâssée, on les appelle verbes à montée du sujet, souvent abrégé en « verbe à montée ».

Une autre propriété des verbes à montée est qu’ils peuvent souvent prendre un sujet impersonnel, par exemple, *il* explétif, ce qui témoigne également de leur caractéristique de l’absence du rôle thématique externe. Dans cette construction, puisque *il* explétif occupe la position sujet de la phrase matrice, le sujet de la phrase enchâssée reste à sa place initiale, comme le démontre la phrase (7).

Le déplacement du DP permet d’établir une relation entre deux constructions avec les

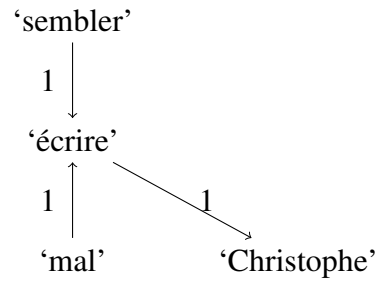


FIGURE 1.10 – RSém des énoncés (8)

verbes à montée. Comme Tellier (2016) l'explique, elles diffèrent dans la mesure où le déplacement du sujet enchâssé a eu lieu dans la construction (6), mais pas dans celle avec le pronom explétif (7). Notre objectif sera donc de modéliser ces deux constructions dans le cadre de GenDR en nous appuyant sur cette analyse syntaxique.

(6) *Christophe semble écrire mal.*

(7) *Il semble que Christophe écrive mal.*

1.4.3 Verbes à montée et à contrôle dans les structures de dépendances

Dans la perspective de la syntaxe de dépendances (Mel'čuk et Polguère, 2009; Mel'čuk, 2011), les verbes à montée et à contrôle sont analysés d'une manière similaire.

Verbes à montée

(8) a. *Il semble que Christophe écrive mal.*
b. *Christophe semble écrire mal.*

La montée est une opération syntaxique, dans laquelle la lexie L étant l'actant syntaxique **i** de L_1 devient l'actant syntaxique **j** de L_2. Les énoncés en (8) véhiculent un sens identique représenté par la RSém dans la figure 1.10. On peut y constater que le sémentème prédicatif ‘sembler’ n'a que l'actant ‘écrire’ étiqueté **1**. Cette RSém peut naturellement se dériver en RSyntP illustrée par la structure gauche de la figure 1.11, associée à l'énoncé (8-a). Dans le cadre de la dépendance syntaxique, l'énoncé (8-a) peut se transformer en (8-b) représentée par la structure droite (fig. 1.11), si la lexie CHRISTOPHE se déplace du verbe subordonné ÉCRIRE dans la position du premier actant du verbe principal SEMBLER. Cette transformation est une montée, comme l'illustre le contraste des deux structures juxtaposées dans la figure

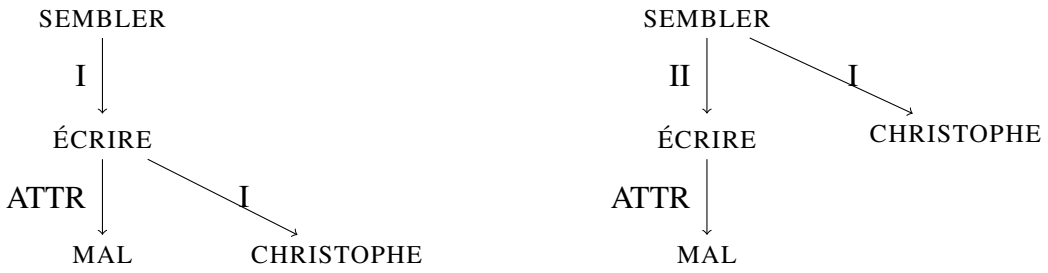


FIGURE 1.11 – RSyntP illustrant la montée du sujet vers la position de sujet en (8-b)

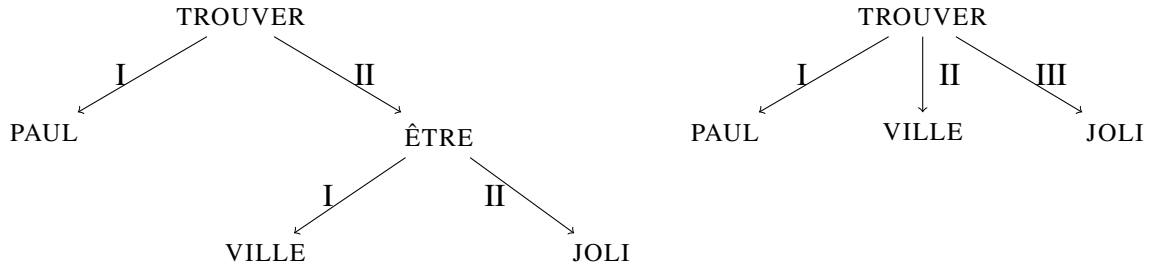


FIGURE 1.12 – RSyntP illustrant la montée du sujet vers la position d'objet en (9)

1.11.

En outre, des verbes peuvent prendre le sujet monté comme leur complément d'objet. Pour mieux comprendre cette montée particulière, prenons l'exemple de l'énoncé (9-a). Dans cet énoncé, le verbe *trouver* gouverne PAUL, VILLE et JOLI à la fois. Un aspect intéressant de cette construction est son équivalence sémantique avec une forme plus développée en (9-b). D'après la syntaxe de dépendances, l'énoncé (9-a) dérive de l'énoncé (9-b) par la montée de la lexie VILLE. Plus précisément, cette lexie se détache du verbe subordonné ÊTRE et se rattache à la lexie TROUVER comme son complément d'objet. Cette dérivation est représentée dans la figure 1.12.

(9) a. *Paul trouve la ville jolie.*
b. *Paul trouve que la ville est jolie.*

Verbes à contrôle

Dans l'approche des dépendances syntaxiques, on établit une relation de coréférence entre le DP contrôleur et le DP contrôlé, pour analyser les structures avec des verbes à contrôle. Cette méthode permet de représenter le sujet d'un verbe à l'infinitif par un nœud qui est coréférentiel à l'actant sujet comme en (10-a) ou objet du verbe principal comme en (10-b).

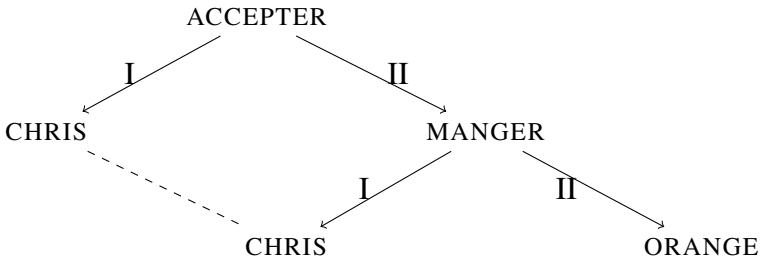


FIGURE 1.13 – RSyntP illustrant la coréférence entre le sujet du verbe à contrôle et celui du verbe subordonné en (10-a)

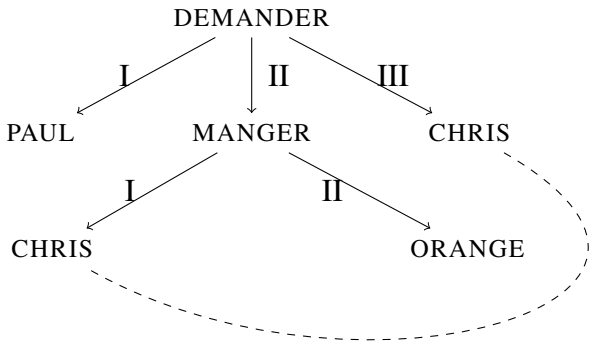


FIGURE 1.14 – RSyntP illustrant la coréférence entre l'objet du verbe à contrôle et le sujet du verbe subordonné en (10-b)

(10) a. *Chris accepte de manger une orange.*
b. *Paul demande à Chris de manger une orange.*

1.5 Synthèse

Dans ce chapitre, nous avons brièvement présenté GenDR et les verbes à montée et à contrôle pour faciliter la compréhension de leur implémentation dans les prochains chapitres.

Nous avons vu que GenDR, fondé sur la théorie Sens-Texte (TST), opère via un transducteur de graphes. Il procède par étapes, en allant d'une RSém à une RSyntS, en passant par une RSyntP. Pour assurer ses bonnes transformations, GenDR est doté des règles de lexicalisation et d'arborisation, ainsi que de trois dictionnaires indispensables : *lexicon*, *semticon* et *gpcon*.

Les verbes à montée et à contrôle que l'on souhaite implémenter présentent des comportements syntaxiques particuliers. Selon la syntaxe traditionnelle, les verbes à montée, comme *sembler*, ont la particularité de déplacer le sujet de la phrase enchâssée vers la phrase matrice. Quant aux verbes à contrôle, ils impliquent un PRO comme sujet du verbe à l'infinitif, qui établit une relation de coréférence avec son antécédent dans la phrase matrice. Dans la perspective de la syntaxe de dépendance, pour les verbes à montée, le premier actant du verbe subordonné se déplace vers la position du premier actant du verbe principal. Pour les verbes à contrôle, le premier actant du verbe subordonné est coréférentiel au premier actant du verbe principal.

Le chapitre suivant exposera la méthodologie pour la construction des dictionnaires pour les verbes à montée et à contrôle.

Chapitre 2

Extraction de données et construction des dictionnaires

L'implémentation des verbes à montée et à contrôle dans GenDR nécessite des ressources lexicales riches et adaptées. Ce chapitre présentera la méthodologie pour la construction des dictionnaires adaptés à GenDR, *gpcon* et *lexicon*. Comme le titre du chapitre le suggère, la démarche se divise en deux parties : d'une part, l'extraction des phrases contenant les verbes à montée et à contrôle à partir des corpus annotés, à l'aide de l'outil Grew ; et d'autre part, l'élaboration des dictionnaires adaptés à GenDR et fondés sur les données extraites.

2.1 Corpus : le format SUD

Le module traitant les verbes à montée et à contrôle requiert une ressource lexicale riche. Elle permet d'augmenter la couverture de notre réalisateur aussi bien pour le français que pour l'anglais, possiblement pour d'autres langues. Cette ressource lexicale doit avoir la capacité d'être convertie en dictionnaires : lexical et de patrons de régime. Comme expliqué en §1.3.2, le dictionnaire de patrons de régime en tant que dictionnaire de comportements syntaxiques doit comprendre quasi exhaustivement les patrons de régime des verbes en question. Alors, ce critère de richesse s'impose dans la sélection de la ressource lexicale.

Pourtant, nous n'avons pas trouvé une telle ressource lexicale pour les verbes à montée et à contrôle, ni même une liste exhaustive de ces verbes. Face à cette difficulté, nous avons dû adapter notre approche : nous construisons notre propre ressource lexicale à partir des corpus annotés disponibles. L'utilisation de corpus annotés facilite l'extraction et la manipulation des données.

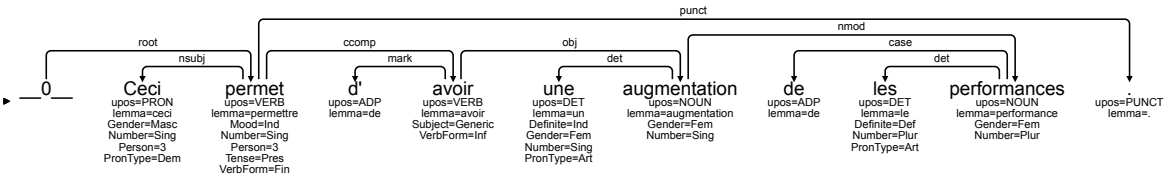


FIGURE 2.1 – Annotation d'UD pour la phrase fr_gsd−ud−train.conllu@ fr−ud−train_09680

Initialement, nous avons envisagé les schémas d'annotation Universal Dependencies (UD). Il s'agit d'un projet pour développer une annotation de corpus arborés multilingue. Ses annotations se basent sur l'évolution des *Stanford dependencies* (de Marneffe *et al.*, 2006; de Marneffe et Manning, 2008; de Marneffe *et al.*, 2013), les étiquettes des parties du discours de Google (Petrov *et al.*, 2012) et *Interset* interlingua pour les étiquettes morphosyntactiques (Zeman, 2008). Ce projet de la communauté ouverte contient plus de 200 corpus dans plus de 150 langues.

Cependant, malgré l'abondance de corpus disponibles, nous avons constaté que les étiquettes et la philosophie d'annotation syntaxique de surface d'UD ne correspondent pas à celles qu'on utilise dans le cadre de GenDR, notamment pour des mots fonctionnels. Par exemple, la première préposition *de* dans l'énoncé annoté (*Ceci permet d'avoir une augmentation des performances.*), dont l'arbre est illustré en 2.1. Cette préposition qui doit relier syntaxiquement le verbe principal et le verbe subordonné est régie par ce dernier via la relation étiquetée par `mark`. Cette structure est différente de celle employée dans la RSyntS, où l'on trouve **permettre** −**comp_obj**→ **de** −**comp_prep**→ **avoir**. Donc, l'utilisation de UD nécessiterait une adaptation manuelle, bien que possible, mais minutieuse et laborieuse.

De plus, un autre inconvénient majeur de la méthodologie d'annotation d'UD a remis en question son utilisation pour notre projet. Elle ne permet pas de distinguer les verbes à contrôle des verbes à montée dans ses structures arborescentes, comme le montre la figure 2.2. On voit que, par exemple, les phrases construites autour de *permettre* (verbe à contrôle) et de *commencer* (verbe à montée) ont été attribuées à une structure arborescente identique dans UD. L'absence de distinction complique leur séparation et, par conséquent, l'établissement de *gpcon*, étant donné que les GP des verbes à montée et à contrôle diffèrent complètement. Pour ces raisons, nous avons dû exclure UD comme ressource lexicale.

Nous nous sommes ensuite tournés vers Surface-Syntactic Universal Dependencies (SUD)

FIGURE 2.2 – Absence de distinction dans les annotations d'UD entre les verbes à contrôle et à montée

(Gerdes *et al.*, 2018a, 2019, 2022). SUD est aussi un format d'annotation pour les corpus arborés de dépendances syntaxiques. Contrairement à UD, il traite les adpositions, les conjonctions, les auxiliaires et les copules comme têtes syntaxiques, ce qui est plus cohérent avec les théories syntaxiques majeures, comme la figure 2.3 l'illustre.

> UD parts with surface syntax criteria and applies the criterion of "content word as head" whereas surface syntax uses distributional criteria of each individual word. The main criterion is that the surface syntactic head determines the distribution of the unit. (Gerdes *et al.*, 2018a, p. 3)

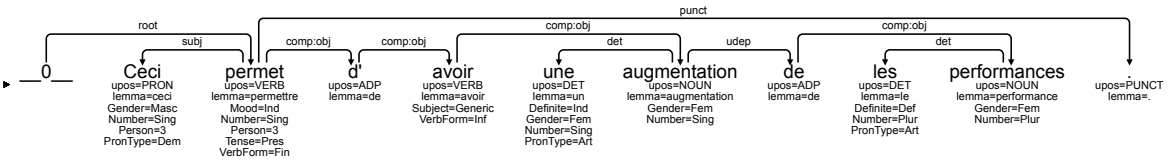


FIGURE 2.3 – Annotation de SUD pour la phrase fr_gsd−sud−train.conllu@fr−sud−train_09680

Un avantage significatif de SUD est que la plupart de ses étiquettes sont similaires à celles employées dans GenDR, ce qui facilite potentiellement l'intégration et réduit le travail manuel. Par ailleurs, la méthodologie d'annotation de SUD permet de séparer les verbes à montée et à contrôle comme la figure 2.4 le montre. Dans les deux structures gauches, *permettre* et *vouloir* gouvernent (directement et indirectement) leur verbe subordonné via la relation `comp:obj`. En revanche, *commencer* et *sembler* des structures droites le font à travers la relation `comp:obl` ou la relation `comp:pred`. On bénéficie de cette différence d'étiquettes de dépendance pour les distinguer car les annotations de SUD n'expriment pas explicitement PRO ou le déplacement du DP.

Cependant, cette distinction n'est pas absolue. Il arrive que des constructions impliquant des verbes à montée et à contrôle soient annotées identiquement. Un exemple concret est

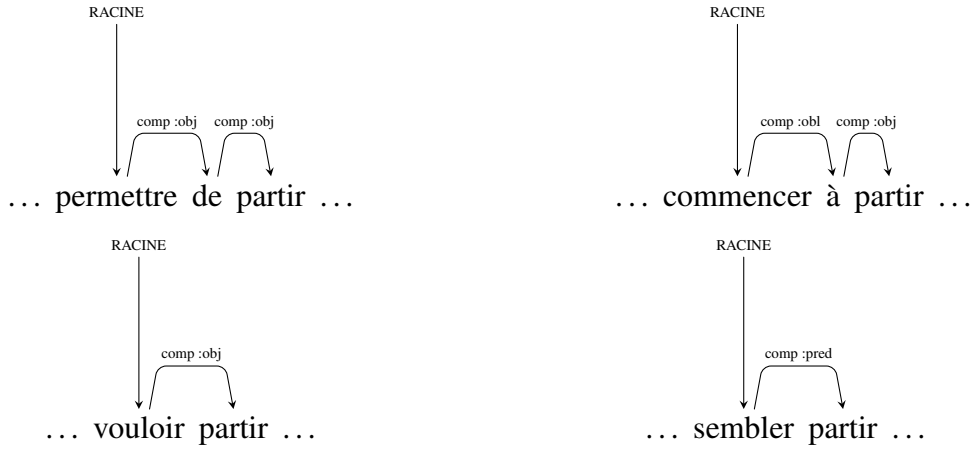


FIGURE 2.4 – Distinction des annotations en SUD pour les verbes à contrôle et à montée

FIGURE 2.5 – Comparaison des annotations en SUD pour les verbes *aider* (contrôle) et *commencer* (montée)

présent dans la figure 2.5, où les annotations pour le verbe à contrôle *aider* et le verbe à montée *commencer* sont identiques. Cette situation nous a conduits à adapter la méthodologie d'extraction des phrases. Notre stratégie initiale visait à extraire les phrases contenant les verbes à montée et à contrôle des corpus par leurs annotations distinctes, sans recourir à une liste prédéfinie de ces verbes. Des expériences ont démontré l'inefficacité de cette approche, révélant l'impossibilité de séparer complètement ces verbes par patron syntaxique. La liste mentionnée s'est avérée obligatoire pour l'extraction.

Finalement, nous avons opté pour une méthode hybride : l'utilisation d'une liste prédéfinie de verbes à montée et à contrôle pour aider à extraire les phrases pertinentes des corpus annotées par SUD. Les corpus utilisés seront listés en tableau 2.1. (La présente recherche s'est appuyé sur les corpus SUD en version 2.12, qui était la plus à jour au moment de notre étude. À noter que depuis, une nouvelle version 2.14 a été publiée en juin 2024.)

Nom de corpus	Références
English-Atis	Kuzgun *et al.* (2021)
English-ESLSpok	Kyle *et al.* (2022)
English-EWT	Silveira *et al.* (2014)
English-GENTLE	Aoyama *et al.* (2023)
English-GUM	Zeldes (2017)
English-GUMReddit	Behzad et Zeldes (2020); Zeldes (2017)
English-LinES	Ahrenberg (2007)
English-ParTUT	Sanguinetti et Bosco (2015)
English-PUD	Uszkoreit *et al.* (2017)
French-FQB	Seddah et Candito (2016)
French-GSD	Guillaume *et al.* (2019)
French-ParisStories	Kahane *et al.* (2021)
French-ParTUT	Sanguinetti et Bosco (2015)
French-PUD	Uszkoreit *et al.* (2017)
French-Rhapsodie	Gerdes *et al.* (2018b)
French-Sequoia	Candito et Seddah (2012)

TABLEAU 2.1 – Corpus d'extraction des phrases

2.2 Extraction de patrons de régime avec Grew

Graph Rewriting (Grew) (Bonfante *et al.*, 2018) est un système de réécriture de graphes pour traitement du langage naturel (TAL). Il peut manipuler de nombreux types de représentations linguistiques, telles que la représentation des séquences étiquetées par parties du discours, la représentation des syntaxes de dépendance de surface comme SUD, la représentation des syntaxes de dépendance profonde, et la représentation sémantique Abstract Meaning Representation (AMR) ou Dependency Minimal Recursion Semantics (DMRS). Grew offre un outil, *Grew-Matching* développé par Guillaume (2019, 2021), qui permet d'effectuer des recherches dans un corpus de structures syntaxiques à partir de requêtes. Les utilisateurs ont l'option d'utiliser Grew soit via une interface disponible en ligne [1], soit en l'installant localement pour une utilisation via la ligne de commande, offrant ainsi des options adaptées à différents besoins.

1. `https://match.grew.fr`

Listing 2.1 – Grammaire de `grep` pour chercher une requête donnée

```
$ grew grep -request <request_file> -i <input>
```

2.3 Méthodologie

2.3.1 Extraction des phrases

Puisque Grew supporte les formats d'annotation SUD et offre un outil de recherche, nous l'avons utilisé pour manipuler les corpus et en extraire des données. En dépit de l'interface disponible en ligne, nous avons installé localement Grew pour trois avantages majeurs :

— exploiter pleinement les corpus : il est important de noter que les résultats affichés en ligne se limitent à 1000 occurrences par corpus par requête.

— obtenir des résultats plus groupés : l'utilisation de la ligne de commande sous Linux permet de regrouper des occurrences correspondant à la requête de tous les corpus dans un fichier de sortie.

— fichier de sortie facile à manipuler : la version locale permet de renvoyer le fichier de sortie au format JSON plus facile à manipuler avec Python que le tableau produit par l'interface en ligne.

Ces avantages ont rendu l'outil de recherche local particulièrement attractif pour notre travail. Concrètement, nous avons effectué nos recherches en utilisant la commande `grep` de *Grew-Matching* directement depuis le terminal. Sa grammaire est présentée en 2.1.

Une requête se présente sous la forme d'un fichier au format `req` qui contient un patron de recherche, comme l'illustre le listing 2.2. La commande `grep` permet d'identifier le patron de la requête au sein des données. Le système parcourt les graphes représentant chaque phrase, à la recherche de structures correspondant exactement au patron défini. Lorsqu'une correspondance est détectée, le système l'enregistre en précisant les associations entre les nœuds du patron et ceux de la phrase. À l'issue de l'exécution, le système renvoie un fichier au format JSON. Ce fichier regroupe toutes les phrases où le patron a été identifié, accompagnées des détails sur les correspondances spécifiques pour chacune d'entre elles. Pour lancer cette opération, la commande prend en entrée deux arguments : le fichier de requête contenant le patron, ainsi que les données, comme le montre le listing 2.1. La préparation de ces données est une étape cruciale du processus, dont les détails sont expliqués ci-dessous.

Listing 2.2 – Exemple de patron pour chercher un lemme donné

```
pattern { X [lemma="aimer"] }
```

Listing 2.3 – Exemple de patron pour chercher un lemme et une relation donnés

```
pattern {
    X -[comp:obj]-> Y;
    X [lemma="aimer"]
}
```

Préparation des requêtes

La préparation des requêtes pour la recherche des phrases contenant les verbes à montée et à contrôle se déroule en deux étapes principales. La première consiste à établir des listes de ces verbes en anglais et en français, comme discuté en §2.1. Pour l'anglais, nous avons choisi la liste répertoriée dans le guide d'annotation du corpus *English Treebank* [2] de Taylor (2006), en raison de sa relative exhaustivité. Ce choix s'explique par l'absence de listes plus complètes ou véritablement exhaustives. Quant à la version française, elle a été élaborée en traduisant la liste anglaise avec des ajustements manuels nécessaires.

Avant de passer à l'étape suivant, il est important de comprendre que la requête est formulée selon la syntaxe de *Grew-Matching*, similaire aux expressions régulières. On peut chercher des phrases contenant un ou des éléments catégoriels spécifiques, tels qu'un lemme ou une relation. Le listing 2.2 montre une requête simple visant à trouver les phrases avec le lemme *aimer* représenté par la variable X. La requête en 2.3, quant à elle, permet de chercher les phrases contenant le lemme *aimer* représenté par X qui gouverne un autre nœud (Y) via la relation `comp:obj`. Si on veut chercher plusieurs relations en même temps, on doit les séparer par « | », comme *X -[comp :obj | comp :obl]-> Y* qui renverront les phrases contenant la relation `comp:obj` ou `comp:obl`.

Dans la seconde étape, nous avons procédé à la création d'une requête individuelle pour chaque verbe. Ces requêtes se composent essentiellement d'un patron de recherche. Toutefois, il est important de noter la formulation distincte entre les patrons des verbes à montée et ceux des verbes à contrôle, compte tenu de leurs différences syntaxiques. Cette distinction dans la formulation des patrons est cruciale pour assurer l'identification correcte des structures spécifiques à chaque verbe dans les corpus.

2. https://www-users.york.ac.uk/~lang22/TB2a_Guidelines.htm

Listing 2.4 – Patron du verbe à montée *begin*

```
pattern {
    * -[comp:aux | root | cc]-> G;
    G [lemma="begin"]
}
```

1. Verbes à montée : Compte tenu de la complexité de l'analyse syntaxique des verbes à montée, il est difficile de séparer les lexies impliquant la montée au sein de leur vocable. Un vocable est un regroupement de lexies qui partagent le même signifiant mais qui se distinguent sémantiquement, alors qu'une lexie est une unité lexicale associée à un sens spécifique. Par exemple, le vocable MONTER contient entre autres deux lexies :
   MONTER$_{I.1}$ Se déplacer dans un mouvement de bas en haut, vers un lieu plus haut.
   MONTER$_{II.1}$ S'élever dans l'air, dans l'espace.
   Cette contrainte nous oblige d'élargir la cherche à toutes les entrées contenant le verbe en question. Par conséquent, le patron de recherche ne contient qu'un nœud qui contraint le verbe principal (G) comme en 2.4. Cependant, cette méthode entraîne malheureusement un travail manuel important lors du nettoyage.
2. Verbes à contrôle : Puisque nous nous concentrons exclusivement sur le phénomène de contrôle, nous écartons les autres entrées non pertinentes. Dans ce cas-ci, le patron de recherche pour les verbes à contrôle se décline en deux cas possibles :
   — soit le verbe principal (G) ne nécessite pas de préposition entre le verbe subordonné (D) et lui-même. Le patron inclut alors deux nœuds qui les contraignent respectivement, ainsi que les relations les liant.
   — soit le verbe principal (G) requiert une préposition (D) pour établir le contrôle sur le verbe subordonné. Le patron comporte alors deux nœuds qui contraignent le verbe principal (G) et la préposition (D), ainsi que les relations les liant.
   Heureusement, les deux cas peuvent être combinés dans une unique requête en spécifiant les options alternatives pour la patrie du discours du dépendant (D) et pour la relation entre le verbe principal (G) et ce dépendant, comme en 2.5.

Préparation des données

Grew accepte deux modes d'entrée de données : *Mono corpus* et *Multi corpora*. Puisque nous avons l'ambition d'extraire les phrases contenant le verbe cible de tous les corpus français et anglais, le mode *Multi corpora* a mieux répondu à notre besoin. Ainsi, nous avons

Listing 2.5 – Patron du verbe à contrôle *allow*

```
pattern {
  * -[comp:aux|root|cc]-> G;
  G [lemma="allow"];
  G -[comp:obl|comp:obj]-> D;
  D [upos=PART|VERB]
}
```

```
{
  "corpora": [
    {
      "id": "SUD_English-Atis",
      "directory":
 ↪  "/mnt/hgfs/share/sud-treebanks-v2.12/SUD_English-Atis",
      "files": [
        "en_atis-sud-dev.conllu",
        "en_atis-sud-test.conllu",
        "en_atis-sud-train.conllu"
      ]
    }
  ]
}
```

FIGURE 2.6 – Extrait du répertoire de corpus `eng_fre.json`

développé un script en Python. Il permet de compiler les informations essentielles sur les corpus (nom, chemin et fichiers au format CONLLU inclus) dans un fichier JSON. Comme l'exemple fourni sur le site de Grew, nous avons nommé ce fichier `eng_fre.json` utilisant les abréviations des langues impliquées. La figure 2.6 montre un extrait du fichier. Nous disposions ainsi des deux arguments nécessaires pour l'appel de fonction.

Avant d'exécuter la fonction `grep`, nous devions faire charger au préalable le fichier `eng_fre.json` à Grew en utilisant la commande `compile`. Nous pouvions ensuite soumettre chaque requête comme argument à la fonction `grep` et sauvegarder les résultats dans un nouveau fichier au format JSON sous le nom du verbe cherché. Les commandes évoquées sont comme suit :

```
$ grew compile -i eng_fre.json
$ grew grep -request accepter.req -i eng_fre.json > accepter.json
```

Étant donné le nombre important de requêtes de verbes à traiter individuellement dans

la liste, l'utilisation d'une boucle s'est avérée efficace et importante pour optimiser nos opérations et éviter les répétitions. Pour ce faire, nous avons développé un fichier `batch` qui optimise le processus. Il permet de charger les données une seule fois et d'appeler la fonction `grep` en répétition dans une boucle `for`. La boucle itère à travers toutes les requêtes de verbes, compilant les résultats de chacune dans un fichier JSON distinct. Cette approche automatise et simplifie considérablement le traitement des multiples requêtes.

2.3.2 Construction des dictionnaires

Structure des résultats de Grew

```
"SUD_English-ParTUT": [
{
  "sent_id": "en_partut-ud-1950",
  "matching": { "nodes": { "G": "15", "D": "17" }, "edges": {} }
},
{
  "sent_id": "en_partut-ud-478",
  "matching": { "nodes": { "G": "2", "D": "4" }, "edges": {} }
},
{
  "sent_id": "en_partut-ud-296",
  "matching": { "nodes": { "G": "12", "D": "14" }, "edges": {} }
}
```

FIGURE 2.7 – Extrait du fichier de sortie suivant la requête *allow*

Les résultats JSON englobent toutes les phrases contenant le verbe cible, à partir desquelles nous devons identifier ses différents patrons de régime. Avant de procéder à la construction des dictionnaires, nous allons présenter la manière d'exploiter ces résultats JSON.

Ces résultats en JSON servent en effet de répertoire nous permettant de localiser précisément les phrases et les nœuds pertinents au sein des corpus. Chaque fichier des résultats est structuré de manière hiérarchique, comme la figure 2.7 le montre. Au niveau supérieur, on trouve les noms des corpus, tel que `SUD_English-ParTUT`. Sous chaque corpus, une liste contient des dictionnaires, chacun représentant une phrase correspondant au patron de recherche. Chaque dictionnaire de phrase inclut des informations clés comme l'identifiant de la phrase (`sent_id`) et un sous-dictionnaire `matching` qui précise les numéros des tokens correspondant aux nœuds définis dans le patron de recherche. Ces numéros représentent en fait l'identifiant du lemme dans une phrase de surface du corpus annoté.

Dans les sous-sections suivantes, nous allons présenter la méthodologie adoptée de la manipulation des résultats JSON. Elle consiste à retrouver, dans le corpus correspondant, la phrase identifiée dans les résultats JSON, puis à révéler le patron de régime en filtrant les dépendants non pertinents du verbe cible au sein de cette phrase.

Manipulation des résultats de Grew

D'abord, une question se pose : comment localiser efficacement une phrase spécifique au sein du corpus volumineux. Cette tâche peut être décomposée en deux étapes principales :

1. Localisation du corpus pertinent dans le volumineux dossier SUD. Face à l'éparpillement complexe des corpus dans le dossier SUD, nous avons bénéficié du répertoire `eng_fre.json` précédemment établi. Ce fichier nous a permis d'accéder directement aux corpus via les chemins d'accès indiqués.
2. Repérage de la phrase cible au sein du corpus. Pour cette étape, nous avons utilisé la librairie `pyconllu` dédiée à charger et manipuler facilement les corpus annotés au format `CoNLL`. Elle permet de charger tout le corpus dans une liste, où chaque élément est un objet `Sentence` représentant une phrase du corpus. Chaque objet `Sentence` comporte l'identifiant de la phrase comme attribut `id` et ses commentaires associés, ainsi que des objets `Token` représentant le noyau d'un objet `Sentence` stockent les annotations. L'attribut `id` de l'objet `Sentence` équivaut à `sent_id` de la phrase associée dans le corpus `CoNLL`.

   Pour localiser une phrase issue des résultats au sein du corpus, nous pouvons comparer l'attribut `id` du objet `Sentence` avec le `sent_id` recherché. S'ils sont identiques, la phrase est trouvée. Deux approches s'offraient à nous pour cette recherche :
   — La boucle `for` : Cette méthode parcourt chaque objet `Sentence` du corpus, jusqu'à trouver une correspondance d'identifiant. Pourtant, étant donné que le nombre important de phrases dans un corpus (généralement plusieurs milliers) et le fait que chaque itération commence obligatoirement depuis le premier élément, nous nous sommes interrogés sur l'efficacité de cette approche. Sa complexité temporelle serait légèrement inférieure à $O(n)$ dont n est égal au nombre de phrases à chercher.
   — L'utilisation d'un dictionnaire : Cette alternative consiste à convertir les objets `Sentence` du corpus en un dictionnaire à l'aide de la compréhension de dictionnaire, utilisant leur identifiant comme clé et l'objet `Sentence` comme valeur. Puisque la complexité temporelle amortie du dictionnaire n'est que $O(1)$, nous

pouvons localiser très rapidement la phrase correspondante dans le dictionnaire, par la clé ou par la méthode `dict.get('sent_id')`.

Les résultats JSON d'un verbe impliquent souvent des phrases issues de plusieurs corpus. Étant donné le rapport entre le nombre de verbes et de corpus, soit 149 verbes versus 10 corpus pour l'anglais, et 98 verbes versus 7 corpus pour le français, il est probable que certains corpus apparaissent dans les résultats de beaucoup de verbes. Par conséquent, il faut que les dictionnaires convertis des corpus soient réutilisables.

Pour ce faire, nous avons créé un dictionnaire global où stocker les dictionnaires convertis de chaque corpus, utilisant le nom du corpus comme clé, le dictionnaire converti lui-même comme valeur. En pratique, quand le programme passe à un corpus dans les résultats JSON comme `SUD_English-ParTUT` dans la figure 2.7 pour chercher ses phrases, il vérifie si le corpus existe déjà dans le dictionnaire global. Si c'est le cas, nous récupérons directement le corpus sous forme de dictionnaire et y localise la phrase par sa clé. Dans le cas contraire, le programme appelle la fonction permettant de convertir le corpus en dictionnaire et y trouve la phrase cible. Le programme stocke également le nouveau dictionnaire converti dans le dictionnaire global pour l'usage ultérieur. Cette approche a considérablement amélioré l'efficacité du processus. Le programme ne consacre du temps à la conversion qu'une seule fois par corpus, tandis que le temps alloué à la recherche des phrases dans le dictionnaire reste minimal.

Prenons l'exemple de la phrase identifiée par `en_partut-ud-478` pour clarifier le processus. Selon les résultats JSON en 2.8, elle fait partie du corpus `SUD_English-ParTUT`. Si ce corpus n'a pas encore été converti en dictionnaire Python dans le dictionnaire global, nous procédons à sa conversion. Dans ce dictionnaire converti, chaque identifiant de phrase sert de clé, associée à l'objet `Sentence` correspondant comme valeur. Puis, nous récupérons l'objet `Sentence` avec l'identifiant de la phrase. Donc, nous avons localisé cette phrase.

```
"SUD_English-ParTUT": [
    {
      "sent_id": "en_partut-ud-478",
      "matching": { "nodes": { "G": "2", "D": "4" }, "edges": {} }
    },
  ...
]
```

FIGURE 2.8 – Extrait des résultats JSON suivant la requête *allow*

Une fois la phrase localisée dans le corpus, on obtient l'objet `Sentence` correspondant et

l'assigne à la variable `sent_target`. Les nœuds importants sont aussi faciles à accéder grâce à leurs identifiants, comme ceux montrés dans la figure 2.8. Par exemple, pour `"nodes": {"G": "2", "D": "4" }`, l'index du verbe principal est 2, et 4 correspond au dépendant. Comme les objets `Token` sont indexés dans l'objet `Sentence`, nous pouvons obtenir un objet `Token` spécifique par index. Ensuite, l'objet `Token` du verbe principal a été affecte à la variable `token_target`.

Préparation des `Token` pour patron de régime

Dans un souci d'organisation et de lisibilité du code, nous avons procédé à une restructuration, en encapsulant les fonctionnalités de recherche précédemment décrites dans la classe `GrewResultsAnalyser`. Elle prend en entrée trois arguments : les résultats en JSON, le répertoire de corpus et la langue. Alors, chaque objet `GrewResultsAnalyser` représente les résultats associés à un verbe. Ce qui est le plus important de cette classe est qu'elle renvoie un ensemble d'objets `GovPattern` dont chacun représente un patron de régime associé à une phrase des résultats. C'est-à-dire que cet ensemble englobe les GP que le verbe associé utilise.

Concrètement, la classe `GovPattern` est destinée à la construction et la représentation du patron de régime. Chaque objet `GovPattern` possède plusieurs attributs, notamment le nom de GP et son tableau. L'initialisation d'un objet nécessite trois arguments : la phrase sous forme d'objet `Sentence`, le verbe cible sous forme d'objet `Token` et la langue spécifiée.

La construction d'un objet `GovPattern` se déroule en quatre étapes :

1. Filtrage des dépendants. L'objectif de cette étape est de garder seulement les dépendants pertinents pour le GP. Le processus débute par le filtrage des dépendants du verbe en fonction de leur relation de dépendance, représenté par attribut `deprel` des objets `Token`. Cette opération se fait en parcourant les objets `Token` de l'objet `Sentence`, en excluant les `deprel` spécifiques. Ces dernières ont été manuellement collectées au préalable dans un ensemble.

   Dans un schéma de SUD, les verbes à contrôle et à montée gouvernent le verbe subordonné par intermédiaire d'une préposition. En conséquence, le verbe subordonné ne s'attache pas à son gouverneur dans la structure syntaxique de surface. Néanmoins, il est important d'intégrer la préposition et le verbe subordonné dans le patron de régime. Pour prendre en compte de cette complexité, le filtrage s'opère en deux étapes :

   (a) On sélectionne les dépendants directs en excluant les `deprel` à ignorer ;

(b) On sélectionne les dépendants de la préposition, si celle-ci figure parmi les dépendants directs.

Tous les dépendants sélectionnés forment alors la liste des candidats pour l'objet `GovPattern`.

2. Restauration de la préposition. Cette étape consiste à restaurer la préposition lorsque l'ordonnance a replacé un pronom dans sa position initiale au sein de la liste des candidats. Par exemple, la phrase *on lui donne un livre* doit être transformée en *on donne un livre à lui*. Pour accomplir cette tâche, la classe `GovPattern` intègre une fonction spécialisée qui insère la préposition *à*, sous forme d'objet `ComparableToken`, en lui assignant des valeurs spécifiques pour ses attributs, tels que `id`, `deprel`, `lemma`, etc. Cet objet est ensuite inséré juste avant le pronom replacé.

   Cependant, cette opération de restauration n'est pas appliquée systématiquement. Elle est omise pour certains pronoms tels que `y`, `en`, `dont` et `où`, qui représentent généralement des syntagmes adverbiaux que nous ignorons dans notre analyse. Puisque cette liste de pronoms, spécifique au français, peut varier selon la langue étudiée, elle est stockée dans un fichier de configuration externe, `config.ini`, accessible via le module `configparser`. Cette approche permet une personnalisation des configurations linguistiques selon les besoins spécifiques de chaque langue analysée.

3. Gestion des phrases à sujet nul. Une considération supplémentaire dans la construction de l'objet `GovPattern` concerne le traitement des phrases à sujet nul. Dans ces structures de surface, le verbe peut apparaître sans gouverner un sujet, ce qui pourrait conduire à la génération d'un patron de régime incomplet, tandis que d'un point de vue syntaxique, chaque GP doit être associé à un sujet, à l'exception des verbes à montée.

   Pour assurer l'intégrité syntaxique du patron de régime, nous avons implémenté une fonction qui permet de créer un nouvel objet `ComparableToken` représentant le sujet manquant et de l'insérer en première position dans la liste des candidats.

4. Ordonnancement des candidats. La dernière étape consiste à ordonner les candidats de la liste et l'objet `Token` du verbe cible, pour préparer la construction de l'objet `GovPattern`. Cette tâche se complique particulièrement en français, où les pronoms peuvent se déplacer, entraîne l'inefficacité de l'identifiant de l'objet `Token`, qui représente sa position finale, pour l'ordonnance. Pour surmonter ce défi, nous avons ordonné les candidats suivant leur `deprel`. Cependant, comme les objets `Token` ne sont pas directement comparables en raison de l'absence de méthodes pertinentes, nous avons créé une nouvelle classe, nommée `ComparableToken`. Cette classe, dérivée de `Token`, est pourvue d'un dictionnaire `deprel-valeur` et de l'opérateur `__lq__` nouvellement

défini, remplaçant celui de la classe `Token`. Ces ajouts permettent la comparaison des objets `ComparableToken`, facilitant ainsi leur ordonnance à l'aide de la méthode simple `sorted()`.

Cette approche en quatre étapes garantit que le programme traite correctement même les structures syntaxiques complexes et assure la cohérence entre elles et leur patrons de régime renvoyés.

Continuons l'exemple précédent de la phrase `en_partut-ud-478` gouverné par *allow* en (1), pour illustrer ce processus complexe. Nous identifions d'abord ses dépendants pertinents pour le patron de régime. Puisqu'on a obtenu l'objet `Sentence` de la phrase et les `Token` importants lors de la dernière étape, on filtre les dépendants du verbe cible pour garder seulement ceux pertinents comme candidats. Pour cette phrase, on retient les objets `Token` de *Facebook* (Proper Noun), *member* (Noun), *to* (Part) et *send* (Verb). Ensuite, on les convertit avec l'objet `Token` de *allow* en objets `ComparableToken` et stocke tout dans la liste des candidats. Puisque cette phrase a un sujet et n'implique pas de pronom déplacé, on ordonne les candidats en fonction de leur attribut `deprel`. Les `Token` pertinents et ordonnés sont alors prêts à former le GP de la phrase.

(1) *Facebook allows members to send invites to their contacts by entering an email address or by uploading their contacts.*

Une fois les candidats ordonnés, nous procédons à la mise à jour des attributs des `Token` sélectionnés dans la liste, car les objets `GovPattern` doivent être uniformes. Les points clés de la procédure de mise à jour sont les suivants :

1. Traitement du sujet explétif. Si un objet `Token` représente un sujet explétif, son `upos` (partie du discours) est remplacé par `Exp` et son `deprel` par `subj`. Cette modification permet de distinguer clairement les sujets explétifs dans les noms du GP.
2. Normalisation des parties du discours nominales. Pour un objet `Token` dont la partie du discours fait partie de le groupe "pronom, nom, nom propre, nombre, symbole, déterminant", son `upos` est remplacé par `N`. Cette normalisation simplifie la représentation des éléments nominaux dans le patron de régime.
3. Gestion des mots outils. Si la partie du discours d'un objet `Token` fait partie de "préposition, particule, conjonction", son `upos` est remplacé par `prep` ou par `conj`. La nouvelle `upos` et le lemme forment un tuple `tool_word` qui est assigné comme attribut à l'objet `Token` suivant de la liste. Cet assignement est du au fait que la préposition

et la conjonction ne constituent pas elles-mêmes des actants dans le tableau du patron de régime.

4. Traitement du mode et de la finitude. Pour un objet `Token` dont la partie du discours est un verbe et une auxiliaire, on extrait son mode et/ou sa finitude qui se trouvent dans l'attribut `features` de l'objet `Token` ; puis on les assigne au même objet `Token` comme attribut `mood`.

Avant de continuer, revenons à notre exemple précédent de la phrase `en_partut-ud-478` en (1). À l'issue de la dernière étape, on a obtenu les candidats de `Token` ordonnés : *Facebook* (Proper Noun), *allow* (Verb), *member* (Noun), *to* (Part) et *send* (Verb). Maintenant, on met à jour leurs attributs :

1. *Facebook* : PropNoun -> N ;
2. *allow* : Verb ->V ;
3. *member* : Noun -> N ;
4. *to* : Part -> prep ;
5. *send* : Verb ->V ; `tool_word` : prep=to ; `mood/fin` : finiteness=INF ;

Génération des patrons de régime

Pour un objet `GovPattern`, une fois les attributs des objets `Token` dans sa liste des candidats mis à jour, nous pouvons en générer le nom et le tableau du patron de régime.

Pour **le nom de GP**, chaque candidat va contribuer la valeur de son attribut `upos` en majuscules, suivie, le cas échéant, de celle de `tool_word` ou `mood` en minuscules, en excluant les candidats dont `upos` est la préposition ou la conjonction. Ensuite, ces éléments sont assemblés à l'aide de traits de soulignement pour former le nom de GP, qui est affecté à l'attribut `name` de l'objet `GovPattern`.

Le tableau de GP, quant à lui, se présente sous forme de dictionnaire enchâssé. Nous initialisons d'abord un dictionnaire vide où on ajoute progressivement des paires clé-valeur. Les clés sont représentées par des chiffres romains ; les valeurs sont des dictionnaires internes dans lequel on décompose les attributs de l'objet `GovPattern` et insère leurs valeurs :

— Information basique : `rel = Token.deprel, dpos = Token.upos`

— Information optionnelle : en décomposant l'attribut tuple `mood` ou `tool_word` dont le premier élément devient la clé et le second est la valeur. Par exemple, `prep = de` ou `conj = que`.

Finalement, l'objet `GovPattern` prend le dictionnaire complété en tant que son attribut `gp_table_dict`.

Quant à notre « fameux » exemple de la phrase `en_partut-ud-478` en (1), l'opération lui conduit à générer le GP `N_V_N_to_Vinf` pour *allow*. La figure 2.9 illustre son tableau. Ces deux éléments (nom et tableau) sont enregistrés comme attributs dans un objet `GovPattern` associé à la phrase.

```
N_V_N_to_Vinf {
        I={rel=subj dpos=N}
        II={rel=comp_obj dpos=N}
        III={rel=comp_obl dpos=V finiteness=INF prep=to}
}
```

FIGURE 2.9 – Tableau de GP de *allow* dans la phrase `en_partut-ud-478`

Donc, chaque phrase correspond à un objet `GovPattern` pourvu des éléments les plus importants `name` et `gp_table_dict` dans ses attributs. Ensuite, il est nécessaire de regrouper ces objets dans un ensemble représentant les différents patrons de régime utilisés par le verbe cible. Alors, cet ensemble est affecté à l'attribut `gp_name_content_set` de l'objet `GrewResultsAnalyser` lié au verbe cible. Puisque chaque verbe est associé un ensemble, idéalement, en combinant ces ensembles, nous pourrons en construire facilement les dictionnaires finaux *gpcon* et *lexicon*.

Pourtant, cette méthode présente deux défauts :

1. Les phrases qui partagent le même patron de régime renvoient sûrement des objets GP différents, car chacun d'entre eux est une instance distincte de l'objet `GovPattern` en mémoire de la machine. Cela va créer des doublons indésirables dans l'ensemble. Pour le résoudre, nous avons intégré la méthode magique `__eq__` à la classe `GovPattern`, permettant d'identifier les objets `GovPattern` en basant sur leurs tableaux de GP. Parce que si les tableaux sont identiques, alors leurs noms de GP sont sûrement identiques, donc, les objets `GovPattern` sont identiques ; mais l'inverse n'est pas vrai. Par exemple, initialement, le nom de GP de *devoir* est `N_V_Vinf` même que celui de *vouloir*, tandis que leurs tableaux sont complètement différents.
2. Il est possible que des noms de GP identiques correspondent à des tableaux différents. Soit la différence est grande, comme dans les tableaux de *devoir* et *vouloir* ; soit la seule différence se trouve dans l'étiquette de relation, en raison d'annotations erronées ou des normes différents de certains corpus moins connus. Malheureusement, cette différence

ne manifeste pas dans le nom de GP. Pour éviter la confusion, nous avons créé une fonction qui permet de modifier le nom de GP en lui attachant un numéro de série.

Nom	Tableau	Pool	Action
Vrai	Vrai	Existant	Renvoyer celui existant
Vrai	Faux	Inexistant	Attacher un numéro de série et renvoyer l'objet `GovPattern` modifié
Faux	Vrai	s/o	s/o
Faux	Faux	Inexistant	Ajouter dans le *pool* et renvoyer tel quel

TABLEAU 2.2 – Matrice décisionnelle pour le traitement des objets `GovPattern` selon la présence de leur nom et tableau dans le *pool*

Pour améliorer le système, il serait judicieux de créer une classe `Pool` contenant un *pool* d'objets `GovPattern` comme attribut, chacun associé à un patron de régime absolument distinct. Ce *pool* contrôle l'objet `GovPattern` que l'ensemble d'objets `GovPattern` va recevoir. Son principe de fonctionnement est de vérifier chaque nouvel objet `GovPattern` nouvellement renvoyé par rapport aux éléments déjà présents dans le *pool* de la classe `Pool`. Cette vérification se fait en deux étapes : d'abord le nom de GP, puis le tableau de GP, donnant lieu à quatre circonstances possibles montrées dans le tableau 2.2. La valeur (Vrai ou Faux) indique si le nom ou le tableau de GP existe déjà dans le *pool*.

Suite à cette approche, les objets de `GovPattern` sont soigneusement examinés et ajustés avant de s'ajouter dans l'ensemble de l'objet `GrewResultsAnalyser` lié au verbe cible. L'approche permet de réduire énormément le travail de nettoyage manuel.

Élaboration des dictionnaires préliminaires

À cet égard, les ensembles associés aux verbes se convertissent ensuite en un dictionnaire général dont le nom de GP sert de clé et le tableau de GP devient sa valeur. Puis nous sauvegardons le dictionnaire général sous forme de JSON, intitulé `gpcon_eng_to_clean.json`, comme le montre la figure 2.10.

Nous enregistrons également les noms de GP utilisé par chaque verbe dans un dictionnaire dont le verbe fait office de clé et les noms de GP devient sa valeur. Puis, ce dictionnaire est aussi sauvegardé sous forme de JSON, intitulé `lexicon_eng_to_clean.json` présenté à la figure 2.11.

```
"N_V_N_Vger": {
    "I": {
        "rel": "subj",
        "dpos": "N"
    },
    "II": {
        "rel": "comp_obj",
        "dpos": "N"
    },
    "III": {
        "rel": "comp_obl",
        "dpos": "V",
        "finiteness": "GER"
    }
},
"N_V_N_Vinf": {
    "I": {
        "rel": "subj",
        "dpos": "N"
    },
    "II": {
        "rel": "comp_obj",
        "dpos": "N"
    },
    "III": {
        "rel": "comp_obl",
        "dpos": "V",
        "finiteness": "INF"
    }
}
```

FIGURE 2.10 – Extrait de `gpcon_eng_to_clean.json`

Les dictionnaires préliminaires sont sauvegardés au format JSON en raison de sa compatibilité avec Python et notamment sa lisibilité pour l'édition. L'édition manuelle consiste à corriger et compléter le nom de GP et le tableau de GP ; et il est également nécessaire de désambiguïser les patrons de régime liés aux différents lexèmes. Une fois le travail manuel terminé, nous renommons les dictionnaires en `gpcon_eng_cleaned` et `lexicon_eng_cleaned`.

```
"prefer": [
    "N_V_to_Vinf",
    "N_V_Vger"
],
"prepare": [
    "N_V_to_Vinf",
    "N_V_Vinf"
],
"press": [
    "N_V_N_to_Vinf_2"
]
```

FIGURE 2.11 – Extrait de `lexicon_eng_to_clean.json`

Élaboration des dictionnaires finaux

À l'issue du processus manuel, le programme recharge les dictionnaires ajustés et effectue deux dernières opérations :

— Marquer `rs` les noms de GP impliquant la montée. Cette étape, bien qu'optionnelle, apporte une précision supplémentaire, car sans cette marque, les noms de GP se distingueraient uniquement par leur numéro de série ajouté lors de l'étape suivante.

— Redonner ou récupérer le numéro de série. Cette étape est obligatoire, parce que le nettoyage manuel peut avoir créé des discontinuités dans la numérotation des noms de GP d'une même série. Par exemple, si le patron de régime `N_V_N_to_Vinf_2` est supprimé dans les dictionnaires, mais que son « frère » `N_V_N_to_Vinf_3` est toujours présent, cette étape permet de rétablir une numérotation cohérente.

Les dictionnaires finaux sont maintenant prêts à être construits. Le processus se déroule en deux phases, une pour chaque dictionnaire :

1. *lexicon* est composé de deux parties. La première partie s'appuie sur un dictionnaire intermédiaire (comme le bloc ci-dessous l'indique) facilement construit du fichier JSON `lexicon_eng_cleaned`.

```
{[gp_name1, gp_name2]: {verb1, verb2...}}
```

   Le premier verbe de l'ensemble est désigné pour représenter la classe qui utilise les GP collectés dans `[gp_name1, gp_name2]`. Un extrait de la première patrie de *lexicon* est présenté ci-dessous. Il faut noter que dans l'entrée `accepter: "accepter"`, le premier « accepter » est une lexie, alors que le second « "accepter" » représente une

classe.

```
accepter: "accepter"
choisir: "accepter"
...
```

La seconde partie décrit les noms de GP et leurs diathèses utilisés communément par les verbes de la catégorie. Chaque entrée suit en général une structure similaire. Un extrait de la seconde partie de *lexicon* est comme suit :

```
"accepter": verb {
    gp = { id=N_V_de_Vinf   dia=12}
}
```

Nous avons créé la fonction `convert_lexicon_str` qui permet d'automatiser cette construction, y compris la plupart de diathèses, car elles correspondent majoritairement au même nombre et au même ordre d'actants. Toutefois, certaines ont besoin de modification manuelle, notamment pour les patrons de régime impliquant des phrases explétives. En combinant deux parties, *lexicon* est achevé.

2. *gpcon* a une structure relativement simple. Chaque entrée représentant un patron de régime, est composé de deux parties.

```
//demander ordonner permettre promettre proposer
N_V_de_Vinf_à_N {
    I={rel=subj dpos=N}
    II={rel=comp_obj dpos=V finiteness=INF prep=de}
    III={rel=comp_obl dpos=N prep=à}
```

La première est un groupe des verbes utilisant le GP en question. Ce groupe est généré à partir d'un dictionnaire intermédiaire (comme illustré dans le bloc ci-dessous) construit des dictionnaires précédemment nettoyés. Par exemple, la paire clé-valeur du dictionnaire intermédiaire `N_V_de_Vinf_à_N: {demander, ordonner, permettre, promettre, proposer}` signifie que le patron de régime `N_V_de_Vinf_à_N` est employé par ces verbes.

```
    {gp_name1: {verb1, verb2...}, gp_name2: {verb3...}...}
```

La seconde n'est que le tableau de GP sous forme de chaîne de caractère, comme celui dans la figure 2.10. Nous avons créé la fonction `get_str` qui permet la formalisation. En combinant le deux parties, *gpcon* est ainsi réalisé.

Il est important de noter que le code utilisé est en fait un peu plus complexe que l'explication ci-dessus, car il intègre des mécanismes pour conserver les traces du travail manuel effectué et le rendre réutilisable, évitant de recommencer le travail manuel en cas de problèmes ou défauts éventuels. Ces détails d'implémentation ne sont toutefois pas intéressants à discuter ici.

2.4 Synthèse

Dans ce chapitre, nous nous sommes concentrés sur l'élaboration des dictionnaires adaptés à GenDR *gpcon* et *lexicon* pour les verbes à montée et à contrôle. La démarche s'est articulée autour de deux axes : l'extraction des phrases contenant les verbes en question et la construction des dictionnaires à partir des données extraites.

La première partie du travail a commencé par sélectionner le format d'annotation des corpus. Initialement, nous avons envisagé l'UD. Toutefois, ce dernier s'est révélé inadéquat en raison de son incompatibilité avec les étiquettes de GenDR et de son incapacité à distinguer les verbes à montée des verbes à contrôle. Nous avons finalement retenu le SUD qui utilise des étiquettes similaires à celles dans GenDR et permet une possible distinction entre les deux types de verbes. Par ailleurs, nous avons choisi les corpus compatibles avec le SUD, listés en 2.1.

En ce qui concerne l'extraction des données, nous avons adopté une approche hybride, utilisant une liste prédéfinie de verbes à montée et à contrôle et un outil de recherche Grew. Nous avons créé pour chaque verbe une requête selon la grammaire de Grew. Grew a extrait les phrases pertinentes des corpus en fonction de la requête et exporté les résultats dans un tableau au format `.csv`.

La construction des dictionnaires, quant à elle, s'est appuyée sur un ensemble de classes Python développées. La classe `GrewResultsAnalyser` prend en entrée les données extraites et les décompose en phrases individuelles. `GovPattern` construit le patron de régime du verbe dans la phrase. La classe `ComparableToken` assure l'ordre syntaxique des tokens dans la phrase. Le processus de construction des patrons de régime suit une séquence rigoureuse : filtrage des dépendants pertinents, restauration des prépositions si nécessaire, traitement des cas particuliers comme les phrases à sujet nul, mise en ordre des dépendants sélectionnés et uniformisation des attributs du dépendant. Ensuite, nous avons converti les patrons construits au format `JSON` pour créer des dictionnaires préliminaires. Ces derniers ont fait l'objet d'un travail de nettoyage et de correction manuelle qui inclut crucialement la

désambiguïsation des patrons liés aux différents lexèmes. Puis, nous avons marqué les patrons de montée avec `rs` et renuméroté les séries de patrons. Finalement, pour construire le `lexicon`, nous avons donné à chaque GP une diathèse et catégorisé les verbes en fonction de leurs GP. Pour le `gpcon`, nous avons détaillé chaque GP et regroupé les verbes qui l'emploie.

Dans le prochain chapitre, nous présenterons l'implémentation des verbes à montée et à contrôle au sein de GenDR avec des nouveaux modules.

Chapitre 3

Traitement des verbes à montée et à contrôle

Ce chapitre se consacre à l'implémentation des verbes à contrôle et à montée au sein de GenDR, à l'aide de deux nouveaux modules. D'abord, nous expliquerons comment construire la RSém du verbe à contrôle et éliminer le sujet du verbe subordonné à l'infinitif. Ensuite, nous présenterons un nouveau module spécifique pour les verbes à montée, en détaillant son emplacement dans GenDR et ses règles.

3.1 Verbes à contrôle

Comme expliqué en §1.4, les verbes à contrôle et à montée se distinguent syntaxiquement et présentent des différences dans le traitement au sein de GenDR. Les verbes à contrôle, plus simples à implémenter, n'impliquent pas de déplacement de nœud, contrairement aux verbes à montée (hormis les déplacements syntaxiques communs à tous les verbes).

Dans une construction avec verbe à contrôle, le mode infinitif du verbe subordonné prive la tête de la proposition enchâssée de sa capacité d'assigner le cas nominatif au sujet (Chomsky, 1980; Rouveret et Vergnaud, 1980). Cela se traduit par l'absence apparente d'un sujet audible pour le verbe subordonné dans la phrase de surface. La syntaxe de dépendances affirme plus catégoriquement que seulement le verbe fini possède un sujet, et l'expression explicite du sujet est obligatoire avec un verbe fini dans les RSyntS. Dans la RSém servant de graphe d'entrée, on utilise alors soit un PRO de coréférence, soit un nœud de coréférence comme le premier actant du verbe subordonné pour représenter correctement la construction avec un verbe à contrôle et notamment pour faciliter l'application des règles syntaxiques lors

de la première transition. Aux cours des essais, la méthode utilisant le nœud de coréférence s'est avérée plus pratique, et c'est celle qui a été retenue finalement.

La figure 3.1 représente la transition de la RSém à la RSyntP pour la phrase *Paul accepte de manger l'orange.* Selon la coréférence expliquée en §1.4.3, le sujet de la proposition infinitive coréfère à son antécédent qui est sujet de la phrase principale. Ainsi, la RSém inclut un second 'Paul' comme sujet de l'infinitive, lié au sujet de la principale par l'étiquette `coref`.

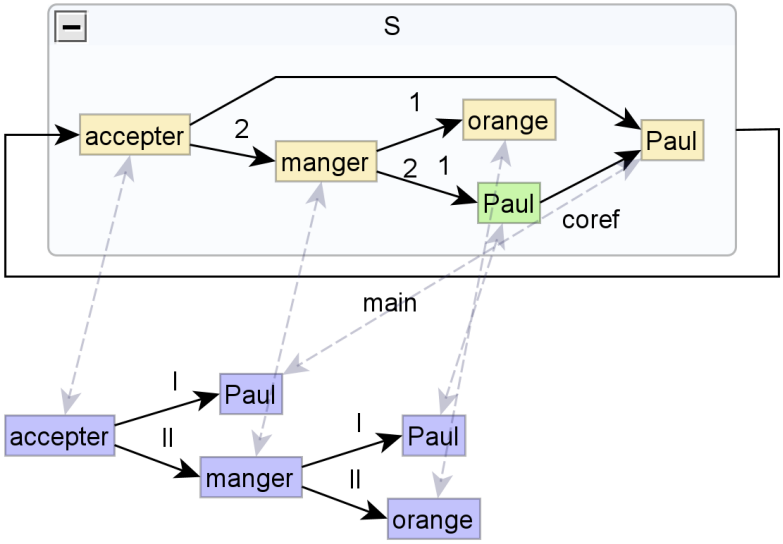


FIGURE 3.1 – RSém-RSyntP pour le verbe ACCEPTER

Dans cette même figure 3.1, le système sélectionne pour le verbe ACCEPTER le patron de régime `N_V_de_Vinf`. Ce patron assigne au verbe subordonné l'attribut `finiteness=INF` indiquant le mode infinitif. Grâce à cet attribut, le verbe subordonné MANGER va perdre son sujet redondant PAUL dans la RSyntS, comme l'illustre la figure 3.2. Toutefois, bien que détaché de la structure principale, le nœud PAUL qu'il faut disparaître reste dans le graphe. Cela soulève une question importante : comment peut-on éliminer définitivement les nœuds détachés de la structure principale ?

Pour le résoudre, nous avons créé un petit module utilitaire `ssynt-ssynt-treecheck` inspiré des travaux de Kahane et Lareau (2005), illustré par le listing 3.1. Celui-ci permet de parcourir la structure arborescente à partir du nœud racine et dériver une nouvelle structure connexe au même niveau de représentation. Il procède de manière récursive, passant du nœud racine à ses nœuds fils, puis les nœuds fils de ces derniers, ainsi de suite. Comme le module ne parcourt que les nœuds connectés au sein de l'arborescente, ceux détachés sont automatiquement exclus du traitement. Par conséquent, les nœuds détachés ne figurent plus dans la

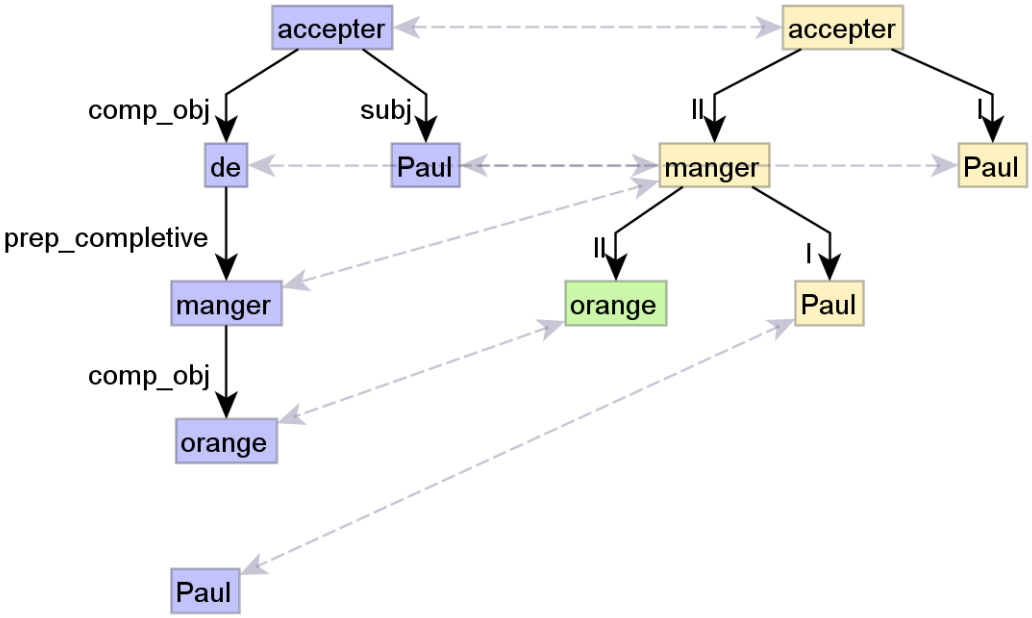


FIGURE 3.2 – RSyntP (droit) -> RSyntS (gauche) pour le verbe accepter

structure finale, aboutissant à leur suppression définitive.

Pour illustrer l'utilisation de ce module, reprenons l'exemple précédent illustré par la figure 3.2. On applique le module `treecheck` à la RSyntS pour éliminer PAUL détaché, comme la figure 3.3 l'illustre. Nous obtenons finalement une bonne RSyntS connexe, comme la figure 3.4 l'illustre.

Cette approche présente l'avantage d'être conforme aux principes syntaxiques. Dans la RSyntP, un sujet coréféré occupe la position sujet du verbe subordonné, satisfaisant ainsi le critère thématique et le patron de régime par rapport au verbe. Ce sujet implicite est ensuite éliminé lors du passage à la RSyntS finale, ce qui aboutit à la structure de phrase que nous

Listing 3.1 – Module `treecheck`

```
left
?Xl{
  l:?r-> l:?Yl
}
right
rc:?Xr {rc:<=>?Xl
  ?r-> ?Yr{<=>?Yl
  }
}
```

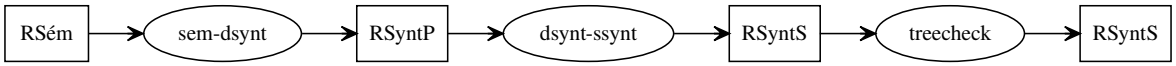


FIGURE 3.3 – Processus de GenDR avec le module `treecheck`

utilisons couramment dans le langage quotidien. Si on laisse vide la position du premier actant du verbe subordonné, le processus s'arrêtera après la sélection du patron de régime pour le verbe subordonné. En effet, 'manger' qui est bi-actanciel ne régit en ce cas que son seconde actant, tandis que MANGER sélectionne son patron de régime qui inclut crucialement `dia=12`. Cette divergence met en échec au déclenchement de la règle `actant_gp_ij` qui exige une correspondance exacte du nombre d'actants. Par conséquent, la création des nœuds subordonnés de MANGER ne peut aboutir.

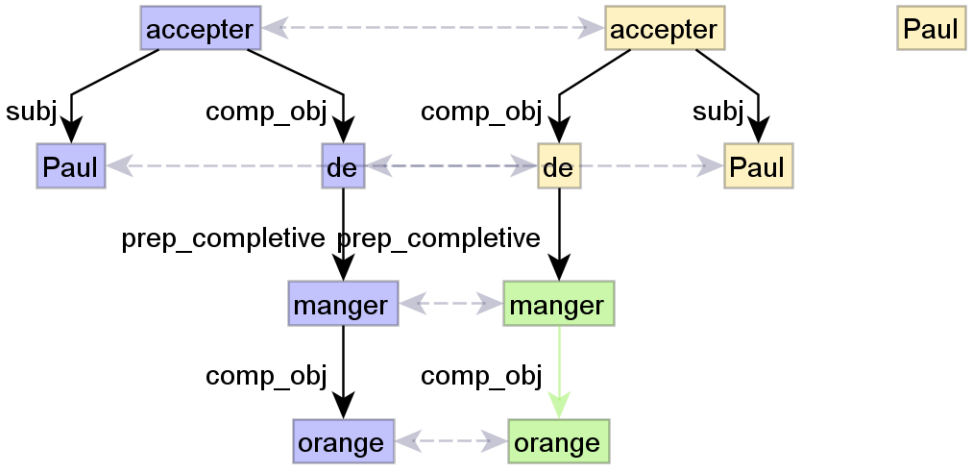


FIGURE 3.4 – RSyntS originale (droite) -> RSyntS connexe (gauche) pour le verbe *accepter*

3.2 Verbe à montée

3.2.1 Moment de l'application

Comme expliqué en §1.4, le sujet du verbe à montée se déplace à la position sujet de la principale à partir d'une position enchâssée. Pour atteindre notre objectif, la nouvelle version de GenDR doit être capable de reproduire un déplacement similaire. En plus des actuels modules, un module dédié à la modélisation du phénomène de montée est alors nécessaire.

Parallèlement, une question importante se pose : à quel moment ce nouveau module doit-il s'appliquer dans la séquence des opérations de GenDR ? Nous avons expérimenté trois possibilités distinctes, comme le montre la figure 3.5 :

1. Moment 1 (M1) : application du nouveau module entre RSém et RSyntP.
2. Moment 2 (M2) : application du nouveau module entre RSyntP et RSyntS.
3. Moment 3 (M3) : application du nouveau module après la RSyntS.

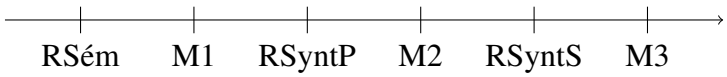


FIGURE 3.5 – Moments d'application possibles du nouveau module dans l'actuel modèle

Moment 1

Le moment 1 permet d'effectuer le déplacement avant la RSyntP, donc avant l'arborisation et la lexicalisation profonde, produisant une RSém modifiée. Pourtant, cette approche entraîne des difficultés sur la création de nœuds lors de l'arborisation profonde, car la montée a changé le nombre d'arguments du verbe subordonné et du verbe principal. Par conséquent, ces prédicats avec leur nouvelle structure argumentale ne correspondent plus au patron de régime défini dans le *gpcon* qu'ils sont censés utiliser.

(1) *Paul semble dormir.*

Prenons l'exemple (1) illustré par la figure 3.6. Si on approuve le moment 1 pour la montée du nœud, 'Paul' se déplace vers 'sembler' et 'dormir' perd son sujet 'Paul'. En conséquence,

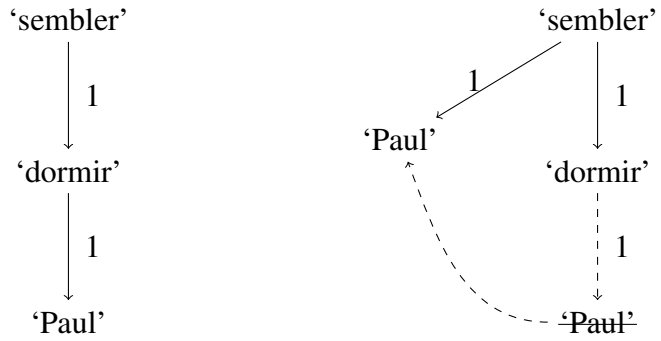


FIGURE 3.6 – RSém et RSém après la montée

le verbe principal ‘sembler’ prend désormais deux arguments, tandis que ‘dormir’ n’en a plus aucun.

Lors de l’arborisation, le système sélectionne pour le verbe SEMBLER le patron de régime `N_Vrs_Vinf` et en même temps lui assigne l’attribut `dia=dia1` dont 1 correspond à la diathèse du GP sélectionné. Cependant, le processus se termine lors de l’application de la règle `actant_gp_i`, car l’une des conditions `?Xr.dia=dia+?i+?j` n’est pas remplie. Le membre gauche `?Xr.dia` est égal à `dia1` (attribut de SEMBLER), tandis que le membre droite `dia+?i+?j` est égal à `dia12`, étant donné deux arguments gouvernés par ‘sembler’ qui est mono-actanciel.

Nous pouvons constater la même erreur sur ‘dormir’ qui est un verbe mono-actanciel, alors qu’il n’a plus d’actants après la montée. La règle syntaxique permettant de créer le nœud ne peut s’y appliquer correctement pour la même raison que ‘sembler’. Donc, la montée anticipée perturbe la création du nœud et entraîne l’échec de la génération de la RSyntP.

(2) *Il semble pleuvoir.*

En outre, l’inadéquation du moment 1 pour la montée se manifeste de manière plus évidente, face à la phrase (2). Comme le sujet explétif *il* ne se présente pas dans la RSém, aucun nœud ne va se déplacer vers le niveau supérieur, ce qui provoque un blocage dans le processus de génération. À l’appui de l’exclusion du premier moment, nous présenterons un argument supplémentaire sous l’angle des fonctions lexicales en 3.2.2. Compte tenu les problèmes rencontrés dans ces deux exemples, on trouve que le premier moment n’est pas approprié.

Moment 2

Le deuxième moment proposé vise à réaliser la montée du nœud après l’arborisation et la lexicalisation profondes, mais avant celles de surface, produisant une RSyntP modifiée. Cependant, il rencontre le problème similaire à celui du premier moment. Lors de l’arborisation de surface, les règles syntaxiques ne peuvent que créer les nœuds pour les actants correspondant à ceux définis dans le *gpcon*. En conséquence, l’actant monté ne sera pas pris en compte, puisque le verbe principal ne le gouverne initialement pas. Cet actant n’apparaît pas dans la RSyntS produite.

Par exemple, dans la figure 3.7, la règle de surface `syn_actant_dir` en 3.2 ne traite pas l’actant monté PAUL. En effet, cette règle sert à convertir une relation profonde ($?Xl$–**?r**→$?Yl$) en celle de surface ($?Xr$–**?rel**→$?Yr$). Concrètement, elle permet de créer un nœud ($?Yr$) représentant le $r$ème actant régi par son gouverneur ($?Xr$) et d’établir la relation ($?D.rel$) entre

Listing 3.2 – Règle `synt_actant_dir`

```
left
?Xl{
  l:?r-> ?Yl{}
}
?D <- gpcon::(?Xl.gpid).(?r)
right
rc:?Xr{
  rc:<=> ?Xl
  ?D.rel-> ?Yr{ <=> ?Yl
  }
}
conditions
not ?r=ATTR;
not ?r=COORD;
not ?r=APPEND;
?Xl.gpid;
not (?D.rel=subjective or ?D.rel=subj); // Subj done by other rules
not ?D.prep;
not ?D.conj;
?Xr.ssynt=OK;  // Wait for full lexicalization
```

eux définie déjà dans le GP de ($?Xl$) dans *gpcon*. En ce cas, PAUL qui est monté à la position de l'actant de SEMBLER ne correspond pas à l'actant attendu selon le patron de régime de SEMBLER ($?Xl$), qui doit être un verbe. L'index ($?r$) ne peut alors être déterminé et, PAUL n'apparaît pas dans la RSyntS produite. La transformation de la RSyntP après la montée (fig. 3.7) en RSyntS ne peut donc pas aboutir. Ainsi, le deuxième moment s'avère également inadéquat.

Moment 3

Le troisième moment est devenu la dernière option, à l'exclusion des moments précédents inadéquats. Selon l'axe (fig. 3.5), l'application du nouveau module au Moment 3 ne perturbe certainement pas les transitions RSém→ RSyntP et RSyntP→ RSyntS. Aussi ce moment doit-il permettre l'application en question sans préjudice du reste des étapes de la TST, telles que la transformation en Représentation morphologique profonde. Toutefois, puisque GenDR ne les modélise pas encore dans l'actuelle version basée sur le *gpcon*, nous ne pouvons pas effectuer de tests. En tout cas, le Moment 3 a pour l'instant le moindre impact sur l'actuelle

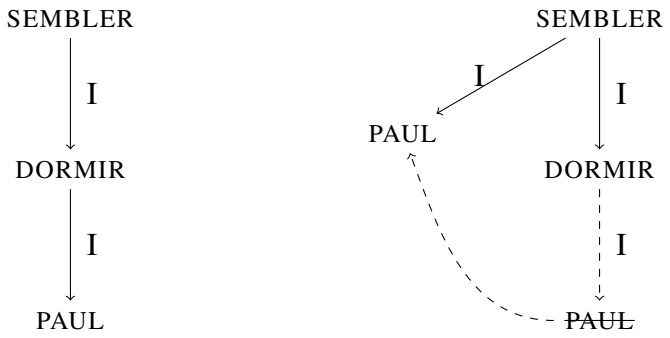


FIGURE 3.7 – RSyntP et RSyntP après la montée

version de GenDR. Par ailleurs, après le Moment 3, on obtient la RSyntS tout en respectant la syntaxe de dépendance. Donc, ce moment ne perturbera pas la modélisation des prochaines étapes.

En résumé, à la lumière des problèmes rencontrés dans les deux moments précédents, il est plus approprié de réaliser la montée au Moment 3. Cela signifie que le nouveau module intervient, après que les règles syntaxiques aient terminé la construction de la structure arborescente de surface. Le nouvel organigramme de programmation sera présenté dans la figure 3.14.

3.2.2 Règles

Les verbes à montée sont souvent flexibles. Il est grammatical pour eux de prendre un sujet explétif et une proposition. Notre nouvelle version de GenDR doit aussi répondre au besoin de produire des paraphrases.

Pourtant, la réalisation nécessiterait des modifications dans l'actuel module `sem-dsynt`. Dans la section 1.3.3, nous avons montré l'arborisation profonde dans GenDR. La sélection des patrons de régime peut apporter des paraphrases syntaxiques. Ceci est aussi valable pour les verbes à montée. Par exemple, il y a souvent deux patrons de régime typiques pour des verbes à montée : `N_Vrs_Vinf` et `Exp_V_que_Vsub`. Le second GP est plus compliqué que le premier en termes d'arborisation. Le second implique la création du nœud de IL explétif qui n'existe pas dans la structure d'entrée, RSém, parce que le sujet explétif est sémantiquement vide. Malheureusement, les actuelles règles syntaxiques du module `Sem-Dsynt` ne permettent pas de créer un nœud sans correspondance dans la structure arborescente. Nous avons alors ajouté une nouvelle règle dans le groupe de règles syntaxiques. Elle ressemble à la règle `actant_gp_i`, mais crée particulièrement un nœud vide sous le verbe principal puis

Listing 3.3 – Règle `actant_gp_rs_i`

```
left
?Xl{
  l:?i-> ?Yl{
  }
}
right
rc:?Xr{
  rc:<=> ?Xl
  dsynt=OK
  I-> ?Wr {
    dsynt=created
    dlex=gpcon::(?Xr.gpid).(I).(dlex)
  }
  II-> ?Yr{ <=> ?Yl
    dsynt=created
  }
}
conditions
?Xr.dsynt=gp_selected;
gpcon::(?Xr.gpid).(I).(dlex);
?Xr.dia=dia+X+?i;
```

lexicalisé suivant le `dlex` inscrit dans l'actant **I** du GP, ce qui représente le sujet explétif. La nouvelle règle est renommée `actant_gp_rs_i` (voir le listing 3.3).

Il est également possible de considérer une phrase fondée sur le patron de régime particulier `Exp_V_que_Vind_à_N`. Par exemple, *Il me semble que Paul parle français.* Dans cette structure, en plus du sujet explétif et de la proposition, le verbe principal SEMBLER gouverne un pronom datif MOI. Pourtant, ni la nouvelle règle `actant_gp_rs_i` ni les autres règles syntaxiques existante ne sont capables de gérer cette construction. Par conséquent, il est nécessaire de faire une autre règle permettant la création du nœud pour accueillir le pronom datif.

Cette nouvelle règle s'inspire largement de la règle `actant_gp_rs_i`, mais elle se distingue par l'ajout d'un troisième actant dans la « fenêtre droite », représentant le pronom datif ajouté . Nous proposons de nommer cette règle `actant_gp_rs_ij`, dont **j** symbolise le troisième actant ajouté.

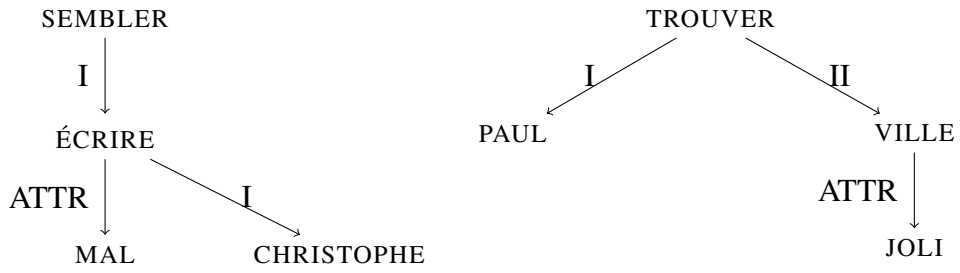


FIGURE 3.8 – Distinction de gouvernance du sujet dans les RSyntP entre *sembler* et *trouver*

Fonctions lexicales

Dans certains cas, la montée demande l'aide d'une fonction lexicale. Il est possible qu'un verbe gouverne directement à la fois un nom et un adjectif dans une phrase. Le nom et l'adjectif s'interprètent comme une proposition, appelée proposition réduite qui fait partie de la construction *small clause* proposée par Williams (1975). L'exemple du verbe *trouver* en (3-a) montre bien cette construction. Comme expliqué en §1.4.3, la phrase (3-a) véhicule le même sens que sa forme longue en (3-b) et la forme réduite dérive crucialement de sa forme longue.

(3) a. *Paul trouve la ville jolie.* =(9)
b. *Paul trouve que la ville est jolie.*

Cependant, les verbes tels que *trouver*, *rendre*, ne sont pas considérés comme des verbes à montée au sens strict, parce que leur sujet de surface est initialement dans cette position, au lieu de monter à partir d'une position enchâssée. De plus, ces verbes peuvent gouverner leur propre sujet, une capacité que les verbes à montée typiques, comme *sembler*, ne possèdent pas. La figure 3.8 illustre cette distinction.

Néanmoins, il serait avantageux de traiter dans le cadre de GenDR les verbes prenant la proposition réduite de manière similaire aux verbes à montée pour deux raisons :

1. Similarité syntaxique : Ces verbes prenant la proposition réduite permettent de monter le sujet de la proposition dans la position de leur objet. Ainsi, il s'agit aussi d'une forme de montée, mais vers une destination différente pour le nœud déplacé, comme expliqué en §1.4.3.
2. Cohérence du module : Cette approche permet d'utiliser un module unique pour modéliser à la fois les verbes à montée et les verbes prenant la proposition réduite, simplifiant ainsi l'architecture du système.

Les verbes à montée et les verbes prenant la proposition réduite se distinguent par la

position d'arrivée de leur argument déplacé. Pour gérer cette différence, nous pouvons ajuster la position d'arrivé de l'argument dans le patron de régime du verbe en modifiant la valeur de l'attribut `raise_to`. On détaillera ce changement dans la sous-section suivante.

Afin de modéliser une phrase en forme longue comme en (3-b), l'utilisation de la fonction lexicale est indispensable. Le verbe support *être*, sémantiquement vide, doit être généré par la fonction lexicale $\texttt{Oper}_1$ lors de la transition entre la RSém et la RSyntP. Dans la RSyntP illustrée par la figure 3.9, $\texttt{Oper}_1$ s'applique à JOLI et renvoie ÊTRE. Pour la modélisation d'une phrase en forme réduite comme en (3-a), une question se pose également : faut-il aussi recourir à la fonction lexicale lors de la première transition ?

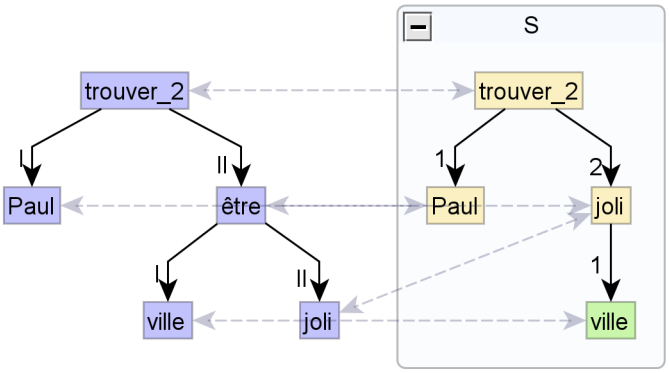


FIGURE 3.9 – $\texttt{Oper}_1$ appliquée à 'joli' ; RSyntP à gauche, RSém à droite

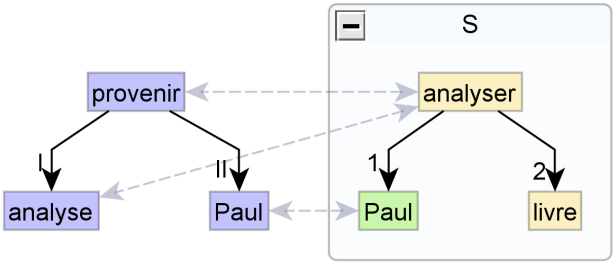


FIGURE 3.10 – $\texttt{Func}_1$ appliquée à 'analyser' ; RSyntP à gauche, RSém à droite

Avant de répondre à la question, nous avons d'abord convenu d'intégrer la fonction lexicale dans la nouvelle version de GenDR. Nous avons alors créé la règle `Vsupport_i` dans le module `sem-dsynt`. Elle permet de traiter chaque fois un prédicat comme une base (L) en insérant le verbe support liant la base avec son argument. La règle est compatible avec les fonctions lexicales $\texttt{Func}_i$ ou $\texttt{Oper}_i$. Leur indice peut prendre la valeur 1, 2 ou 3. Un exemple de la $\texttt{Func}_1$ est illustré en 3.10.

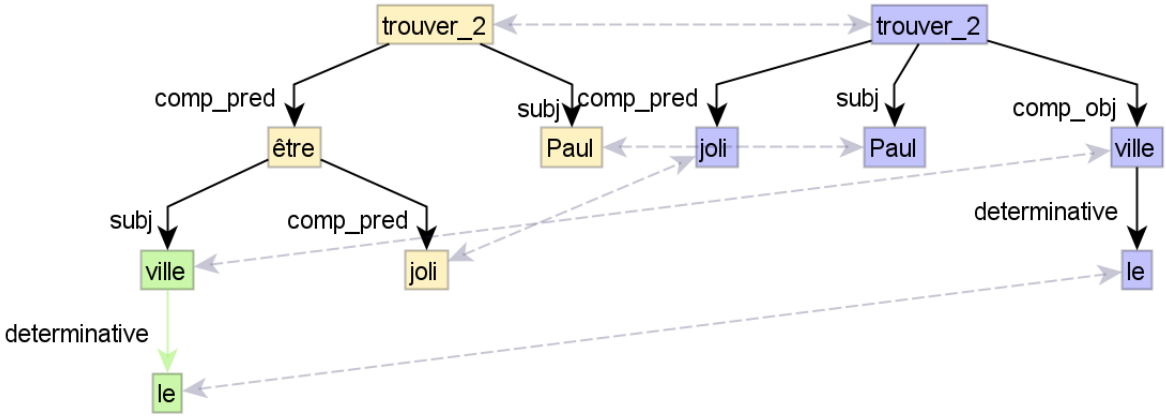


FIGURE 3.11 – La montée du modifié VILLE

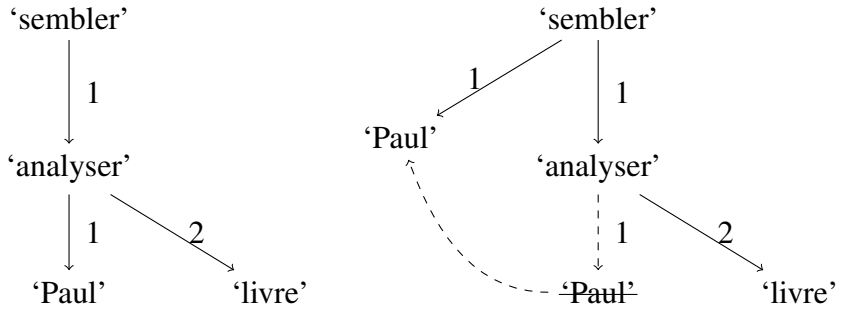


FIGURE 3.12 – RSém et RSém (après la montée).

Revenons maintenant à la question précédente. La réponse est affirmative. Puisque les phrases en (3) transmettent le même sens, elles dérivent d'une même RSém. On suppose que la transformation de la version réduite en (3-a) se base sur la version longue en (3-b). Cela implique que la version réduite doit passer également par une SSyntP qui contient le verbe support ÊTRE, structurellement identique à la SSyntP de la version longue, comme illustrée dans la figure 3.9. Ensuite, à partir de cette SSyntP, plusieurs opérations se produisent pour obtenir la RSyntS finale de la version réduite :

1. Le sujet de l'enchâssée VILLE monte dans la position objet du verbe principal TROUVER ;
2. Le verbe support ÊTRE est supprimé par le filtrage ;
3. L'adjectif JOLI se détache de ÊTRE et se rattache directement à TROUVER.

La transformation est illustrée dans la figure 3.11. Ce processus se fait dans la phase du nouveau module que nous allons présenter en détail dans les sous-sections suivantes. Avant

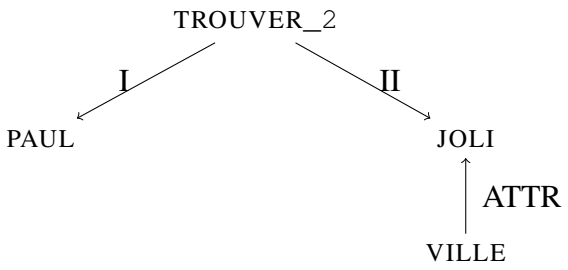


FIGURE 3.13 – Fausse RSyntP, si la $\texttt{Oper}_1$ n'était pas appliquée.

de continuer, nous avons mentionné en §3.2.1 que l'intégration des fonctions lexicales fournirait un autre argument en faveur de l'exclusion du Moment 1. Par exemple, pour l'énoncé *Paul semble analyser un livre.*, si la montée s'effectue au premier moment, nous avons une nouvelle RSém (après la montée) illustrée par la figure 3.12. Mais cette montée prive ANALYSER de la possibilité de se transformer en *faire l'analyse* à travers la $\texttt{Oper}_1$, car le premier actant 'Paul' du prédicat 'analyser' ('analyse') est monté et s'est rattaché à 'sembler'. Nous perdons alors une paraphrase.

Cette approche présente des avantages significatifs :

1. Unification des verbes à montée et des verbes à proposition réduite : Cette approche permet de traiter de manière uniforme les verbes à montée classiques et les verbes prenant une proposition réduite. Cette unification simplifie le module et manifeste sa flexibilité dans le traitement des structures syntaxiques complexes.
2. Cohérence avec l'analyse syntaxique : La SSyntP, qui inclut le verbe support pour former une proposition avec l'adjectif et son modifié, correspond étroitement à l'analyse syntaxique de la proposition réduite.
3. Transition plus naturelle : Elle rend la transition entre la RSyntP et la RSyntS plus fluide et naturelle. Sans cette méthode, comme le montre la figure 3.13, la RSyntP se trouverait dans une impasse du à due à la position inhabituelle des nœuds JOLI et VILLE. Cette configuration aurait rendu difficile une transition légitime vers la RSyntS.

Il est à noter que cette implémentation des fonctions lexicales n'est pas complète. Elle n'a pour objectif que d'augmenter la flexibilité dans le traitement des verbes à montée et des verbes au comportement syntaxique similaire. Une implémentation complète des fonctions lexicales (Lambrey et Lareau, 2015; Lambrey, 2016) dans la version de GenDR basée sur le *gpcon* nécessiterait des recherches plus approfondies à l'avenir.

Nouveau module `ssynt-ssynt`

Au début de §3.2, nous avons déterminé que c'est après la RSyntS que le nouveau module serait appliqué, comme la figure 3.14 l'illustre. L'objectif principal de ce module est de monter un nœud ou un groupe de nœuds enchâssé vers le niveau supérieur, tout en effectuant possiblement d'autres opérations auxiliaires.

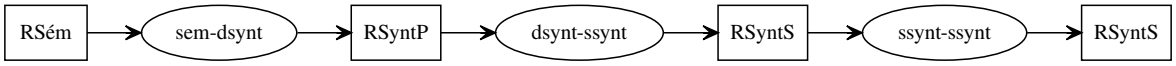


FIGURE 3.14 – Processus de GenDR avec le module `ssynt-ssynt`

D'abord, le module recopie les nœuds de la structure d'entrée comme base de la nouvelle structure. Les arcs indiquant les nouvelles dépendances des nœuds, sous-entendent leur déplacement. C'est ceux que nous cherchons à établir et justifier.

Pour la plupart des situations, la montée est la seule tâche à effectuer. Avant opérer le déplacement, il faut identifier les nœuds concernés : la **victime**, nœud perdant son dépendant et le **récepteur**, nœud auquel s'adjoint le nœud déplacé. La victime et le récepteur doivent alors se distinguer des autres par attribut spécial pour faire savoir la règle de montée. La victime se dote de l'attribut `raise_from` et le récepteur a `raise_to`. Alors, comment ces attributs leur sont-ils assignés ? Comme mentionné en §1.3.3, les nœuds actants sont contraints par les attributs inscrits dans le patron de régime du gouverneur dans le *gpcon* et par les grammèmes durant la première transition. Toutefois, la contrainte de la première transition ne convient pas à l'attribut spécial. Premièrement, la victime est souvent un verbe ordinaire dont le patron de régime ne comporte pas l'attribut `raise_from`. Deuxièmement, la victime n'est pas « victime » perdant son dépendant en toutes circonstances, mais seulement s'il est gouverné par un verbe à montée. C'est la raison pour laquelle la contrainte en question n'est pas idéale pour assigner les attributs spéciaux. Alors, d'où proviennent-ils ? En effet, les attributs spéciaux sont définis dans le GP du verbe à montée. En tant que gouverneur, il les assigne à son dépendant, à travers des règles syntaxiques du module `dsynt-synt`, lors de la deuxième transition (RSyntP→ RSyntS).

Pour le faire, nous avons étendu les capacités de ces règles au-delà de leurs fonctions initiales. Désormais, elles peuvent attribuer respectivement à la victime et au récepteur les deux attributs spéciaux mentionnés, déjà définis dans le patron de régime du verbe à montée. Pour illustrer ceci, le listing 3.4 présente cette attribution (soulignée avec la ligne ondulée) effectuée par l'une de ces règles modifiées `synt_actant_dir` qu'on a précédemment présentée.

Listing 3.4 – Règle `synt_actant_dir`

```
left
?Xl{
  l:?r-> ?Yl{}
}
?D <- gpcon::(?Xl.gpid).(?r)
right
rc:?Xr{
  rc:<=> ?Xl
  raise_to=?D.raise_to
  ?D.rel-> ?Yr{ <=> ?Yl
    raise_from=?D.raise_from
  }
}
conditions
not ?r=ATTR;
not ?r=COORD;
not ?r=APPEND;
?Xl.gpid;
not (?D.rel=subjective or ?D.rel=subj); // Subj done by other rules
not ?D.prep;
not ?D.conj;
?Xr.ssynt=OK;  // Wait for full lexicalization
```

Elle assigne `raise_to` à $?Xr$ qui représente le verbe à montée et `raise_from` à $?Yr$ qui représente le verbe subordonné. La valeur de `raise_to` signifie la relation entre la victime et l'actant à monter ; celle de `raise_from` indique la relation qui lie le récepteur et l'actant monté.

Prenons un exemple simple comme en (4). Nous nous concentrons directement sur sa RSyntS. Dans cette RSyntS illustrée par la figure 3.15, DORMIR est victime portant l'attribut `raise_from=subj` et SEMBLER_6 est récepteur avec `raise_to=subj`. Donc, on s'attend à ce que le sujet PAUL se détache de la victime DORMIR et le récepteur SEMBLER l'accueillie et établit avec lui la relation **subj** dans la transition suivante.

(4) *Paul semble dormir.*

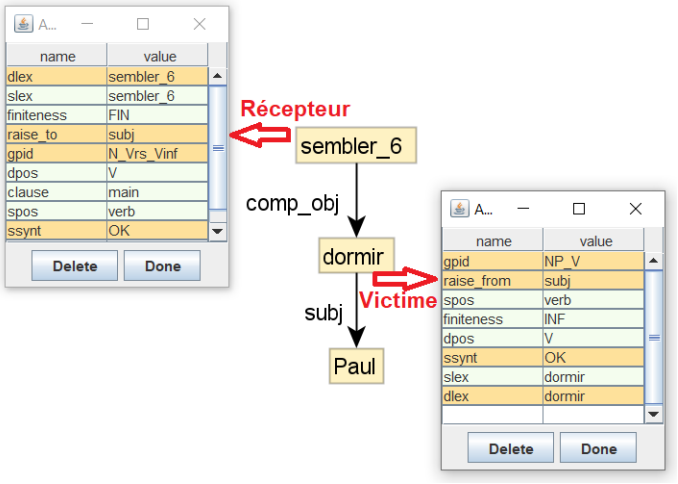


FIGURE 3.15 – La victime et le récepteur portant les attributs spéciaux.

Ainsi, les nœuds concernés portent les attribués spéciaux, prêts à passer au nouveau module. Dans ce dernier, le système commence par recopier les nœuds de la structure d'entrée avec la règle `node`, à condition que le nœud portant `raise_from` ne soit pas verbe copule. La condition permet d'exclure le verbe support redondant généré par la fonction lexicale. Par exemple, ÊTRE éliminé pendant la transition illustrée par la figure 3.11. En tant que nœud intermédiaire n'ayant pas de place dans le patron de régime `N_V_ADJ_N`, ÊTRE doit être supprimé dans la structure finale. Cette manière est plus radicale que le module `treecheck` pour supprimer un nœud, en raison du fait d'éviter sa création dans la nouvelle structure.

Une fois les nœuds recopiés, le module peut dessiner les arcs à l'aide des règles syntaxiques. Le module en comporte pour l'instant quatre :

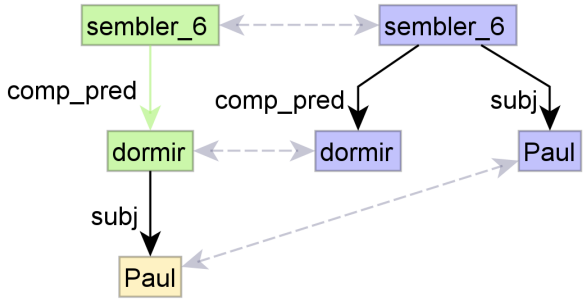


FIGURE 3.16 – Application du nouveau module à la RSyntS 3.15.

— `synt_dir` est la plus utilisée. Elle permet de lier le gouverneur et son dépendant avec la même relation que celle originale dans la RSyntS d'entrée, à moins que le gouverneur porte `raise_from`, chargé par la règle `synt_raising`. La règle est aussi « atomique » : chaque application ne peut établir qu'un arc. Dans l'énoncé précédent (4), la relation de gouvernance entre SEMBLER et DORMIR est établie par cette règle, comme l'illustre la figure 3.16.

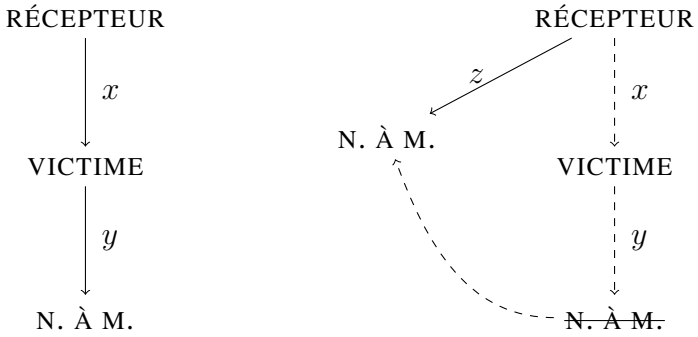


FIGURE 3.17 – Montée générale réalisée par `synt_raising`. "nœud à montée" (N. À M.)

— `synt_raising` est le noyau du module. Elle permet de déplacer le nœud à montée de la victime vers le récepteur. Elle cible donc une structure hiérarchique comportant ces trois nœuds disposés sur trois niveaux : récepteur-victime-nœud à montée (voir la figure 3.17). L'application de la règle est conditionnée par deux facteurs : le récepteur doit posséder l'attribut `raise_to`, et la victime doit avoir l'attribut `raise_from` dont la valeur est égale à la relation entre la victime et le nœud à montée. Lorsque ces conditions sont remplies, la règle permet de déplacer le nœud régi par la victime vers récepteur. La relation entre le nœud déplacé et le récepteur est étiquetée avec la

valeur de `raise_to`. Encore dans l'exemple 3.16, SEMBLER-DORMIR-PAUL correspondent à la structure hiérarchique mentionnée ; SEMBLER et PAUL possèdent respectivement `raise_to=subj` et `raise_from=subj`. Alors, comme les conditions sont satisfaites, PAUL monte et se rattache à SEMBLER avec la relation **subj** (la valeur de `raise_to=subj`).

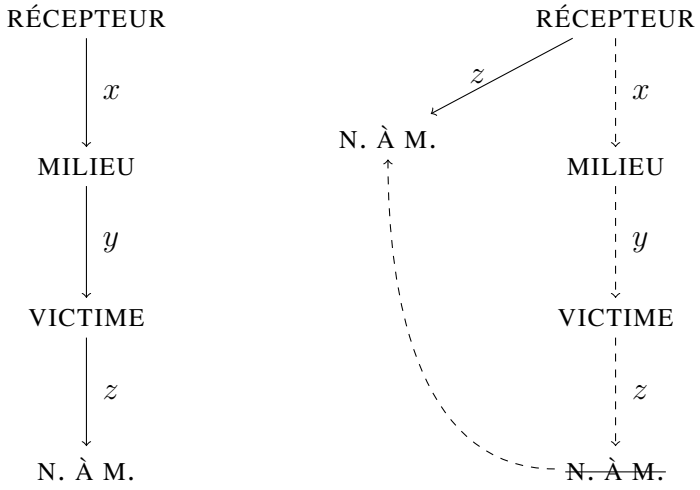


FIGURE 3.18 – Montée réalisée par `synt_raising_with_mid`. "nœud à montée" (N. À M.)

— `synt_raising_with_mid` est une version légèrement plus complexe de la règle `synt_raising`. Elle permet de déplacer le nœud à montée de la victime vers le récepteur, dépassant le nœud du milieu entre eux. Le nœud du milieu peut prendre une préposition. Donc, une structure hiérarchique à quatre niveaux est la cible de cette règle : récepteur-milieu-victime-nœud à montée, en 3.18. La règle peut alors réaliser la construction où un verbe à montée gouverne son actant verbal par intermédiaire d'une préposition, comme *Paul commence à partir*.

— `synt_reconnection` permet de reconnecter le nœud perdant son gouverneur au récepteur. Elle prend en charge aussi une structure hiérarchique à trois niveaux : récepteur-nœud à supprimer-nœud à reconnecter, comme l'illustre la figure 3.19. Le nœud à supprimer est souvent un verbe copule généré par la fonction lexicale. Il régit le nœud de base modificateur déclenchant la fonction lexicale et le nœud modifié, sujet de la proposition enchâssée. Si le nœud à supprimer est exclus par la règle nodale, le nœud modifié et le nœud modificateur perdent leur gouverneur. Alors, `synt_raising` monte le nœud modifié à la position du complément d'objet du récepteur ; `synt_reconnection` reconnecte le nœud modificateur au récepteur. Cette

règle s'y applique à condition que le nœud à supprimer soit un verbe copule portant `raise_from` et que la relation entre le nœud à supprimer et le nœud modificateur corresponde à celle du deuxième actant du verbe support, définie dans le dictionnaire `gpcon`. Ainsi, l'adjectif JOLI est rattaché à TROUVER_2, illustré dans la figure 3.11.

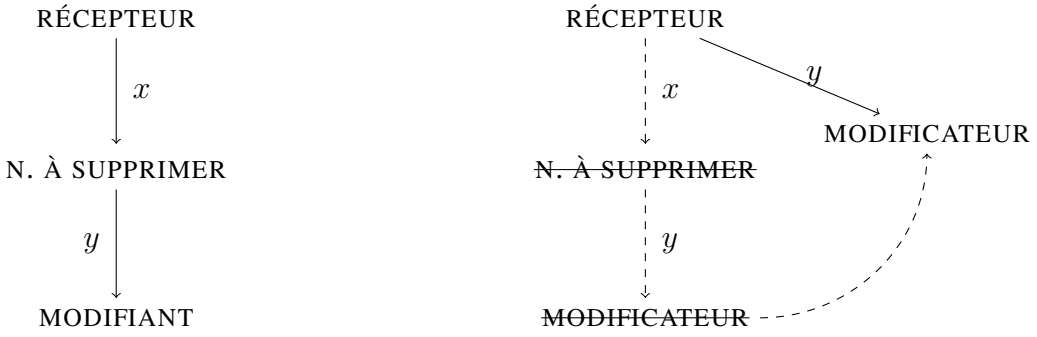


FIGURE 3.19 – Reconnection du MODIFICATEUR au RÉCEPTEUR réalisée par la règle `synt_reconnection`.

Avant de finir cette sous-section, il est important de préciser que les règles `synt_raising` et `synt_raising_with_mid` n'établissent que la relation entre le nœud monté et le récepteur, indiquée par la ligne continue dans les figures 3.17 et 3.18. Les relations illustrées par la ligne pointillée, comme elle le suggère implicitement, ne sont pas créées par ces règles, mais servent uniquement de lignes de repère.

À l'aide des nouveaux modules, GenDR est désormais capable de gérer les verbes à montée et à contrôle.

3.3 Problématique

Bien que l'implémentation des fonctions lexicales ne fasse pas partie de notre objectif principal, ce sujet important mérite des réflexions. Tel que mentionné à la fin de §3.2.2, la fonction lexicale implémentée dans le cadre du présent mémoire n'est pas complète et présente des limitations.

(5) a. *Paul aime Marie.*
b. *Paul est amoureux de Marie.*

Pour les phrases (5), le système ne peut pas générer correctement la paraphrase (5-b) (voir la figure 3.20), mais seulement (5-a). Lorsque la fonction lexicale $\texttt{Oper}_1$ s'est appliquée à

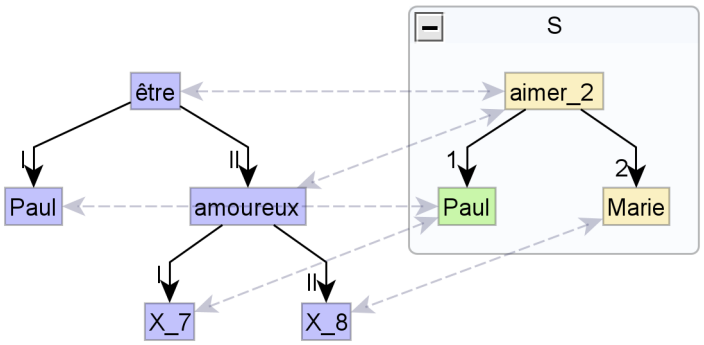


FIGURE 3.20 – Impossible de contraindre des nœuds après la fonction lexicale

AMOUREUX, AMOUREUX s'est divisé en ÊTRE et AMOUREUX. Les actants de la base autres que son premier actant PAUL, n'ont pas été contraints par `constraints_gp`. Le processus s'est arrêté à la création des nœuds anonymes (non lexicalisés) `X_7` et `X_8`. Tandis que si nous vérifions les attributs des nœuds, les conditions pour appliquer la règle de contrainte de nœud, comme la prochaine étape, sont remplies.

Nous supposons que le problème viendrait de l'incompatibilité entre les fonctions lexicales et la version actuelle de GenDR basée sur le *gpcon*, car l'autre graphe dans lequel $\texttt{Oper}_1$ n'est pas appliquée marche correctement, comme l'illustre la figure 3.21, étant donné que les deux graphes sont issus de la même RSém.

Entrons plus en détail dans la version actuelle de GenDR basée sur le `gpcon`.

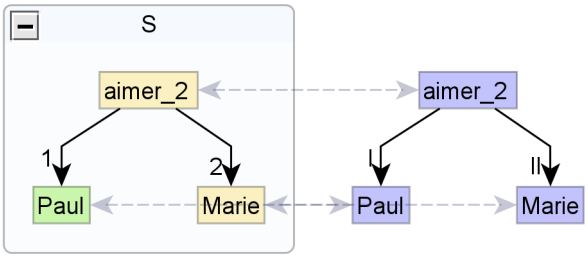


FIGURE 3.21 – La transition à la RSyntP sans appliquer la fonction lexicale

En théorie, nous marquons de chiffre arabe la relation sémantique entre un prédicat et son dépendant dans la RSém. Cette relation sémantique se traduit ensuite en arc syntaxique profond dans la RSyntP. L'ordre des actants s'établit suivant la diathèse encodée dans le régime du prédicat. Ils s'étiquettent souvent de chiffres romains. Toutefois, ce qui est particulier est que la relation qui représente la modification est notée **ATTR** ; la gouvernance est aussi inverse. Tel que nous avons mentionné en §1.3.1, un modifié sémantiquement gouverné par un modificateur le gouverne syntaxiquement par l'arc étiqueté de **ATTR** dans la RSyntP.

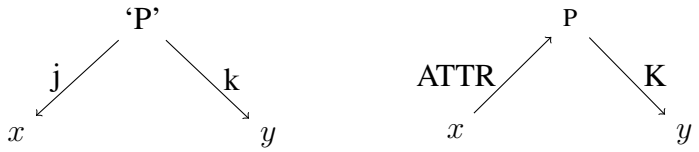


FIGURE 3.22 – RSém et RSyntP d'un adjectif bianctaciel.

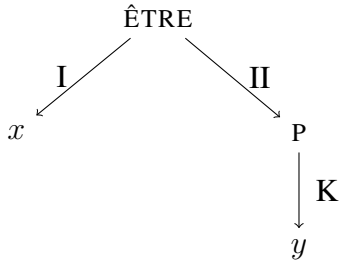


FIGURE 3.23 – RSyntP d'un adjectif bianctaciel après l'application de la $\texttt{Oper}_1$.

Pour un adjectif bi-actanciel ou multi-actanciel (figure 3.22), le prédicat adjectival 'P' gouverne ses dépendants 'x' et 'y' par les arcs **j** et **k** dans la RSém. La relation sémantique **j** devient celle syntaxique étiquetée d'**ATTR** dans la RSyntP. L'autre **k** se traduit en arc syntaxique en chiffre romain **K**. Si la fonction lexicale s'y appliquait, nous aurions la difficulté à positionner **ATTR**, comme le montre la figure 3.23. Parce que le nœud à modifier est détaché de P et s'adjoint au verbe support ÊTRE en tant que sujet, autrement dit son premier actant. Ainsi, x ne peut plus gouverner P par l'arc **ATTR** ; il n'y a plus de position disponible pour la relation de modification (figure 3.23).

Revenons à la figure 3.20. Après l'application de $\texttt{Oper}_1$, se posent les problèmes de la relation de gouvernance entre PAUL et AMOUREUX et la position d'**ATTR** comme dans la figure 3.23. D'ailleurs, même si AMOUREUX ne gouverne qu'un actant MARIE, leur arc devrait être étiqueté de **II** au lieu de **I**, car c'est son premier actant PAUL que AMOUREUX a perdu. Donc, l'incompatibilité entre la version de GenDR basée sur le *gpcon* et les fonctions lexicales semble difficile à contourner.

Cependant, ce problème ne serait pas impossible à résoudre. Nous supposons qu'une solution soit de perfectionner la traduction des relations sémantiques en arcs syntaxiques profonds pour les prédicats adjectivaux dans la version actuelle.

3.4 Synthèse

Dans ce chapitre, nous avons expliqué l'implémentation des verbes à montée et à contrôle au sein de GenDR.

Verbes à contrôle : Dans la RSém, le premier actant du verbe subordonné est coréférent au premier actant du verbe à contrôle. Pour éliminer le sujet à l'infinitif, nous avons ajouté le module utilitaire `ssynt-ssynt-treecheck` comme dernière étape.

Verbes à montée : Nous avons réfléchi au moment optimal d'application d'un nouveau module spécifique pour leur implémentation. Après l'analyse de trois possibilités en 3.5, nous avons choisi le Moment 3 (après la RSyntS). Nous avons également introduit des fonctions lexicales qui servent à gérer des constructions impliquant la montée atypique ainsi que leurs paraphrases en forme longue, entre autres, « *Paul trouve la ville jolie.* » et « *Paul trouve que la ville est jolie.* » Nous avons développé le nouveau module spécifique `ssynt-ssynt` qui comporte une règle nodale et quatre règles syntaxiques qui permettent de gérer les relations directes entre les nœuds, réaliser la montée syntaxique, traiter les cas avec une préposition et reconnecter les nœuds après la suppression d'un nœud, entre autres, un verbe copule.

Pourtant, les fonctions lexicales intégrées présentent une problématique : le système ne contraint pas des actants de la base de la fonction lexicale, en particulier lorsque la base est un prédicat adjectival.

Malgré cette limitation, notre intégration a amélioré les capacités de GenDR, permettant désormais de traiter les verbes à montée et à contrôle.

Chapitre 4

Conclusion

Les verbes à montée et à contrôle jouent un rôle crucial dans l'utilisation quotidienne tant orale qu'écrite. Toutefois, ils impliquent des phénomènes syntaxiques complexes, notamment la coindexation pour les verbes à contrôle et le déplacement syntagmatique pour les verbes à montée et les verbes prenant la proposition réduite. Ces complexités représentent un défi significatif dans le domaine de la GAT. Néanmoins, l'objectif de la présente recherche était de modéliser les verbes à montée et à contrôle français et anglais dans un réalisateur de texte profond multilingue, GenDR.

GenDR, fondé sur la théorie Sens-Texte, s'exécute sur un transducteur de graphes, MATE. Il prend en entrée une RSém et renvoie une ou plusieurs RSyntS, en passant par une ou des RSyntP. La transduction nécessite des ressources lexicales appropriées et une grammaire qui permet la lexicalisation et l'arborisation.

Dans ce contexte, notre objectif peut se diviser en deux sous-objectifs :

1. Construire des dictionnaires adaptés à l'utilisation de GenDR qui encode les comportements syntaxiques des verbes à montée et à contrôle.
2. Concevoir des règles permettant de modéliser les phénomènes syntaxiques pertinents.

La structure de ce mémoire reflète l'organisation de notre travail, divisé en deux chapitres principaux correspondant à ces tâches de recherche.

Dans le chapitre 2, nous avons procédé à l'extraction des phrases comportant les verbes à montée et à contrôle. Ce processus a commencé par déterminer la ressource lexicale riche. Nous avons choisi les corpus représentés par les schèmes d'annotation SUD dont les étiquettes répondent à notre besoin. Ensuite, nous avons extrait des corpus sous SUD toutes les phrases contenant les verbes à montée et à contrôle compilées en format JSON, à l'aide de

Grew. Nous avons développé un script en Python permettant de renvoyer les patrons de régime en tableau et de compiler les résultats dans un fichier JSON modifiable. Puis nous avons utilisé ce script pour identifier les patrons de régime pour chaque verbe à partir des phrases extraites. Les patrons de régime renvoyés ont fait l'objet des travaux manuels : le nettoyage et la désambiguïsation. Enfin, nous avons élaboré trois dictionnaires : sémantique, lexical et de patrons de régime. Nous avons ainsi atteint notre premier objectif.

Le chapitre 3 présente le développement et l'implémentation d'un module spécifique pour les verbes à montée et à contrôle dans la version de GenDR basée sur un *gpcon* développée par Galarreta-Piquette (2018). D'abord, nous avons déterminé que le moment approprié de l'application du nouveau module est après la RSyntS. Ensuite, nous avons créé des règles permettant les fonctions de base et le traitement des phénomènes syntaxiques pour la RSyntS d'entrée. De plus, nous avons implémenté la fonction lexicale partielle dans le module `sem-dsynt` pour améliorer la flexibilité du paraphrasage des verbes à montée et d'autres verbes. Ainsi, la seconde tâche s'est achevée et nous avons atteint l'objectif principal.

En outre, une petite investigation menée à la fin du chapitre 3 a permis d'élucider l'origine de l'erreur qui se produit lorsque le système applique la fonction lexicale pour transformer un verbe en *être* et sa forme adjectivale. Cette découverte ouvre une piste de recherche qui mériterait d'être approfondie dans de futurs travaux.

Notre travail apporte plusieurs contributions. Tout d'abord, nous avons élargi la couverture de GenDR en intégrant les verbes à montée et à contrôle, ainsi que les verbes syntaxiquement similaires aux verbes à montée. Par ailleurs, nous avons développé une approche pour extraire les patrons de régime des SUD. Puisque les SUD supportent les corpus de nombreuses langues, notre approche présente l'avantage d'être réutilisable pour acquérir les patrons de régime de ces langues et en construire des dictionnaires.

Enfin, notre recherche a abouti à une avancée inattendue : elle ouvre la possibilité d'implémenter les fonctions lexicales complètes dans la version basée sur le *gpcon*. Cette percée est importante, car l'intégration des fonctions lexicales dans cette version demeure jusqu'ici un défi important et permettrait d'exploiter pleinement le potentiel de GenDR, optimisant considérablement ses performances, notamment la capacité du paraphrasage.

Bibliographie

AHRENBERG, L. (2007). LinES : An English-Swedish parallel treebank. Dans NIVRE, J., KAALEP, H.-J., MUISCHNEK, K. et KOIT, M., dir., *Proceedings of the 16th Nordic Conference of Computational Linguistics (NODALIDA 2007)*, pp. 270–273, Tartu, Estonia. University of Tartu, Estonia.

AOYAMA, T., BEHZAD, S., GESSLER, L., LEVINE, L., LIN, J., LIU, Y. J., PENG, S., ZHU, Y. et ZELDES, A. (2023). GENTLE : A genre-diverse multilayer challenge set for English NLP and linguistic evaluation. Dans *Proceedings of the 17th Linguistic Annotation Workshop (LAW-XVII)*, pp. 166–178, Toronto, Canada.

BEHZAD, S. et ZELDES, A. (2020). A cross-genre ensemble approach to robust Reddit part of speech tagging. Dans *Proceedings of the 12th Web as Corpus Workshop (WAC-XII)*, pp. 50–56.

BOHNET, B., LANGJAHR, A. et WANNER, L. (2000). A development environment for an MTT-based sentence generator. Dans ELHADAD, M., dir., *INLG'2000 Proceedings of the First International Conference on Natural Language Generation*, pp. 260–263, Mitzpe Ramon, Israel. Association for Computational Linguistics.

BOHNET, B. et WANNER, L. (2010). Open soucre graph transducer interpreter and grammar development environment. Dans CALZOLARI, N., CHOUKRI, K., MAEGAARD, B., MARIANI, J., ODIJK, J., PIPERIDIS, S., ROSNER, M. et TAPIAS, D., dir., *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

BONFANTE, G., GUILLAUME, B. et PERRIER, G. (2018). *Application of Graph Rewriting to Natural Language Processing*. John Wiley & Sons.

CANDITO, M. et SEDDAH, D. (2012). Le corpus sequoia : annotation syntaxique et exploitation pour l'adaptation d'analyseur par pont lexical (the sequoia corpus : Syntactic

annotation and use for a parser lexical domain adaptation method) [in French]. Dans ANTONIADIS, G., BLANCHON, H. et SÉRASSET, G., dir., *Proceedings of the Joint Conference JEP-TALN-RECITAL 2012, volume 2 : TALN*, pp. 321–334, Grenoble, France. ATALA/AFCP.

CHOMSKY, N. (1973). Conditions on transformations. *A Festschrift for Morris Halle/Holt, Reinhart and Winston*.

CHOMSKY, N. (1980). On binding. *Linguistic inquiry*, 11(1):1–46.

CHOMSKY, N. (1981). *Lectures on Government and Binding*. Walter de Gruyter GmbH & Company KG.

CRUSE, D. (1986). *Lexical Semantics*. Cambridge Textbooks in Linguistics. Cambridge University Press.

de MARNEFFE, M.-C., CONNOR, M., SILVEIRA, N., BOWMAN, S. R., DOZAT, T. et MANNING, C. D. (2013). More constructions, more genres : Extending Stanford dependencies. Dans HAJIČOVÁ, E., GERDES, K. et WANNER, L., dir., *Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013)*, pp. 187–196, Prague, Czech Republic. Charles University in Prague, Matfyzpress, Prague, Czech Republic.

de MARNEFFE, M.-C., MACCARTNEY, B. et MANNING, C. D. (2006). Generating typed dependency parses from phrase structure parses. Dans CALZOLARI, N., CHOUKRI, K., GANGEMI, A., MAEGAARD, B., MARIANI, J., ODIJK, J. et TAPIAS, D., dir., *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy. European Language Resources Association (ELRA).

de MARNEFFE, M.-C. et MANNING, C. D. (2008). The Stanford typed dependencies representation. Dans BOS, J., BRISCOE, E., CAHILL, A., CARROLL, J., CLARK, S., COPESTAKE, A., FLICKINGER, D., van GENABITH, J., HOCKENMAIER, J., JOSHI, A., KAPLAN, R., KING, T. H., KUEBLER, S., LIN, D., LØNNING, J. T., MANNING, C., MIYAO, Y., NIVRE, J., OEPEN, S., SAGAE, K., XUE, N. et ZHANG, Y., dir., *Coling 2008 : Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pp. 1–8, Manchester, UK. Coling 2008 Organizing Committee.

DUBÉ, M. et LAREAU, F. (2022). Handling idioms in symbolic multilingual natural language generation. Dans *Proceedings of the 18th Workshop on Multiword Expressions*

(MWE@LREC22), pp. 118–126, Marseille, France. European Language Resources Association.

DUBINSKAITE, I. (2017). Développement de ressources lituaniennes pour un générateur automatique de texte multilingue. Mémoire de maîtrise, Université de Montréal, Montréal, QC.

GALARRETA-PIQUETTE, D. (2018). Intégration de verbnet dans un réalisateur profond. Mémoire de maîtrise, Université de Montréal, Montréal, QC.

GERDES, K., GUILLAUME, B., KAHANE, S. et PERRIER, G. (2018a). SUD or Surface-Syntactic Universal Dependencies : An annotation scheme near-isomorphic to UD. Dans LYNN, T. et SCHUSTER, S., dir., *Universal Dependencies Workshop 2018*, Brussels, Belgium.

GERDES, K., GUILLAUME, B., KAHANE, S. et PERRIER, G. (2019). Improving Surface-syntactic Universal Dependencies (SUD) : surface-syntactic relations and deep syntactic features. Dans *TLT 2019 - 18th International Workshop on Treebanks and Linguistic Theories*, pp. 126–132, Paris, France. Association for Computational Linguistics.

GERDES, K., GUILLAUME, B., KAHANE, S. et PERRIER, G. (2022). Starting a new treebank ? Go SUD ! Theoretical and practical benefits of the Surface-Syntactic distributional approach. Dans *SyntaxFest Depling 2021 - 6th International Conference on Dependency Linguistics*, pp. 35–46, Sofia, Bulgaria. Association for Computational Linguistics.

GERDES, K., KAHANE, S., NAKHLÉ, M., YAN, C., ETIENNE, A. et COURTIN, M. (2018b). Ud english-atis. `https://github.com/UniversalDependencies/UD_French-Rhapsodie`.

GUILLAUME, B. (2019). Graph matching for corpora exploration. Dans *JLC 2019 - 10èmes Journées Internationales de la Linguistique de corpus*, Grenoble, France.

GUILLAUME, B. (2021). Graph Matching and Graph Rewriting : GREW tools for corpus exploration, maintenance and conversion. Dans *EACL 2021 - 16th conference of the European Chapter of the Association for Computational Linguistics : System Demonstrations*, Kiev/Online, Ukraine.

GUILLAUME, B., de MARNEFFE, M.-C. et PERRIER, G. (2019). Conversion et améliorations de corpus du français annotés en Universal Dependencies. *Revue TAL : traitement automatique des langues*, 60(2):71–95.

HE, L. (2020). Un dictionnaire de régimes verbaux en mandarin. Mémoire de maîtrise, Université de Montréal, Montréal, QC.

HORNSTEIN, N. (1999). Movement and Control. *Linguistic Inquiry*, 30(1):69–96.

KAHANE, S., CARON, B., STRICKLAND, E. et GERDES, K. (2021). Annotation guidelines of UD and SUD treebanks for spoken corpora : A proposal. Dans DAKOTA, D., EVANG, K. et KÜBLER, S., dir., *Proceedings of the 20th International Workshop on Treebanks and Linguistic Theories (TLT, SyntaxFest 2021)*, pp. 35–47, Sofia, Bulgaria. Association for Computational Linguistics.

KAHANE, S. et LAREAU, F. (2005). Grammaire d'unification sens-texte : modularité et polarisation. Dans JARDINO, M., dir., *Actes de la 12ème conférence sur le Traitement Automatique des Langues Naturelles. Articles longs*, pp. 21–30, Dourdan, France. ATALA.

KIPPER, K., KORHONEN, A., RYANT, N. et PALMER, M. (2007). A large-scale classification of english verbs. *Language Resources and Evaluation*, 42(1):21–40.

KUZGUN, A., CESUR, N. et YILDIZ, O. T. (2021). Ud english-atis. `https://github.com/UniversalDependencies/UD_English-Atis/tree/master`.

KYLE, K., EGUCHI, M., MILLER, A. et SITHER, T. (2022). A dependency treebank of spoken second language english. pp. 39–45.

LAMBREY, F. (2016). Implémentation des collocations pour la réalisation de texte multilingue. Mémoire de maîtrise, Université de Montréal, Montréal, QC.

LAMBREY, F. et LAREAU, F. (2015). Le traitement des collocations en génération de texte multilingue. Dans LECARPENTIER, J.-M. et LUCAS, N., dir., *Actes de la 22e conférence sur le Traitement Automatique des Langues Naturelles. Articles courts*, pp. 263–269, Caen, France. ATALA.

LAREAU, F., LAMBREY, F., DUBINSKAITE, I., GALARRETA-PIQUETTE, D. et NEJAT, M. (2018). GenDR : A generic deep realizer with complex lexicalization. Dans *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

LAREAU, F. et WANNER, L. (2007). Towards a generic multilingual dependency grammar for text generation. Dans KING, T. H. et BENDER, E. M., dir., *Proceedings of the GEAF07 Workshop*, pp. 203–223, Stanford. CSLI.

LEVIN, B. (1993). *English verb classes and alternations : A preliminary investigation*. The University of Chicago Press, Chicago.

MARTIN, R. (2001). Null Case and the Distribution of PRO. *Linguistic Inquiry*, 32(1):141–166.

MEL'CUK, I. (2013). Tout ce que nous voulions savoir sur les phrasèmes, mais. *Cahiers de Lexicologie*, pp. 129–149.

MEL'ČUK, I. et POLGUÈRE, A. (2009). *Dependency in linguistic description*. John Benjamins Publishing Company.

MELČUK, I. et WANNER, L. (2001). Towards a lexicographic approach to lexical transfer in machine translation (illustrated by the german–russian language pair). *Machine Translation*, 16(1):21–87.

MEL'ČUK, I. (1988). *Dependency Syntax : Theory and Practice*. Daw Book Collectors. State University Press of New York.

MEL'ČUK, I. (1995). The future of the lexicon in linguistic description and the explanatory combinatorial dictionary. Dans LEE, I.-H., dir., *Linguistics in the morning calm*, vol. 3, Hanshin, Seoul.

MEL'ČUK, I. (1996). Lexical functions in lexicography and natural language processing. *Lexical Functions : A Tool for the Description of Lexical Relations in the Lexicon*, pp. 37–102.

MEL'ČUK, I. (2004). Actants in semantics and syntax i : Actants in semantics. *Linguistics*, 42(1):1–66.

MEL'ČUK, I. et MILIĆEVIĆ, J. (2013). *Introduction à la linguistique*, vol. 1. Hermann.

MEL'ČUK, I. (2011). Dependency in language-2011. Dans *Proceedings of International Conference on Dependency Linguistics*, pp. 1–16. Citeseer.

MEL'ČUK, I. (1997). Vers une linguistique sens-texte. Rapport, Collège de France.

MILIĆEVIĆ, J. (2009). Schéma de régime : le pont entre le lexique et la grammaire. *Langages*, 176.

PETROV, S., DAS, D. et MCDONALD, R. (2012). A universal part-of-speech tagset. Dans CALZOLARI, N., CHOUKRI, K., DECLERCK, T., DOĞAN, M. U., MAEGAARD, B., MARIANI, J., MORENO, A., ODIJK, J. et PIPERIDIS, S., dir., *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pp. 2089–2096, Istanbul, Turkey. European Language Resources Association (ELRA).

POLGUÈRE, A. (1990). *Structuration et mise en jeu procédurale d'un modèle linguistique déclaratifdans un cadre de génération de texte*. Thèse de doctorat, Université de Montréal.

POLGUÈRE, A. (1998). La théorie sens-texte. *Dialangue*, 8(9):9–30.

RADFORD, A. (1988). *The Lexicon*, p. 337–400. Cambridge Textbooks in Linguistics. Cambridge University Press.

ROUVERET, A. et VERGNAUD, J.-R. (1980). Specifying reference to the subject : French causatives and conditions on representations. *Linguistic Inquiry*, 11(1):97–202.

SANGUINETTI, M. et BOSCO, C. (2015). Parttut : The turin university parallel treebank. Dans BASILI, R., BOSCO, C., DELMONTE, R., MOSCHITTI, A. et SIMI, M., dir., *Harmonization and Development of Resources and Tools for Italian Natural Language Processing within the PARLI Project*, vol. 589 de *Studies in Computational Intelligence*, pp. 51–69. Springer.

SCHULER, K. K. (2005). *Verbnet : A Broad-coverage, Comprehensive Verb Lexicon*. Thèse de doctorat, University of Pennsylvania, Philadelphia, PA, USA.

SEDDAH, D. et CANDITO, M. (2016). Hard Time Parsing Questions : Building a QuestionBank for French. Dans *Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Proceedings of the 10th edition of the Language Resources and Evaluation Conference (LREC 2016), Portorož, Slovenia.

SILVEIRA, N., DOZAT, T., de MARNEFFE, M.-C., BOWMAN, S., CONNOR, M., BAUER, J. et MANNING, C. D. (2014). A gold standard dependency corpus for English. Dans *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*.

TAYLOR, A. (2006). *Treebank 2a Guidelines*. University of York.

TELLIER, C. (2016). *Éléments de syntaxe du français 3e édition*, chap. 10, pp. 171–195. Chenelière.

USZKOREIT, H., MACKETANZ, V., BURCHARDT, A., HARRIS, K., MARHEINECKE, K., PETROV, S., KAYADELEN, T., ATTIA, M., ELKAHKY, A., YU, Z., PITLER, E., LERTPRADIT, S., KIRCHNER, J., LAMBERTINO, L., POPEL, M., ZEMAN, D., MANNING, C., SCHUSTER, S. et REDDY, S. (2017). Ud english-pud. `https://github.com/UniversalDependencies/UD_English-PUD/blob/master`.

WANNER, L. (1992). Lexical choice and the organization of lexical resources in text generation. Dans *Proceedings of the 10th European conference on Artificial intelligence*, pp. 495–499.

WILLIAMS, E. S. (1975). *Small Clauses in English*, pp. 249 – 273. Brill, Leiden, The Netherlands.

ZELDES, A. (2017). The GUM corpus : creating multilayer resources in the classroom. *Language Resources and Evaluation*, 51(3):581–612.

ZEMAN, D. (2008). Reusable tagset conversion using tagset drivers. Dans CALZOLARI, N., CHOUKRI, K., MAEGAARD, B., MARIANI, J., ODIJK, J., PIPERIDIS, S. et TAPIAS, D., dir., *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA).

ŽOLKOVSKIJ, A. et MEL'ČUK, I. (1967). O sisteme semantičeskogo sinteza. ii : Pravila preobrazovanija. *Informacionnye processy i sistemy*, 2:17–27.

Annexe A

Données

A.1 *gpcon* français

Listing A.1 – *gpcon* français

```
gpcon {

//apparaître_4 paraître_7 sembler_7
Exp_V_ADJ {
  I={rel=subj dpos=Exp dlex=il_expl}
  II={rel=comp_pred dpos=Adj}
}

//apparaître_5
Exp_V_que_Vind {
  I={rel=subj dpos=Exp dlex=il_expl}
  II={rel=comp_obj dpos=V mood=IND conj=que}
}

//paraître_3 paraître_5 sembler_3 sembler_5
Exp_V_que_Vind_à_N {
  I={rel=subj dpos=Exp dlex=il_expl}
  II={rel=comp_obj dpos=V mood=IND conj=que}
  III={rel=comp_obl dpos=N prep=à}
}
```

```
//paraître_6 sembler_6
Exp_V_que_Vsub {
  I={rel=subj dpos=Exp dlex=il_expl}
  II={rel=comp_obj dpos=V finiteness=FIN mood=SUB conj=que}
}

//apparaître arrêter commencer continuer finir échouer
N_V {
  I={rel=subj dpos=N}
}

//apparaître_3 paraître sembler trouver
N_V_ADJ {
  I={rel=subj dpos=N}
  II={rel=comp_pred dpos=Adj}
}

//conserver déclarer garder prouver rendre trouver_2
N_V_ADJ_N {
  I={rel=subj dpos=N}
  II={rel=comp_pred dpos=V raise_from=subj raise_to=comp_obj}
}

//paraître paraître_2 sembler_2
N_V_ADJ_à_N {
  I={rel=comp_pred dpos=V raise_from=subj raise_to=subj}
  II={rel=comp_obl dpos=N prep=à}
}

//arrêter_3 commencer_3 continuer_3 finir_4 paraître_4 sembler_4
N_V_N {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=N}
}

//entendre laisser regarder voir
N_V_N_Vinf {
```

```
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=N}
  III={rel=comp_obj dpos=V finiteness=INF}
}

//commencer_4
N_V_N_avec_N {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=N}
  III={rel=comp_obl dpos=N prep=avec}
}

//accuser avertir contraindre convaincre presser prier
N_V_N_de_Vinf {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=N}
  III={rel=comp_obl dpos=V finiteness=INF prep=de}
}

//commencer_4
N_V_N_par_N {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=N}
  III={rel=comp_obl dpos=N prep=par}
}

//paraître_5 sembler_5
N_V_N_à_N {
  I={rel=comp_pred dpos=V raise_from=subj raise_to=subj}
  II={rel=comp_obl dpos=N prep=à}
}

//aider amener appeler autoriser conduire contraindre déterminer
   encourager engager forcer inciter inviter obliger résoudre_2
   se_plaire
N_V_N_à_Vinf {
  I={rel=subj dpos=N}
```

```
  II={rel=comp_obj dpos=N}
  III={rel=comp_obl dpos=V finiteness=INF prep=à}
}

//adorer aimer détester entendre envoyer espérer laisser nier oser
   partir préférer prétendre prévoir regarder savoir souhaiter
   voir vouloir
N_V_Vinf {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=V finiteness=INF}
}

//accepter choisir comploter demander décider entreprendre essayer
   exiger imaginer jurer menacer mériter oublier permettre
   promettre proposer refuser risquer résoudre se_assurer
   se_contenter se_dépêcher se_efforcer se_rappeler se_soucier
   suffire_2 éviter
N_V_de_Vinf {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=V finiteness=INF prep=de}
}

//demander ordonner permettre promettre proposer
N_V_de_Vinf_à_N {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=V finiteness=INF prep=de}
  III={rel=comp_obl dpos=N prep=à}
}

//demander
N_V_de_Vinf_à_N_par_N {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=V finiteness=INF prep=de}
  III={rel=comp_obl dpos=N prep=à}
  IV={rel=comp_obl dpos=N prep=par}
}
```

```
//commencer_4
N_V_par_N {
  I={rel=subj dpos=N}
  II={rel=comp_obl dpos=N prep=par}
}

//déclarer prouver trouver_2
N_V_que_N_Vfin_ADJ {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=V finiteness=FIN conj=que}
}

//échouer_3
N_V_sur_N {
  I={rel=subj dpos=N}
  II={rel=comp_obl dpos=N prep=sur}
}

//apparaître_2
N_V_à_N {
  I={rel=subj dpos=N}
  II={rel=comp_obl dpos=N prep=à}
}

//apprendre
N_V_à_N_à_Vinf {
  I={rel=subj dpos=N}
  II={rel=comp_obl dpos=N prep=à}
  III={rel=comp_obl dpos=V finiteness=INF prep=à}
}

//aider appeler apprendre_2 chercher conspirer contribuer
   demander_2 hésiter inciter parvenir se_accorder se_limiter
   se_préparer se_résigner suffire tarder tenir tenter viser
N_V_à_Vinf {
  I={rel=subj dpos=N}
  II={rel=comp_obl dpos=V finiteness=INF prep=à}
```

```
}

//aller devoir paraître paraître_6 pouvoir sembler_6
N_Vrs_Vinf {
  I={rel=comp_pred dpos=V finiteness=INF raise_from=subj
   raise_to=subj}
}

//paraître_3 sembler_3
N_Vrs_Vinf_à_N {
  I={rel=comp_obj dpos=V finiteness=INF raise_from=subj
   raise_to=subj}
  II={rel=comp_obl dpos=N prep=à}
}

//arrêter_2 continuer_2 finir_2 venir_de
N_Vrs_de_Vinf {
  I={rel=comp_obj dpos=V finiteness=INF prep=de raise_from=subj
   raise_to=subj}
}

//commencer_5 finir_3
N_Vrs_par_Vinf {
  I={rel=comp_obl dpos=V finiteness=INF prep=par raise_from=subj
   raise_to=subj}
}

//avoir_à commencer_2 continuer_2 échouer échouer_2
N_Vrs_à_Vinf {
  I={rel=comp_obl dpos=V finiteness=INF prep=à raise_from=subj
   raise_to=subj}
}

}
}
```

A.2 *lexicon* français

Listing A.2 – *lexicon* français

```
lexicon {

accepter : "accepter"
accuser : "accuser"
adorer : "adorer"
aider : "aider"
aimer : "adorer"
aller : "aller"
amener : "amener"
apparaître : "apparaître"
apparaître_2 : "apparaître_2"
apparaître_3 : "apparaître_3"
apparaître_4 : "apparaître_4"
apparaître_5 : "apparaître_5"
appeler : "aider"
apprendre : "apprendre"
apprendre_2 : "apprendre_2"
arrêter : "apparaître"
arrêter_2 : "arrêter_2"
arrêter_3 : "arrêter_3"
autoriser : "amener"
avertir : "accuser"
avoir_à : "avoir_à"
chercher : "apprendre_2"
choisir : "accepter"
commencer : "apparaître"
commencer_2 : "avoir_à"
commencer_3 : "arrêter_3"
commencer_4 : "commencer_4"
commencer_5 : "commencer_5"
comploter : "accepter"
conduire : "amener"
conserver : "conserver"
conspirer : "apprendre_2"
```

```
continuer : "apparaître"
continuer_2 : "continuer_2"
continuer_3 : "arrêter_3"
contraindre : "contraindre"
contribuer : "apprendre_2"
convaincre : "accuser"
demander : "demander"
demander_2 : "apprendre_2"
devoir : "aller"
décider : "accepter"
déclarer : "déclarer"
déterminer : "amener"
détester : "adorer"
encourager : "amener"
engager : "amener"
entendre : "entendre"
entreprendre : "accepter"
envoyer : "adorer"
espérer : "adorer"
essayer : "accepter"
exiger : "accepter"
finir : "apparaître"
finir_2 : "arrêter_2"
finir_3 : "commencer_5"
finir_4 : "arrêter_3"
forcer : "amener"
garder : "conserver"
hésiter : "apprendre_2"
imaginer : "accepter"
inciter : "aider"
inviter : "amener"
jurer : "accepter"
laisser : "entendre"
menacer : "accepter"
mériter : "accepter"
nier : "adorer"
obliger : "amener"
```

```
ordonner : "ordonner"
oser : "adorer"
oublier : "accepter"
paraître : "paraître"
paraître_2 : "paraître_2"
paraître_3 : "paraître_3"
paraître_4 : "arrêter_3"
paraître_5 : "paraître_5"
paraître_6 : "paraître_6"
paraître_7 : "apparaître_4"
partir : "adorer"
parvenir : "apprendre_2"
permettre : "permettre"
pouvoir : "aller"
presser : "accuser"
prier : "accuser"
promettre : "permettre"
proposer : "permettre"
prouver : "déclarer"
préférer : "adorer"
prétendre : "adorer"
prévoir : "adorer"
refuser : "accepter"
regarder : "entendre"
rendre : "conserver"
risquer : "accepter"
résoudre : "accepter"
résoudre_2 : "amener"
savoir : "adorer"
se_accorder : "apprendre_2"
se_assurer : "accepter"
se_contenter : "accepter"
se_dépêcher : "accepter"
se_efforcer : "accepter"
se_limiter : "apprendre_2"
se_plaire : "amener"
se_préparer : "apprendre_2"
```

```
se_rappeler : "accepter"
se_résigner : "apprendre_2"
se_soucier : "accepter"
sembler : "apparaître_3"
sembler_2 : "paraître_2"
sembler_3 : "paraître_3"
sembler_4 : "arrêter_3"
sembler_5 : "paraître_5"
sembler_6 : "paraître_6"
sembler_7 : "apparaître_4"
souhaiter : "adorer"
suffire : "apprendre_2"
suffire_2 : "accepter"
tarder : "apprendre_2"
tenir : "apprendre_2"
tenter : "apprendre_2"
trouver : "apparaître_3"
trouver_2 : "déclarer"
venir_de : "arrêter_2"
viser : "apprendre_2"
voir : "entendre"
vouloir : "adorer"
échouer : "échouer"
échouer_2 : "avoir_à"
échouer_3 : "échouer_3"
éviter : "accepter"
"accepter": verb {
  gp = { id=N_V_de_Vinf   dia=12}
}

"accuser": verb {
  gp = { id=N_V_N_de_Vinf   dia=123}
}

"adorer": verb {
  gp = { id=N_V_Vinf    dia=12}
}
```

```
"aider": verb {
  gp = { id=N_V_N_à_Vinf    dia=123}
  gp = { id=N_V_à_Vinf    dia=13}
}

"aller": verb {
  gp = { id=N_Vrs_Vinf    dia=1}
}

"amener": verb {
  gp = { id=N_V_N_à_Vinf    dia=123}
}

"apparaître": verb {
  gp = { id=N_V   dia=1}
}

"apparaître_2": verb {
  gp = { id=N_V_ADJ   dia=12}
}

"apparaître_3": verb {
  gp = { id=N_V_à_N   dia=12}
}

"apparaître_4": verb {
  gp = { id=Exp_V_ADJ   dia=X1}
}

"apparaître_5": verb {
  gp = { id=Exp_V_que_Vind    dia=X1}
}

"apprendre": verb {
  gp = { id=N_V_à_N_à_Vinf    dia=123}
}
```

```
"apprendre_2": verb {
  gp = { id=N_V_à_Vinf    dia=12}
}

"arrêter_2": verb {
  gp = { id=N_Vrs_de_Vinf   dia=1}
}

"arrêter_3": verb {
  gp = { id=N_V_N   dia=12}
}

"avoir_à": verb {
  gp = { id=N_Vrs_à_Vinf    dia=1}
}

"commencer_4": verb {
  gp = { id=N_V_N_avec_N    dia=123}
  gp = { id=N_V_N_par_N   dia=123}
  gp = { id=N_V_par_N   dia=13}
}

"commencer_5": verb {
  gp = { id=N_Vrs_par_Vinf    dia=1}
}

"conserver": verb {
  gp = { id=N_V_ADJ_N   dia=12}
}

"continuer_2": verb {
  gp = { id=N_Vrs_de_Vinf   dia=1}
  gp = { id=N_Vrs_à_Vinf    dia=1}
}

"contraindre": verb {
```

```
  gp = { id=N_V_N_de_Vinf   dia=123}
  gp = { id=N_V_N_à_Vinf    dia=123}
}

"demander": verb {
  gp = { id=N_V_de_Vinf   dia=12}
  gp = { id=N_V_de_Vinf_à_N   dia=123}
  gp = { id=N_V_de_Vinf_à_N_par_N   dia=1234}
}

"déclarer": verb {
  gp = { id=N_V_ADJ_N   dia=12}
  gp = { id=N_V_que_N_Vfin_ADJ    dia=12}
}

"entendre": verb {
  gp = { id=N_V_N_Vinf    dia=123}
  gp = { id=N_V_Vinf    dia=12}
}

"ordonner": verb {
  gp = { id=N_V_de_Vinf_à_N   dia=123}
}

"paraître_2": verb {
  gp = { id=N_V_ADJ_à_N   dia=12}
}

"paraître_3": verb {
  gp = { id=Exp_V_que_Vind_à_N    dia=X12}
  gp = { id=N_Vrs_Vinf_à_N    dia=12}
}

"paraître_5": verb {
  gp = { id=Exp_V_que_Vind_à_N    dia=X12}
  gp = { id=N_V_N_à_N   dia=12}
}
```

```
"paraître_6": verb {
  gp = { id=Exp_V_que_Vsub    dia=X1}
  gp = { id=N_Vrs_Vinf    dia=1}
}

"permettre": verb {
  gp = { id=N_V_de_Vinf   dia=12}
  gp = { id=N_V_de_Vinf_à_N   dia=123}
}

"échouer_3": verb {
  gp = { id=N_V_sur_N   dia=12}
}

}
```

A.3 *gpcon* anglais

Listing A.3 – *gpcon* anglais

```
gpcon {

//appear_4 seem_7
Exp_V_ADJ {
  I={rel=subj dpos=Exp dlex=it_expl}
  II={rel=comp_pred dpos=Adj}
}

//appear_5 seem_6
Exp_V_that_Vind {
  I={rel=subj dpos=Exp dlex=it_expl}
  II={rel=comp_obj dpos=V mood=IND conj=that}
}

//seem_3 seem_5
Exp_V_that_Vind_to_N {
```

```
  I={rel=subj dpos=Exp dlex=it_expl}
  II={rel=comp_obj dpos=V mood=IND conj=that}
  III={rel=comp_obl dpos=N prep=to}
}

//appear begin continue fail happen remain start stop
N_V {
  I={rel=subj dpos=N}
}

//appear_2 end_up keep prove remain_2 seem
N_V_ADJ {
  I={rel=subj dpos=N}
  II={rel=comp_pred dpos=Adj}
}

//keep_2 make prefer_2 prove_2 report_2 stain
N_V_ADJ_N {
  I={rel=subj dpos=N}
  II={rel=comp_pred dpos=V raise_from=subj raise_to=comp_obj}
}

//seem_2
N_V_ADJ_to_N {
  I={rel=comp_pred dpos=Adj raise_from=subj raise_to=subj}
  II={rel=comp_obl dpos=N prep=to}
}

//begin_2 continue_2 remain_3 seem_4 start_2 stop_2
N_V_N {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=N}
}

//catch leave want_2
N_V_N_Vger {
  I={rel=subj dpos=N}
```

```
  II={rel=comp_obj dpos=N}
  III={rel=comp_obl dpos=V finiteness=GER}
}

//help let make_2
N_V_N_Vinf {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=N}
  III={rel=comp_obl dpos=V finiteness=INF}
}

//warn_2
N_V_N_against_Vger {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=N}
  III={rel=comp_obl dpos=V finiteness=GER prep=against}
}

//prod push
N_V_N_into_Vger {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=N}
  III={rel=comp_obl dpos=V finiteness=GER prep=into}
}

//seem_5
N_V_N_to_N {
  I={rel=comp_pred dpos=V raise_from=subj raise_to=subj}
  II={rel=comp_obl dpos=N prep=to}
}
```

```
//admonish allow ask authorize bribe caution choose_2 commission
   compel convince direct dispatch empower enable encourage entice
   expect_2 force help hire impel induce influence inspire invite
   lead leave license like_2 move need_2 order permit pledge
   prefer_3 press prod program prompt push report require
   sensitize slate spur take_occasion_to teach tell trust urge
   want_2 warn wish wish_2
N_V_N_to_Vinf {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=N}
  III={rel=comp_obl dpos=V finiteness=INF prep=to}
}

//admit allow avoid bother delay deny end_up_2 enjoy like love
   mean mind miss prefer propose quit require risk stand try
N_V_Vger {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=V finiteness=GER}
}

//figure
N_V_Vind {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=V mood=IND conj=that}
}

//help let need prepare
N_V_Vinf {
  I={rel=subj dpos=N}
  II={rel=comp_obj dpos=V finiteness=INF}
}

//warn_2
N_V_against_Vger {
  I={rel=subj dpos=N}
  II={rel=comp_obl dpos=V finiteness=GER prep=against}
}
```

```
//admit afford agree aim allow apply arrange ask attempt bother
   care choose claim come compel conspire decide decline deserve
   determine elect expect flock forget get_to go_to guarantee hate
   have_to help hesitate hope intend know learn leave like look
   love manage mean motivate need negotiate oblige offer opt plan
   pledge plot prefer prepare proceed profess promise prompt
   propose refuse require resolve rush schedule scramble seek
   select set_out sound strive struggle suffice swear tell
   threaten try undertake use_to venture vote vow wait want wish
N_V_to_Vinf {
  I={rel=subj dpos=N}
  II={rel=comp_obl dpos=V finiteness=INF prep=to}
}

//sign
N_V_to_Vinf_for_N {
  I={rel=subj dpos=N}
  II={rel=comp_obl dpos=V finiteness=INF prep=to}
  III={rel=comp_obl dpos=N prep=for}
}

//sign
N_V_to_Vinf_to_N {
  I={rel=subj dpos=N}
  II={rel=comp_obl dpos=V finiteness=INF prep=to}
  III={rel=comp_obl dpos=N prep=to}
}

//stop_5
N_Vrs_N_Vger {
  I={rel=comp_obj dpos=V raise_from=subj raise_to=subj}
}

//begin_3 continue_3 keep_3 start_3 stop_4
N_Vrs_Vger {
```

```
  I={rel=comp_obj dpos=V finiteness=GER raise_from=subj
   raise_to=subj}
}

//seem_3
N_Vrs_Vinf_to_N {
  I={rel=comp_obj dpos=V finiteness=INF raise_from=subj
   raise_to=subj}
  II={rel=comp_obl dpos=N prep=to}
}

//appear_3 begin_3 continue_3 fail_2 ought prove_3 remain_4 seem_6
   start_3 stop_3 tend
N_Vrs_to_Vinf {
  I={rel=comp_obl dpos=V finiteness=INF prep=to raise_from=subj
   raise_to=subj}
}

}
```

A.4 *lexicon* anglais

Listing A.4 – *leixcon* anglais

```
lexicon {

admit : "admit"
admonish : "admonish"
afford : "afford"
agree : "afford"
aim : "afford"
allow : "allow"
appear : "appear"
appear_2 : "appear_2"
appear_3 : "appear_3"
appear_4 : "appear_4"
appear_5 : "appear_5"
```

```
apply : "afford"
arrange : "afford"
ask : "ask"
attempt : "afford"
authorize : "admonish"
avoid : "avoid"
begin : "appear"
begin_2 : "begin_2"
begin_3 : "begin_3"
bother : "admit"
bribe : "admonish"
care : "afford"
catch : "catch"
caution : "admonish"
choose : "afford"
choose_2 : "admonish"
claim : "afford"
come : "afford"
commission : "admonish"
compel : "ask"
conspire : "afford"
continue : "appear"
continue_2 : "begin_2"
continue_3 : "begin_3"
convince : "admonish"
decide : "afford"
decline : "afford"
delay : "avoid"
deny : "avoid"
deserve : "afford"
determine : "afford"
direct : "admonish"
dispatch : "admonish"
elect : "afford"
empower : "admonish"
enable : "admonish"
encourage : "admonish"
```

```
end_up : "appear_2"
end_up_2 : "avoid"
enjoy : "avoid"
entice : "admonish"
expect : "afford"
expect_2 : "admonish"
fail : "appear"
fail_2 : "appear_3"
figure : "figure"
flock : "afford"
force : "admonish"
forget : "afford"
get_to : "afford"
go_to : "afford"
guarantee : "afford"
happen : "appear"
hate : "afford"
have_to : "afford"
help : "help"
hesitate : "afford"
hire : "admonish"
hope : "afford"
impel : "admonish"
induce : "admonish"
influence : "admonish"
inspire : "admonish"
intend : "afford"
invite : "admonish"
keep : "appear_2"
keep_2 : "keep_2"
keep_3 : "keep_3"
know : "afford"
lead : "admonish"
learn : "afford"
leave : "leave"
let : "let"
license : "admonish"
```

```
like : "admit"
like_2 : "admonish"
look : "afford"
love : "admit"
make : "keep_2"
make_2 : "make_2"
manage : "afford"
mean : "admit"
mind : "avoid"
miss : "avoid"
motivate : "afford"
move : "admonish"
need : "need"
need_2 : "admonish"
negotiate : "afford"
oblige : "afford"
offer : "afford"
opt : "afford"
order : "admonish"
ought : "appear_3"
permit : "admonish"
plan : "afford"
pledge : "ask"
plot : "afford"
prefer : "admit"
prefer_2 : "keep_2"
prefer_3 : "admonish"
prepare : "need"
press : "admonish"
proceed : "afford"
prod : "prod"
profess : "afford"
program : "admonish"
promise : "afford"
prompt : "ask"
propose : "admit"
prove : "appear_2"
```

```
prove_2 : "keep_2"
prove_3 : "appear_3"
push : "prod"
quit : "avoid"
refuse : "afford"
remain : "appear"
remain_2 : "appear_2"
remain_3 : "begin_2"
remain_4 : "appear_3"
report : "admonish"
report_2 : "keep_2"
require : "allow"
resolve : "afford"
risk : "avoid"
rush : "afford"
schedule : "afford"
scramble : "afford"
seek : "afford"
seem : "appear_2"
seem_2 : "seem_2"
seem_3 : "seem_3"
seem_4 : "begin_2"
seem_5 : "seem_5"
seem_6 : "seem_6"
seem_7 : "appear_4"
select : "afford"
sensitize : "admonish"
set_out : "afford"
sign : "sign"
slate : "admonish"
sound : "afford"
spur : "admonish"
stain : "keep_2"
stand : "avoid"
start : "appear"
start_2 : "begin_2"
start_3 : "begin_3"
```

```
stop : "appear"
stop_2 : "begin_2"
stop_3 : "appear_3"
stop_4 : "keep_3"
stop_5 : "stop_5"
strive : "afford"
struggle : "afford"
suffice : "afford"
swear : "afford"
take_occasion_to : "admonish"
teach : "admonish"
tell : "ask"
tend : "appear_3"
threaten : "afford"
trust : "admonish"
try : "admit"
undertake : "afford"
urge : "admonish"
use_to : "afford"
venture : "afford"
vote : "afford"
vow : "afford"
wait : "afford"
want : "afford"
want_2 : "want_2"
warn : "admonish"
warn_2 : "warn_2"
wish : "ask"
wish_2 : "admonish"
"admit": verb {
  gp = { id=N_V_Vger    dia=12}
  gp = { id=N_V_to_Vinf   dia=12}
}

"admonish": verb {
  gp = { id=N_V_N_to_Vinf   dia=123}
}
```

```
"afford": verb {
  gp = { id=N_V_to_Vinf   dia=12}
}

"allow": verb {
  gp = { id=N_V_N_to_Vinf   dia=123}
  gp = { id=N_V_Vger    dia=13}
  gp = { id=N_V_to_Vinf   dia=13}
}

"appear": verb {
  gp = { id=N_V   dia=1}
}

"appear_2": verb {
  gp = { id=N_V_ADJ   dia=12}
}

"appear_3": verb {
  gp = { id=N_Vrs_to_Vinf   dia=1}
}

"appear_4": verb {
  gp = { id=Exp_V_ADJ   dia=X1}
}

"appear_5": verb {
  gp = { id=Exp_V_that_Vind   dia=X1}
}

"ask": verb {
  gp = { id=N_V_N_to_Vinf   dia=123}
  gp = { id=N_V_to_Vinf   dia=12}
}

"avoid": verb {
```

```
  gp = { id=N_V_Vger    dia=12}
}

"begin_2": verb {
  gp = { id=N_V_N   dia=12}
}

"begin_3": verb {
  gp = { id=N_Vrs_Vger    dia=1}
  gp = { id=N_Vrs_to_Vinf   dia=1}
}

"catch": verb {
  gp = { id=N_V_N_Vger    dia=123}
}

"figure": verb {
  gp = { id=N_V_Vind    dia=12}
}

"help": verb {
  gp = { id=N_V_N_Vinf    dia=123}
  gp = { id=N_V_N_to_Vinf   dia=123}
  gp = { id=N_V_Vinf    dia=13}
  gp = { id=N_V_to_Vinf   dia=13}
}

"keep_2": verb {
  gp = { id=N_V_ADJ_N   dia=12}
}

"keep_3": verb {
  gp = { id=N_Vrs_Vger    dia=1}
}

"leave": verb {
  gp = { id=N_V_N_Vger    dia=123}
```

```
  gp = { id=N_V_N_to_Vinf   dia=123}
  gp = { id=N_V_to_Vinf   dia=12}
}

"let": verb {
  gp = { id=N_V_N_Vinf    dia=123}
  gp = { id=N_V_Vinf    dia=12}
}

"make_2": verb {
  gp = { id=N_V_N_Vinf    dia=123}
}

"need": verb {
  gp = { id=N_V_N_to_Vinf   dia=123}
  gp = { id=N_V_Vinf    dia=12}
  gp = { id=N_V_to_Vinf   dia=12}
}

"prepare": verb {
  gp = { id=N_V_Vinf    dia=12}
  gp = { id=N_V_to_Vinf   dia=12}
}

"prod": verb {
  gp = { id=N_V_N_into_Vger   dia=123}
  gp = { id=N_V_N_to_Vinf   dia=123}
}

"seem_2": verb {
  gp = { id=N_V_ADJ_to_N    dia=12}
}

"seem_3": verb {
  gp = { id=Exp_V_that_Vind_to_N    dia=X12}
  gp = { id=N_Vrs_Vinf_to_N   dia=12}
}
```

```
"seem_5": verb {
  gp = { id=Exp_V_that_Vind_to_N    dia=X12}
  gp = { id=N_V_N_to_N    dia=12}
}

"seem_6": verb {
  gp = { id=Exp_V_that_Vind   dia=X1}
  gp = { id=N_Vrs_to_Vinf   dia=1}
}

"sign": verb {
  gp = { id=N_V_to_Vinf_for_N   dia=123}
  gp = { id=N_V_to_Vinf_to_N    dia=123}
}

"stop_5": verb {
  gp = { id=N_Vrs_N_Vger    dia=1}
}

"take_2": verb {
  gp = { id=N_V_for_ADJ_N   dia=12}
}

"take_3": verb {
  gp = { id=N_V_N_N   dia=123}
}

"warn_2": verb {
  gp = { id=N_V_N_against_Vger    dia=123}
  gp = { id=N_V_against_Vger    dia=13}
}

}
```