Mémoire

Favoriser la réutilisation de corpus annotés : conversion vers le standard d'annotation Universal Dependencies

---

Par Clémence CAULE

Sous la direction de Aleksandra Miletic et Dejan Stosic

Département des Sciences du Langage

Master 1 – Linguistique, Informatique et Technologies du Langage

Année 2021-2022

Remerciements

Je tiens à adresser mes remerciements à mes deux directeurs de mémoire, Madame Aleksandra Miletic et Monsieur Dejan Stosic, pour leurs conseils et leur accompagnement tout au long de ce mémoire. Merci d'avoir pris le temps de m'expliquer ces nouvelles notions et de me guider dans mes recherches. Une attention particulière à Madame Aleksandra Miletic pour sa disponibilité et ses encouragements.

J'adresse mes remerciements à l'ensemble des enseignants du Master LITL, Madame Lydia-Mai Ho-Dac, Madame Cécile Fabre et Monsieur Ludovic Tanguy pour leur accompagnement durant l'année scolaire.

Je voudrais remercier ma famille pour leur soutien. Merci à mes parents de m'avoir donné cette chance de continuer mes études dans un domaine qui me plaît et merci à mes sœurs qui me soutiennent dans tout ce que j'entreprends.

Pour finir , je tiens à remercier mes amis. Merci Anaëlle et Wissam pour votre soutien infaillible, vous avez su trouver les bons mots quand j'en avais besoin. Merci Gabriel et Emma pour vos précieux conseils et votre présence. Merci Nikola, Clara et Mikołaj pour ces pauses qui m'ont permis de respirer et de continuer à avancer.

Table des matières

Introduction

Ce mémoire s'inscrit dans la suite d'un travail précédent (Miletic, 2018), qui a porté sur la création d'un premier treebank pour le serbe. Ce corpus, nommé ParCoTrain, est d'un intérêt spécial pour le TAL du fait que le serbe est une langue avec très peu de ressources libres. Ce corpus a été constitué dans le cadre du projet ParCoLab lancé en 2010 dont l'objectif est la constitution d'un corpus parallèle serbe-français-anglais.

En effet, lors de la conception du corpus ParCoTrain, il existait peu de corpus annotés pour le Serbe. De plus, une grande majorité de ressources n'est pas en libre accès donc la constitution de corpus de grande envergure est compliquée. Il est difficile de développer des outils automatiques pour le Serbe en TAL du fait de ce manque d'outils et de ressources préalables. Il est donc important de rendre accessible, pour le plus grand nombre possible, les ressources déjà existantes comme ParCoTrain.

Ainsi, nous cherchons, dans ce mémoire, à favoriser la réutilisation de corpus annotés. Pour se faire, nous proposerons une conversion des annotations de ParCoTrain vers le standard d'annotation Universal Dependencies à l'aide de l'outil préexistant Grew.

Grew est un outil basé sur un modèle computationnel de réécriture de graphes. A partir d'un ensemble de règles qui lui sont données, il peut effectuer automatiquement la conversion des annotations. C'est un outil de réécriture de graphes dédié aux applications en traitement automatique des langues. Il peut manipuler de nombreux types de représentation linguistique. Il a été utilisé sur la séquence étiquetée POS, la syntaxe de dépendance de surface, la syntaxe de dépendance profonde, la représentation sémantique et peut être utilisé pour représenter n'importe quelle structure basée sur des graphes.

Ce mémoire est structuré de la manière suivante. Nous commençons, dans la première partie, par présenter l'état de l'art des notions de notre sujet. Puis, dans le deuxième chapitre, nous présentons les données qui servent de support à notre travail. Nous poursuivons avec une troisième partie sur la méthodologie que nous avons mis en place afin d'effectuer la conversion de nos annotations. Dans le quatrième chapitre, nous présentons les résultats de

ces conversions et enfin, dans la dernière partie, nous discutons des pertes d'informations entraînées par la conversion et de la stratégie mise en place pour éviter cela.

Chapitre 1 - Recherches antérieures et ressources existantes pour la conversion d'annotations

Cet état de l'art va se concentrer sur la place du serbe dans le traitement automatique des langues ainsi que sur les outils et les méthodologies existantes pour la conversion d'annotations.

1. Le serbe dans le TAL

1.1. Spécificités linguistiques

Le serbe est une langue slave qui est très difficile pour le TAL. En effet, c'est une langue à morphologie flexionnelle riche, à ordre des constituants flexibles et qui ne comporte pas d'article. Le serbe utilise deux alphabets : le latin et le cyrillique. C'est également une langue pro-drop ce qui veut dire que la réalisation du sujet dans la phrase est facultative. Miletic (2018), dans sa thèse, dresse un profil linguistique du serbe très détaillé et dont nous relèverons, ici, les principales caractéristiques.

D'un point de vue morphologique, le serbe compte 7 cas dans son système de déclinaisons nominales : le nominatif, le génitif, le datif, l'accusatif, le vocatif, l'instrumental et le locatif. Si l'on ajoute à cela les marques de nombre (singulier ou pluriel), un nom peut avoir 14 formes différentes.

Les adjectifs portent les marques de genre (féminin, masculin ou neutre), de cas (cités précédemment), de nombre, de comparaison (comparatif, positif ou superlatif) et de définitude (défini ou indéfini). Un adjectif peut ainsi apparaître sous 126 formes différentes.

Au niveau de la conjugaison, le serbe prend en compte 9 temps. Chacune des formes porte les marques de nombre et de personne (première, deuxième ou troisième). Les participes passés portent également des marques de genre. Un verbe peut donc avoir plus de 120 formes fléchies.

Cette richesse morphologique pose problème dans le traitement du serbe en TAL car elle provoque des ambiguïtés dans les formes fléchies. En effet, on observe un haut taux de syncrétisme au niveau des déclinaisons notamment pour les adjectifs.

D'un point de vue syntaxique, le serbe est classé comme une langue SVO c'est-à-dire qu'elle suit l'ordre sujet-verbe-objet. Cependant, c'est aussi une langue libre dans l'ordre de ses constituants. En effet, selon le mot que le locuteur veut mettre en emphase, il le positionne à différentes places dans la phrase. De ce fait, il existe 5 variations de construction syntaxique en plus de l'ordre SVO. En outre, les structures discontinues sont autorisées en serbe. Ainsi, pour retrouver les gouverneurs ou les fonctions syntaxiques d'un mot, nous devons vérifier les traits morphologiques de ces derniers.

Toutes ces spécificités syntaxiques posent des problèmes dans le traitement automatique du serbe. En effet, l'analyse syntaxique ne relève pas seulement de la syntaxe mais doit aussi prendre en compte les traits morphologiques puisque l'ordre des mots est très variable. La diversité morphosyntaxique induit une dispersion des données. Il est rare de trouver dans un corpus toutes les occurrences des variations formelles possibles, d'où la difficulté pour les outils automatiques d'apprendre à les maîtriser.

Il faut également noter que ces difficultés intrinsèques à la langue semblent couplées à un manque de volonté, de moyens et/ou de ressources pour la création des outils du TAL. Le tchèque, qui exhibe des propriétés linguistiques similaires à priori difficiles pour le TAL, est une langue bien dotée.

1.2. Les ressources existantes pour le serbe

La deuxième moitié du 20ème siècle est marquée par le développement de ressources dans certaines langues. On peut diviser ces ressources en trois catégories : les ressources lexicales, les corpus et les outils du TAL. Dans les ressources lexicales, nous pouvons trouver des dictionnaires comme SrpMD (Krstev) qui est un dictionnaire morphologique serbe. Dans le deuxième type de ressources, se développent des corpus de grande envergure comme COCA (Davies, 2010) pour l'anglais ou SrpKor (Krstev et Vitas, 2005) qui est un corpus du serbe contemporain lemmatisé et étiqueté en parties du discours. Ce dernier a été annoté

automatiquement sans correction manuelle, ce qui, comme le soulève Miletic (2018), ne représente pas une base idéale pour l'apprentissage et l'évaluation des outils automatiques. MULTEXT-East (Krstev et al., 2004) qui est un recueil de données multilingue pour la recherche et le développement en ingénierie linguistique dans plusieurs langues de l'Est dont le serbe est lui adapté à l'évaluation d'outils du TAL mais ces corpus restent trop peu nombreux (Miletic, 2018). En ce qui concerne les outils de traitement automatique des langues, on retrouve principalement des étiqueteurs qu'ils soient pour l'annotation morphosyntaxique, la lemmatisation ou l'annotation syntaxique. En serbe, nous pouvons citer, par exemple, AlfaNum Tagger (Sečujski et al., 2002) qui est un étiqueteur morpho-syntaxique serbe à base de règles et Btagger (Gesmundo et Samardzic, 2012) qui est un étiqueteur et lemmatiseur par apprentissage automatique. Malgré toutes ces ressources à disposition, la création d'un treebank pour le serbe est une entreprise récente (Jakovljević et al., 2014).

En effet, du fait de la dispersion de données, de larges corpus sont nécessaires afin de pouvoir étudier toutes les occurrences des phénomènes existants pour des langues comme le serbe. Cependant, la complexité de son annotation est un frein à la constitution de ressources. De plus, certaines ressources existantes ne sont pas en libre accès, ce qui ralentit considérablement le développement d'outils automatiques pour le TAL serbe. C'est le cas des travaux de Vitas & Krstev (2004) qui, d'après nos recherches, n'ont pas mis à disposition leurs corpus de travail.

L'intérêt d'avoir des ressources en serbe est, tout d'abord, de développer des outils automatiques pour le traitement du serbe. En effet, comme évoqué précédemment, le serbe a besoin de corpus de grande envergure pour que des outils automatiques soient pertinents. De plus, l'existence de corpus et d'outils ouvre la voie aux analyses linguistiques à grande échelle, fondées sur des données attestées.

Plusieurs ressources disponibles en serbe ont été créées grâce à celles disponibles en croate. En effet, ces deux langues sont quasiment identiques (cf. Chapitre 1, partie 1.3.). ParCoTrain (Miletic, 2018) (cf. Chapitre 2), le corpus sur lequel nous travaillons, est une des nouvelles ressources qui a pu être constituée grâce aux outils développés pour le croate.

1.3. Le croate, une langue proche du serbe

Le croate, comme le bosniaque, le monténégrin et le serbe, est une langue slave. Ces quatre langues étaient auparavant considérées comme une seule et même langue en Yougoslavie (qui n'existe plus de nos jours). C'est pourquoi celles-ci étaient souvent regroupées sous l'appellation serbo-croate ou BCMS (abréviation de bosnien, croate, monténégrin et serbe). Cependant, elles sont aujourd'hui officiellement considérées comme quatre langues distinctes.

Certains linguistes, comme Thomas (1994) et Miletic (2018), considèrent les différences entre ces langues comme des variations présentes dans des dialectes d'une même langue. En effet, ces quatre langues sont similaires au niveau phonologique, morphologique, syntaxique et lexical.

Ainsi, le croate qui est mieux doté en TAL est un avantage considérable pour la création de nouvelles ressources et outils pour le TAL serbe. C'est notamment grâce à cette situation particulière, et grâce à la libre diffusion des ressources et données de la communauté du TAL croate que la création d'un corpus arboré pour le serbe a été possible (Miletic et al. 2019). Les ressources du croate ont également permis la création d'un treebank UD pour le serbe (cf. Chapitre 1, partie 2.2.) ce qui donne la possibilité d'avancer sur les recherches en linguistique et en TAL pour le serbe.

2. Universal Dependencies (UD)

Dans les ressources disponibles en TAL, Universal Dependencies est devenu incontournable lors de l'annotation. En effet, comme nous allons le voir dans les parties suivantes, ce format est très utilisé et est donc une bonne piste pour favoriser la réutilisation de corpus annotés comme le nôtre.

2.1. Présentation générale d'Universal Dependencies

Universal Dependencies (UD) est un projet international lancé dans le but de réduire au minimum la variation interlinguistique dans les annotations de la structure syntaxique. Ce format d'annotation est devenu un standard de facto qui favorise la comparabilité entre les

langues et la réutilisation des ressources annotées avec ce système. UD a permis de créer des banques d'arbres pour plus de 100 langues et met l'accent sur des représentations de surface simples qui autorisent le parallélisme entre des constructions similaires de différentes langues proches comme le serbe et le croate (cf. Chapitre 1, partie 1.3.) mais également entre des langues bien plus différentes, comme le japonais et le grec comme on peut le voir sur le site d'Universal Dependencies.

La philosophie générale d'UD est de fournir un inventaire universel de catégories et de lignes directrices afin de faciliter l'annotation cohérente de constructions similaires dans toutes les langues, tout en permettant des extensions spécifiques à la langue si nécessaire.

La figure 1 (extraite du site internet[1] UD) illustre le parallélisme que l'on peut faire entre les langues. Ici, on voit que pour l'anglais, le bulgare, le tchèque et le suédois, les principales relations grammaticales impliquant un verbe passif, un sujet nominal et un agent oblique sont les mêmes, mais que la réalisation grammaticale concrète varie.

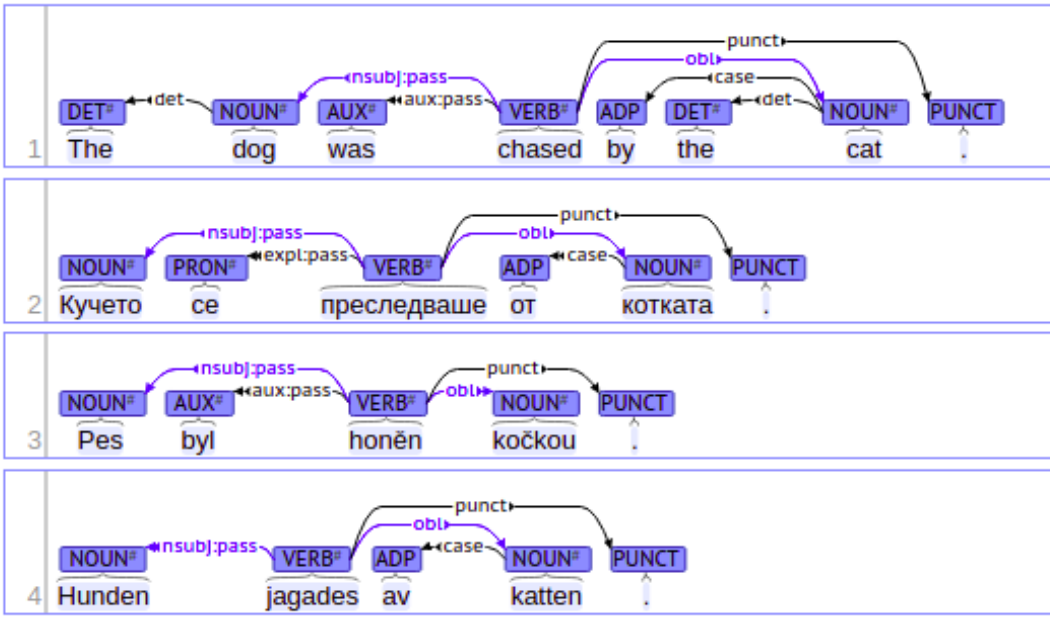

Fig. 1 : Analyse UD illustrant le parallélisme entre les langues

Les annotations UD sont basées sur un point de vue lexical de la syntaxe, ce qui signifie que les relations de dépendance s'établissent entre les mots. Par conséquent, les caractéristiques morphologiques sont codées en tant que propriétés des mots et il n'y a

[1] https://universaldependencies.org/introduction.html

aucune tentative de segmentation des mots en morphèmes. Cependant, il est important de noter que l'unité de base de l'annotation est constituée de mots syntaxiques (et non pas de mots phonologiques ou orthographiques). Par exemple, en français, la préposition et l'article ou le pronom doivent être systématiquement séparés, c'est-à-dire que *du* sera séparé en *de* + *le*. De plus, le fait qu'UD favorise les mots lexicaux aux mots grammaticaux pour établir les têtes des constituants syntaxiques, permet de maximiser le parallélisme entre les langues car les mots lexicaux varient moins que les mots grammaticaux entre les langues comme on peut le voir dans la figure 1.

Universal Dependencies ne cesse d'être amélioré. En effet, de nouveaux treebank sont ajoutés pour des langues qui n'en avaient pas et améliorés pour ceux déjà existants. Une amélioration majeure a été réalisée en 2020 permettant de passer de la version 1 de UD à la version 2. Cette nouvelle version d'UD proposée par Nivre et al. (2020) a permis d'améliorer la tokenisation, l'annotation morphologique, l'annotation syntaxique et les relations de dépendances.

2.2. Universal Dependencies en serbe

Agić et Ljubešić (2015) ont défini un schéma d'annotation UD, en détaillant les règles d'utilisation d'étiquettes pour le croate et en précisant qu'il pouvait également être utilisé pour le serbe du fait de leurs similarités linguistiques (cf. Chapitre 1, partie 1.3.). Dans la suite de leur travail, un jeu d'étiquettes UD pour le serbe a été créé en suivant les méthodes et les outils utilisés pour le croate (Samardžić et al., 2017).

Samardžić et al. (2017) ont suivi une méthode appelée *conversion automatique avec corrections manuelles* dans la documentation UD. Leur approche consiste en quatre étapes : 1) transfert automatique de l'annotation croate en serbe, 2) comparaison et adaptation, 3) conversion et correction automatiques, 4) correction manuelle.

Ainsi, dans un premier temps, le treebank serbe se compose de phrases qui sont alignées sur les phrases croates. Puis, à l'aide d'outils automatiques et d'analyse manuelle, des traits morphosyntaxiques ont été ajoutés pour permettre une annotation plus complète. Samardžić et al (2017) ont ensuite décidé quelles catégories et relations étaient à garder de l'UD croate pour créer l'UD serbe.

Premièrement, ils ont converti les relations de UDv1 vers UDv2 puis ils ont appliqué plusieurs changements. Ils ont notamment modifié les annotations existantes afin d'améliorer la cohérence des annotations et de résoudre des problèmes énoncés par la communauté UD. Ces améliorations permettent par exemple de faire la distinction entre les arguments essentiels (anglais : *core arguments*), qui représentent principalement les sujets et les objets, et tous les autres dépendants.

Le développement de ce nouveau treebank pour le serbe montre qu'UD permet d'améliorer les annotations déjà existantes en utilisant des analyses de parallélisme et croisement linguistique. Ces méthodes permettent de réduire le coût de développement de nouveaux treebanks, ce qui représente un avantage considérable, particulièrement pour les langues slaves comme le serbe. En effet, utiliser le format Universal Dependencies lors de l'annotation morpho-syntaxique et syntaxique d'un corpus serbe permet non seulement l'analyse linguistique détaillée de cette langue, d'en assurer la comparabilité avec d'autres langues mais également la possibilité d'améliorer les outils automatiques pour le serbe et pour les autres langues slaves.

3. Surface-syntactic Universal Dependencies (SUD)

Bien que UD soit très utilisé dans le monde de l'annotation, il ne suit pas les mêmes règles de distribution que notre corpus. En effet, UD est adapté à l'annotation de la syntaxe profonde. Or, notre corpus suit les règles de distribution de la syntaxe de surface. C'est pour cela que nous nous intéressons à Surface-syntactic Universal Dependencies qui est adapté à la syntaxe de surface et qui nous permet de faire une passerelle entre nos annotations de ParCoTrain et le format UD.

3.1. Présentation générale de SUD

SUD a été créé dans une volonté d'élaborer un schéma d'annotation restant aussi proche que possible d'UD mais en appliquant les critères de la syntaxe de surface. Ainsi, le critère principal de distribution en SUD est que "la tête syntaxique de surface détermine la distribution de l'unité" (Gerdes et al., 2018).

Il y a donc deux différences principales entre UD et SUD. Premièrement, dans SUD, les mots grammaticaux tels que les adpositions, les conjonctions de subordination, les auxiliaires et les copules sont des têtes. Deuxièmement, les mots qui sont dans le même paradigme de commutation, c'est-à-dire les mots qui ont la même position syntaxique et la même fonction grammaticale, sont connectés à leurs gouverneurs par la même relation syntaxique. Ce système d'annotation est intéressant pour le présent travail car les critères de distribution sont les mêmes que dans le corpus ParCoTrain (cf. Chapitre 2).

SUD suit le principe de minimisation de la longueur des dépendances (Gerdes et al., 2019) qui peut être défini comme le nombre de mots séparant le gouverneur de son dépendant. En SUD, les mots grammaticaux peuvent être des têtes, ce qui n'est pas possible avec UD. Cette particularité de SUD nous intéresse fortement dans notre travail puisque le corpus dont nous voulons convertir les annotations suit ce même principe. Nous allons donc utiliser SUD comme une passerelle entre les annotations de ParCoTrain et le jeu d'étiquettes d'UD.

La caractérisation des relations en SUD est basée sur le paradigme entier des éléments qui peuvent permuter dans une position de dépendance. Toutes ces relations sont hiérarchisées. Ainsi, les propriétés spécifiques des relations filles s'ajoutent aux propriétés plus générales héritées de la relation mère.

Certaines étiquettes de UD ont été remplacées par d'autres en SUD. En effet, 17 étiquettes de relations ont été remplacées par seulement 3 relations en SUD qui sont *suj, comp, mod* (*sujet*, *complément*, *modifieur*) dont les équivalences sont montrées dans la figure 2. Ce choix est justifié par un critère sémantique énoncé par Gerdes et al. (2019) : un argument d'une unité lexicale L est un participant obligatoire à la description sémantique de L.

Même s'il est possible d'ajouter des sous-relations aux étiquettes, cette réduction nous a posé problème lors de la conversion de nos annotations SUD en UD, notamment concernant les relations étiquetées *mod*. On s'interrogera sur le degré de la perte d'information entraînée

par ces choix (cf, Chapitre 5). Le tableau suivant (cf. Fig. 2) présente les équivalences entre les relations UD et SUD existantes qui sont présentées sur le site internet[2] de SUD.

UD	SUD
nsubj	subj
csubj	
aux	comp:aux
cop	comp:pred
xcomp	comp:obj
case	
mark	
obj	
ccomp	
det	det
nummod	

UD	SUD
ccomp	comp:obl
obl	
iobj	
nmod	udep
obl, acl	mod
advcl	
advmod	
amod	
nummod	
fixed	encoded in node features

Fig 2: Tableau d'équivalence UD-SUD

4. Conversion de SUD vers UD

Si, dans un premier temps, nous convertissons nos étiquettes de ParCoTrain vers SUD de part leurs ressemblances dans la distribution syntaxique, nous avons besoin de convertir nos étiquettes SUD en UD. En effet, comme nous l'avons vu précédemment, UD est le meilleur candidat pour favoriser la réutilisation d'un corpus annoté de part son utilisation massive permettant la comparabilité entre les langues. Ainsi, nous avons trouvé une méthode et des outils de conversion comme Grew Arborator qui ont déjà été utilisés par des chercheurs dans le cadre de transformation de SUD en UD et inversement, comme nous allons le voir dans les parties suivantes.

[2] https://surfacesyntacticud.github.io/conversions/

4.1. La méthode déjà existante

La méthode pour convertir des éléments de SUD vers UD et inversement contient trois grandes étapes (Gerdes et al., 2018) qui s'appliquent à des cas précis de conversion d'UD vers SUD ou inversement. Celles-ci sont : 1) la transformation de la structure de l'élément, 2) le remplacement des étiquettes par de nouvelles, 3) un appariement des deux éléments en SUD et UD. Nous présentons ici deux conversions qui ont déjà été réalisées.

La première conversion (Gerdes et al., 2018) avait pour but de vérifier et d'adapter les étiquettes SUD créées. En effet, il fallait s'assurer que les conversions puissent se faire entre les deux jeux d'étiquettes afin de pouvoir faciliter le passage de l'un à l'autre. La première étape consiste en la transformation des structures "en bouquet" en analyse en cascade pour certaines relations (*conj, fixed et flat*). Ensuite, il faut inverser les relations *aux, cop, mark* et *case* tout en veillant à respecter les relations de dépendances entre les mots. En effet, la hiérarchie des relations doit être respectée et si les informations données ne permettent pas de vérifier cet ordre, le mot le plus proche est considéré comme le gouverneur de la tête lexicale et le suivant, le gouverneur du premier et ainsi de suite. Troisièmement, il faut mettre en correspondance les relations UD et SUD. Cette troisième étape a permis de révéler des problèmes dans leur grammaire de conversion qui ont été corrigés lors de l'amélioration de SUD (Gerdes et al., 2019).

La deuxième conversion visait à transformer des éléments de SUD vers UD (Gerdes et al., 2019). La première étape est la transformation de la chaîne pour la relation *conj* en une structure "en bouquet". Il faut ensuite remplacer les relations *comp:aux*, et *comp:pred* avec un gouverneur *AUX*, et *comp:obj* avec un gouverneur *ADP*, *SCONJ* ou *PART* (ce qui donne *aux, cop, mark et case*). Enfin, il faut mettre en correspondance les relations SUD avec les relations UD.

Ces conversions ont été écrites à l'aide de requêtes Grew qui est un outil de réécriture présenté dans la partie suivante.

4.2. Grew

Grew (Guillaume et al., 2012) est un outil de recherche et de réécriture de graphes spécialisé dans les structures nécessaires au TAL, c'est-à-dire les arbres et graphes de dépendances syntaxiques et sémantiques. Grew peut être utilisé via le terminal sous linux et mac en téléchargeant des packages. Cependant, il existe aussi une version en ligne, Grew-match, où tous les treebanks d'Universal Dependencies dans les différentes syntaxes (classique, profonde et de surface) peuvent être interrogées. En effet, comme cela est expliqué sur la page de présentation de Grew[3], il peut être utilisé sur des séquences étiquetées en POS, en syntaxe de dépendance de surface, en syntaxe de dépendance profonde, en représentation sémantique (AMR, DMRS) mais il permet également de représenter n'importe quelle structure basée sur des graphes.

Grew est un outil basé sur un modèle computationnel de réécriture de graphes, Graph Rewriting System (GRS), qui est un système de règles organisées en modules. "Le logiciel Grew permet de définir un GRS et de l'appliquer à des graphes" (Guillaume et al., 2012). En effet, un script Grew, nommé *grs*, est constitué d'un ensemble de règles et d'une stratégie décrivant comment ces règles d'application doivent être appliquées. Plusieurs paramètres sont définis par Guillaume et al. (2012) dans un système de réécriture de graphes : les graphes, les règles, les modules et les interfaces. Nous en résumons ici les principales caractéristiques.

Les graphes (=*GRS*) sont composés de nœuds contenant des structures de traits et d'un ensemble d'arcs étiquetés codant les relations de dépendances (ou autres relations). Il ne peut y avoir qu'un seul arc par étiquette.

Les règles de réécriture (=*rule*) qui servent à manipuler les graphes sont composées de 3 parties. Le patron positif (=*pattern*) qui est un graphe qu'on cherche à mettre en paire avec le graphe à réécrire ; les patrons négatifs (=*without*) non obligatoires pouvant bloquer des règles ; un ensemble de commandes (=*commands*) décrivant les modifications apportées au graphe et qui permettent d'ajouter, de modifier ou de supprimer des traits, des nœuds et des arcs.

[3] https://grew.fr/

Les modules (=*package*), qui regroupent les différentes règles, permettent de contrôler l'application des graphes et de simplifier le développement du système. Les modules sont ordonnés et la réécriture du système se fait en considérant la sortie d'un module comme entrée du module suivant.

Les interfaces graphiques (=*GUI*) permettent de visualiser les différentes étapes de la réécriture. De plus, elles permettent d'obtenir des statistiques de calcul et de fréquences d'utilisation de modules ou règles. Cependant, ces interfaces ne peuvent plus être utilisées avec la dernière version de Grew.

4.3. Arborator-Grew

Arborator-Grew (Guibon et al., 2020) est un outil d'annotation collaboratif pour le développement de banques d'arbres. Arborator-Grew combine les fonctionnalités de deux outils préexistants : Arborator et Grew. Arborator est un outil collaboratif d'annotation d'arbre de dépendance en ligne. Arborator-Grew est une modernisation d'Arborator, en effet, son stockage de base de données interne a été remplacé par l'interface de programmation Grew. Ceci permet d'ajouter un puissant outil de requête aux fonctionnalités déjà existantes d'Arborator, la création et la correction de treebank. "Arborator-Grew ouvrent de nouvelles voies pour créer, mettre à jour, maintenir et conserver collectivement des treebanks syntaxiques et des banques de graphes sémantiques" (Guibon et al., 2020).

Dans ce premier chapitre, nous venons d'étudier les ressources et les outils existants pour l'annotation et la conversion du serbe. Nous avons pris conscience de l'état actuel des connaissances concernant notre domaine de recherches. Ainsi, le serbe est une langue avec peu de ressources libres disponibles pour l'entraînement d'outils automatiques spécialisés dans cette langue, ce qui représente un frein dans la création et le développement de ceux-ci. Notre corpus de travail, ParCoTrain que nous nous appliquerons à présenter dans le prochain chapitre représente donc une ressource nécessaire et utile dans ce domaine. C'est pour cela que sa conversion vers le format UD, qui est très répandu dans le monde du TAL, est indispensable pour favoriser sa réutilisation.

Chapitre 2 - Les données : ParCoTrain

Dans ce chapitre, nous proposons une description de nos données et par conséquent du travail de Miletic (2018) puisque notre mémoire s'inscrit dans la continuité de son travail.

1. Présentation générale

ParCoTrain[4] est un corpus contenant 101 425 tokens dont l'initiative de constitution a été lancée par Aleksandra Miletic dans le cadre de sa thèse, *Un treebank pour le serbe : constitution et exploitations* (2018). Ce travail de thèse a été proposé par Dejan Stosic et encadré par Dejan Stosic et Cécile Fabre.

Le texte du corpus provient de deux ouvrages littéraires serbe: Bašta, pepeo de D. Kiš de 55 783 tokens et Testament de V. Stevanovic de 45 642 tokens. Les deux textes ont été étiquetés selon la syntaxe en dépendances mais seul le deuxième a été lemmatisé. La lemmatisation permet de réduire la dispersion des données. L'annotation morphosyntaxique a été réalisée dans le but d'optimiser les conditions d'une analyse syntaxique. Le serbe étant une langue à morphologie flexionnelle riche à ordre des constituants flexibles, ce type d'annotation permet de faciliter l'analyse syntaxique.

Un jeu d'étiquettes spécifique à ce corpus a été créé selon des critères précis (cf. Chapitre 2, partie 2.2). Il y a 1042 étiquettes pour les traits morphologiques des parties du discours, 48 étiquettes syntaxiques complétées d'un traitement spécifique de l'ellipse. Un guide d'annotation a également été créé afin de faciliter la compréhension des participants à la campagne d'annotation et des personnes souhaitant utiliser le corpus dans leurs recherches ou autres travaux.

Le corpus sur lequel nous travaillons est constitué de 3 fichiers au format CoNLL-X qui s'intitulent synt_dev, synt_test et synt_train. Le format CoNLL provient d'une tâche partagée proposée lors de la Conférence sur l'apprentissage informatique des langues naturelles. Dans le format CoNLL-X, qui est la dixième version du CoNLL, "toutes les

[4] Le corpus peut être téléchargé à partir du lien suivant : https://github.com/aleksandra-miletic/serbian-nlp-resources

phrases sont dans un fichier texte et elles sont séparées par une ligne blanche après chaque phrase. Une phrase est composée d'un ou plusieurs jetons. Chaque jeton est représenté sur une ligne, composée de 10 champs. Les champs sont séparés les uns des autres par une tabulation. Les 10 champs du format CoNLL-X présentés par Buchholz et Marsi (2006) sont les suivants :

1) ID: Le compteur de token qui recommence à 1 pour chaque nouvelle phrase.
2) FORM: La forme du token ou le signe de ponctuation.
3) LEMMA: Le lemme du token.
4) CPOSTAG: L'étiquette de catégorie grammaticale à gros grain
5) POSTAG: L'étiquette de catégorie grammaticale détaillée ou ""à grain fin"
6) FEATS: Les traits morphologiques et syntaxiques du token.
7) HEAD: L'ID du gouverneur du token.
8) DEPREL: La relation de dépendance qui lie le token à son gouverneur (HEAD).
9) PHEAD: La tête projective du token actuel
10) PDEPREL: La relation de dépendance avec PHEAD

2. Annotations de ParCoTrain

2.1. Méthode d'annotation

L'annotation repose sur un bootstrapping itératif multicouche. Cette méthode basée sur la répétition commence par un entraînement sur un ensemble de phrases issues du corpus ParCoTrain. L'annotation suit les étapes suivantes : étiquetage morphosyntaxique, lemmatisation, analyse syntaxique. Une correction manuelle est effectuée à la sortie de chaque outil ce qui assure une meilleure fiabilité dans les données d'entrée et permet de maximiser les performances des outils et ainsi de faciliter le travail des annotateurs humains.

Un travail d'harmonisation des annotations a été réalisé à la fin de l'annotation afin d'éviter les incohérences qui auraient pu être causées par le bootstrapping.

2.2. Propriétés des annotations

L'annotation syntaxique a été faite en respectant trois principes. Tout d'abord, le jeu d'étiquettes a été constitué spécialement pour ce corpus et est donc adapté spécifiquement au serbe. Deuxièmement, les étiquettes se veulent informatives et pertinentes sur le plan linguistique, mais basées sur des critères de surface afin qu'elles soient accessibles à un parser. Enfin, l'annotation syntaxique suit les principes de la syntaxe en dépendances.

L'annotation morphosyntaxique a également été faite en respectant trois principes. Premièrement, elle contient aussi bien les parties du discours que des traits morphosyntaxiques précis. Deuxièmement, seuls les traits morphosyntaxiques pertinents pour le parsing ont été encodés. Enfin, dans le corpus final, l'annotation morphosyntaxique est décomposée en plusieurs couches (étiquettes POS, étiquettes détaillées, traits individuels), ce qui permet de faciliter la sélection de la couche qui convient dans le cadre du parsing.

Le corpus a été lemmatisé afin de diminuer la dispersion des données et un lexique à large couverture a été intégré pour faciliter le traitement automatique.

Dans ce deuxième chapitre, nous nous sommes familiarisés avec nos données et l'annotation de notre corpus. Nous allons donc devoir convertir les étiquettes morpho-syntaxiques, les traits morphologiques et les relations syntaxiques vers le format SUD dans un premier temps puisque celui-ci se rapprochent de ParCoTrain au niveau du fonctionnement de la distribution syntaxique. Puis dans un deuxième temps, nous allons convertir nos annotations vers le standard Universal Dependencies. Nous nous appliquerons à convertir le maximum des annotations mais nous prévoyons de ne pas pouvoir toutes les convertir du fait de la quantité de celles-ci. Dans le prochain chapitre, nous expliciterons la méthodologie que nous avons employée pour mener à bien l'objectif de ce mémoire.

Chapitre 3 - Méthodologie

Dans ce chapitre, nous présentons la méthodologie utilisée pour mener à bien la conversion. Ainsi, dans un premier temps, nous avons préparé nos données en dressant manuellement les tableaux d'équivalences des étiquettes dans les trois formats (ParCoTrain, SUD, UD). Puis, nous décrivons le traitement des données que nous avons effectué dans les différents outils que nous avons utilisés (Grew et Arborator-Grew).

1. Préparation des données

1.1. Equivalences des étiquettes ParCoTrain-SUD-UD

Avant de convertir automatiquement les étiquettes, nous avons, dans un premier temps, dressé le tableau des correspondances entre les étiquettes morpho-syntaxiques de ParCoTrain et d'Universal Dependencies. En effet, comme vu précédemment, SUD utilise les POS tags d'UD. Ce tableau (cf. Fig. 3) regroupe les catégories et les sous-catégories (en français et en serbe) des étiquettes morpho-syntaxiques présentes dans ParCoTrain et leurs étiquettes d'Universal Dependencies correspondantes.

Catégorie	Sous-catégorie	ParCoTrain	UD
Nom	nom propre *vlastite*	N_prop	PROPN
	nom commun *zajednicke*	N_com	NOUN
	massif *gradivne*		
	abstrait *apstraktne*		
	déverbaux *glagolske*		
	collectif *zbirne*	N_col	
Numéraux	généraux *opsti*	Num_card	NUM
	ordinaux *redni*	Num_ord	
	collectifs *zbirni*	Num_col	
Adjectif	qualificatif *opsini*	A_qual	ADJ
	démonstratif *pokazni*	A_dem	DET
	possessif *prisvojni*	A_pos	
	indéfini *neodredjeni*	A_indef	

Catégorie	Sous-catégorie	ParCoTrain	UD
	interrogatif *upitni*	A_inter	
	relatif *odnosni*	A_rel	
Verbe	principal *glavni*	V_main	VERB
	auxiliaire *pomocni*	V_aux	AUX
Pronom	personnel *licna*	P_pers	PRON
	démonstratif *pokazna*	P_dem	
	indéfini *neodredjena*	P_indef	
	numéral *brojna*	P_num	
	possessif *prisvojna*	P_pos	
	réflexif *povratna*	P_ref	
	interrogatif *upitna*	P_inter	
	relatif *odnosna*	P_rel	
Conjonctions	subordination *subrdinirani*	C_sub	SCONJ

Catégorie	Sous-catégorie	ParCoTrain	UD
	coordination *koordinirani*	C_coord	CCONJ
Adverbe	généraux *opsti*	Adv_gen	ADV
	relatif *odnosni*	Adv_rel	
	interrogatif *uptini*	Adv_inter	
	indéfini *neodredjeni*	Adv_indef	
Particule	-	Part	PART
Ponctuation	-	Z	PUNCT
Autres	-	X	X
Préposition/Adposition	-	Prep	ADP
Symbole	-	traités comme le nom du symbole : % = pourcent	SYM
Interjection	-	I	INTJ
Lettres isolées	-	L	traités comme nom commun : NOUN

Fig. 3 : Tableau d'équivalences ParCoTrain-UD des étiquettes morpho-syntaxiques

Les équivalences des étiquettes morpho-syntaxiques n'ont pas été problématiques à trouver. Cependant, plusieurs remarques sont à relever.

Les lettres isolées n'ont pas d'étiquette propre dans UD. Nous avons choisi de leur attribuer l'étiquette de nom commun (*NOUN*).

Les symboles étant traités, dans ParCoTrain comme le nom du symbole qu'il représente, nous n'avons pas pu leur attribuer l'étiquette UD correspondante (*SYM*). Ils seront donc traités comme des noms communs dans UD (*NOUN*).

Enfin, selon la grammaire traditionnelle, il n'existe pas de déterminants à proprement parler en serbe. Ainsi, dans ParCoTrain, les déterminants sont traités comme des adjectifs. Les adjectifs équivalents à nos adjectifs français sont regroupés dans la sous-catégorie *qualificatifs*.

Dans un deuxième temps, nous avons dressé le tableau des correspondances entre les traits morphologiques de ParCoTrain et ceux d'Universal Dependencies (cf. Fig. 4).

Trait	Nom	ParCoTrain		UD	
Genre	Féminin zenski rod	g =	f	Gender =	Fem
	Masculin muski rod		m		Masc
	Neutre srednji rod		n		Neut
Nombre	Singulier jednina	n =	sg	Number =	Sing
	Pluriel mnozina		pl		Plur
Cas	nominatif nominativ	c =	nom	Case =	Nom
	génitif		gen		Gen

Trait	Nom	ParCoTrain		UD	
	genitiv				
	datif dativ		dat		Dat
	accusatif akuzativ		acc		Acc
	vocatif vokativ		voc		Voc
	instrumental		ins		Ins
	locatif lokativ		loc		Loc
Degré de comparaison	positif pozitiv	_	pos	Degree =	Pos
	comparatif komparativ		comp		Cmp
	superlatif superlativ		sup		Sup
Négation	négation negiran	_	neg	Polarity =	Neg
Personne	1ère personne prvo lice	r =	1	Person =	1
	2ème personne drugo lice		2		2
	3ème personne trece lice		3		3

Fig. 4 : Tableau d'équivalences des traits morphologiques

Le degré de comparaison et la négation ne sont pas présents dans la colonne des features des fichiers CoNLL-X de ParCoTrain. En effet, ils sont uniquement présents dans la colonne des xpos.

Nous avons réalisé un tableau à part pour les traits morphologiques des verbes car ils possèdent de nombreux traits morphologiques propres à leur catégorie (cf. Fig.5).

UD						
VerbForm =	Mood =	Tense =	Voice =	Serbe	Français	ParCoTrain t =
Fin	Imp	-	-	imperativ	impératif	imper
	Ind	Pres	-	prezent	présent	pres
		Past	-	aorist	aoriste	aor
		Fut	-	futur	futur	fut
		Imp	-	imperfekat	imparfait	impf
Conv	-	Pres	-	particip sadasnji	participe présent	partpres
		Past	-	particip prosli	participe passé	partpass
Part	-	-	Act	particip radni	participe actif	partact
		-	Pass	particip trpni	participe passif	partpast
Inf	-	-	-	infinitiv	infinitif	inf

Fig. 5 : Tableau d'équivalences des traits morphologiques des verbes

Pour finir, nous avons dressé le tableau des correspondances pour les étiquettes des relations syntaxiques (cf. Fig. 6). Nous présenterons ici un extrait du tableau des équivalences. Le tableau complet des équivalences des relations syntaxiques et de leurs structures est disponible dans les annexes (cf. Annexe 1). Nous avons choisi de représenter la structure des étiquettes sous le format A-[relation]→B qui est le format utilisé par Grew pour représenter les relations de dépendances.

	ParCoTrain		SUD		UD	
Définition	Étiquette	Structure	Étiquette	Structure	Étiquette	Structure
complément de préposition	ComplPrep	ADP-[ComplPrep]→NOUN	comp :obj	ADP-[comp:obj]→NOUN	case	NOUN- [case]→ ADP

Fig. 6 : Tableau d'équivalences de la relation Complément de préposition

Nous avons regroupé dans ce tableau la définition de l'étiquette, ici *Complément de préposition*, ainsi que l'étiquette syntaxique et la structure de chaque système d'annotation : ParCoTrain, SUD et UD. La colonne *étiquette* indique le nom de l'étiquette et la colonne *structure* représente le sens des relations. Cette dernière nous permet de savoir quel mot est le gouverneur dans la relation syntaxique. Par exemple, dans ParCoTrain, la préposition gouverne le nom dans la relation de Complément de préposition. Or, dans Universal Dependencies c'est le contraire, le nom gouverne la préposition qu'il complète.

1.2. Programmes python pour traiter le corpus dans arborator

Le corpus ParCoTrain étant au format CoNLL-X (fig ??), il est impossible de le rentrer dans Arborator-Grew qui accepte uniquement le format CoNLL-U. En effet, Grew utilise le format CoNLL-U défini par Universal Dependencies qui est basé sur le modèle du format CoNLL-X. Le site internet d'UD fait la liste des dix colonnes du format. Certaines catégories ont été gardées du format CoNLL-X et d'autres ont été remplacées. Voici la liste des 10 colonnes du format CoNLL-U dont nous avons défini les colonnes différentes au format CoNLL-X :

1) ID

2) FORM

3) LEMMA

4) UPOS: Le POS d'universal dependencies du token.

5) XPOS: Le POS spécifique à un treebank du token.

6) FEATS: Les traits morphologiques du token.

7) HEAD

8) DEPREL

9) DEPS: Le graphe de dépendances enrichies sous la forme d'une liste de paires tête-relation de dépendance

10) MISC: Autres annotations

Pour utiliser Grew sur notre corpus, nous avons donc besoin de nous assurer que les colonnes des fichiers CoNLL-X soient traitées comme celles de CoNLL-U. Pour se faire, nous avons écrit un script python (cf. Annexe 3.1) afin de remplacer le numéro de la colonne 9 par un tiret (_). Cette manipulation a permis l'acceptation du fichier par Arborator-Grew. La figure 7 donne un aperçu du corpus avant modification et la figure 8, un aperçu du corpus dont la neuvième colonne a été modifiée.

```
1	Toliko	toliko	Adv	Adv_gen_pos	_	2	DepVAdv	2	DepVAdv
2	znam	znati	V_main	V_main_pres_1_sg_-_-	n=sg|t=pres|r=1	0	Root	0	Root
3	!	!	Z	Z	_	2	Ponct	2	Ponct
```

Fig 7 : Extrait de parcotrain-synt_dev au format CoNLL-X

```
1	Toliko	toliko	Adv	Adv_gen_pos	_	2	DepVAdv	_	DepVAdv
2	znam	znati	V_main	V_main_pres_1_sg_-_-	n=sg|t=pres|r=1	0	Root	_	Root
3	!	!	Z	Z	_	2	Ponct	_	Ponct
```

Fig. 8 : Extrait de parcotrain-synt_dev modifié

Nous avons également rencontré des difficultés au niveau de la taille du corpus. En effet, nous avons fait le choix de découper le corpus en fichiers de 200 phrases afin de faciliter le traitement sur Arborator-Grew. Nous avons donc écrit un script python (cf. Annexe 3.2) pour effectuer ce découpage.

Les deux scripts ont été utilisés via le terminal. Nous avons appliqué le premier script sur les trois sous-corpus : synt_dev, synt_test, synt_train (cf. Chapitre 2). Grâce à la ligne de commande que nous avons écrit dans le terminal, nous avons enregistré les sorties du programme dans des fichiers CoNLL avec la colonne remplacée. Ce sont ces fichiers qui ont servi d'entrée pour le deuxième script qui nous a permis de découper le corpus en 20 fichiers au format CoNLL.

Le tableau suivant (cf. Fig. 9) présente les fichiers avec lesquels nous allons travailler pour effectuer ces conversions.

Sous-corpus	Nom du fichier	Nombre de tokens dans le fichier
synt_dev	dev1	5316
	dev2	5061
synt_test	Test1	5185
	Test2	5345
synt_train	Train1	5575
	Train2	4965
	Train3	5266
	Train4	5594
	Train5	5138
	Train6	5345
	Train7	5080
	Train8	5516
	Train9	5375

	Train10	5326
	Train11	5342
	Train12	5172
	Train13	5973
	Train14	5558
	Train15	5032
	Train16	3743

Fig. 9 : Fichiers utilisés pour nos conversions

Lors de la conversion des traits morphologiques, nous avons remarqué en écrivant les requêtes Grew que celui-ci accède aux différentes features directement par leur nom. Par exemple, c= pour le *cas* dans la dernière colonne. Ainsi, si une feature n'est pas déjà nommée dans le format CoNLL, Grew ne peut pas y accéder. Par conséquent, si on veut pouvoir indiquer la valeur de Case, Gender, Tense… ces noms de features doivent déjà être dans le CoNLL. Pour ce faire, nous avons utilisé un script python (cf. Annexe 3.3) qui, pour chaque ligne contenant un token, insère dans la colonne en question l'ensemble des traits que nous allons utiliser pour la conversion en leur donnant à tous la valeur "_" (cf. Fig. 10). Ainsi, pour chaque token, sans faire la distinction entre les différents POS, nous avons ajouté dans la colonne des traits morphologiques les informations manquantes (Gender=_|Case=_|Number=_|Tense=_ …). Lors de la conversion, les features pertinentes pour le token sont remplies et les autres gardent la valeur "_".

```
1	Toliko	toliko	ADV	Adv_gen_pos	Gender=_|Case=_|Number=_|Degree=_|Polarity=_|Person=_|VerbForm=_|Mood=_|Tense=_|Voice=_	2	DepVAdv	_	DepVAdv=__NOVALUE__
2	znam	znati	VERB	V_main_pres_1_sg_-_-	Gender=_|Case=_|Number=_|Degree=_|Polarity=_|Person=_|VerbForm=_|Mood=_|Tense=_|Voice=_	0	Root	_	n=sg|r=1|Root=__NOVALUE__
3	!	!	PUNCT	Z	Gender=_|Case=_|Number=_|Degree=_|Polarity=_|Person=_|VerbForm=_|Mood=_|Tense=_|Voice=_	2	Ponct	_	Ponct=__NOVALUE__
```

Fig. 10 : Extrait de parcotrain-synt_dev modifié avec les traits morphologiques ajoutés

2. Arborator-Grew

Une fois la préparation des données effectuée, nous pouvons entamer la conversion de nos annotations dont nous allons présenter la méthodologie dans les parties suivantes.

2.1. Conversions

Nous avons décidé de faire nos conversions en plusieurs étapes. Tout d'abord, nous avons converti les POS tags puis les traits morphologiques vers UD directement. Nous avons ensuite converti les étiquettes de relations de ParCotrain vers SUD en adaptant le sens des relations dans les représentations en graphes afin de respecter les lignes directrices de SUD. Nous avons ensuite converti les étiquettes de relations syntaxiques de SUD vers UD en modifiant le sens des relations en fonction des lignes directrices de UD afin de respecter les règles de gouvernance dans les relations.

En convertissant nos étiquettes morpho-syntaxiques sur Arborator, nous avons remarqué que nous devons valider nos phrases une par une. Étant donné le volume de fichiers que nous avons à convertir, nous avons décidé d'utiliser Arborator comme plateforme de test. En effet, Arborator rend une recherche de pattern simple et rapide, ce qui facilite l'observation des informations que nous avons à modifier. De plus, la visualisation de graphes de dépendances nous permet de vérifier le bon fonctionnement de nos requêtes Grew. Nous avons donc utilisé Grew via le terminal pour faire les conversions finales de notre corpus. Pour ce faire, il faut écrire les requêtes Grew dans des fichiers *GRS* qui sont des fichiers où sont décrites les règles de réécritures de graph (Graph Rewriting System).

Nous présentons, dans la partie suivante, différentes requêtes que nous avons testées sur Arborator-Grew et qui seront présentes dans nos fichiers *GRS* utilisés dans le terminal (cf. Annexes 2 et 4).

2.2. Requêtes Grew

Nous avons testé la conversion des étiquettes morpho-syntaxiques sur Arborator grâce à la possibilité de lancer des requêtes Grew via la plateforme. Notre script est un enchaînement de règles de réécriture nommées en fonction des POS sur lesquelles elles opèrent. La figure 11 montre deux règles de réécriture pour le traitement des noms.

```
rule noun {
    pattern {N [upos=N]}
    without {N [xpos=re"N_prop.*"]}
    commands { N.upos = NOUN}
}

rule nounp {
    pattern {N [upos=N, xpos=re"N_prop.*"]}
    commands { N.upos = PROPN }
}
```

Fig. 11 : Règles de réécriture des noms

La règle nommée *noun* traite la conversion des noms communs. Elle recherche dans un premier temps un *pattern*, ici les éléments possédant l'étiquette morpho-syntaxique des noms de ParCoTrain (N). Nous ajoutons la ligne *without* afin de ne pas prendre en compte les noms propres dans la recherche. Pour détecter quels noms sont des noms propres, nous avons utilisé une expression régulière pour chercher les traits morphologiques commençant par "N_prop" dans la colonne xpos du fichier CoNLL-U. Enfin, la ligne *commands* liste les actions à faire. Ici, nous écrivons "NOUN", qui est le POS tag des noms communs dans Universal Dependencies, dans la colonne des *upos* des éléments correspondants au *pattern*. L'ancienne étiquette présente dans cette colonne du fichier CoNLL-U est quant à elle supprimée.

Nous avons également testé nos règles de réécriture des traits morphologiques sur Arborator-Grew. Cette étape, comme expliqué dans le chapitre 3 partie 1.2., requiert la présence préalable des features recherchées dans le pattern. Hormis cela, la conversion ne pose pas de problèmes spécifiques. La figure 6 montre la règle de réécriture des features des verbes aux présents.

```
rule pres {
    pattern { N[ t=pres, VerbForm=_, Mood=_, Tense=_] }
    commands {N.VerbForm=Fin ; N.Mood=Ind ; N.Tense=Pres}
}
```

Fig 12 : Règle de réécriture des traits morphologique des verbes au présent

La règle nommée *pres* traite la conversion des traits morphologiques pour les verbes conjugués au présent. Dans un premier temps, le *pattern*, ici un élément au présent (*t=pres*) et

qui possède les features *VerbForm*, *Mood*, *Tense*. Dans un second temps, avec la ligne *commands*, nous ajoutons les informations aux features citées précédemment. Ici, nous attribuons *Fin* à *VerbForm*, *Ind* à *Mood* et *Pres* à *Tense.*

La réécriture des relations a été plus complexe. En effet, dans certaines relations, le gouverneur, et les dépendants qui lui sont attribués, doivent être inversés. C'est le cas, par exemple, pour la relation *ComplNum* de ParCoTrain qui relie un numéral sous forme paucal au nom qu'il complète. En effet, dans ParCoTrain, le gouverneur de cette relation est le numéral, or dans SUD et UD, le gouverneur est le nom. De ce fait, nous devons, lors de la conversion, inverser la relation de gouvernance et les relations qui leur sont rattachées. La figure 13 montre les règles de réécriture pour le traitement de la relation *ComplNum*.

```
rule ComplNum {
    pattern { e: NUM-[ComplNum]->NOUN}
    commands {
        del_edge e;
        shift NUM ==> NOUN;
        add_edge NOUN-[nummod]->NUM}
}
```

Fig. 13 : Règle de réécriture du complément d'un numéral sous forme du paucal

La règle nommée *ComplNum* traite la conversion de la relation de ParCoTrain vers SUD. Elle recherche dans un premier temps un *pattern*, ici les éléments possédant l'étiquette syntaxique de la relation *ComplNum*. Puis, dans la partie *commands,* nous supprimons la relation existante (*ComplNum*). Grâce à la fonction *shift*, nous basculons les relations de gouvernance de l'élément *NUM* à l'élément *NOUN*. Enfin nous ajoutons une nouvelle relation *nummod* dont le gouverneur est le nom et le numéral est son dépendant.

Chapitre 4 - Résultats

Dans ce chapitre, nous présentons les résultats de nos conversions. Nous allons dans un premier temps, présenter les conversions que nous avons effectuées. Puis, dans un second temps, nous présenterons les étiquettes qui n'ont pas été converties en explicitant les raisons pour lesquelles notre travail n'a pas abouti pour ces annotations.

1. Conversions effectuées

Au niveau de l'annotation morpho-syntaxique, nous avons converti les 35 étiquettes de ParCoTrain (cf. Chapitre 3, Fig. 3). Nous n'avons pas d'équivalence une à une pour certaines étiquettes comme celles des pronoms, par exemple, où 8 sous-catégories de ParCoTrain (pronoms personnels, indéfinis, démonstratifs…) sont converties en une seule étiquette d'UD. En raison de ces réductions dans les étiquettes, nous obtenons 17 étiquettes UD à la fin. A noter que l'étiquette *Autres* n'a pas nécessité de conversion car l'étiquette de ParCoTrain et celle d'UD sont les mêmes (*X*).

Au niveau des traits morphologiques, nous avons converti les 29 traits morphologiques de ParCoTrain (cf. Chapitre 3, Fig. 4 et 5). Les équivalences pour ceux-ci sont de un à un. Ainsi, nous obtenons 29 features en UD.

En ce qui concerne les relations syntaxiques, nous avions 48 étiquettes de ParCoTrain à convertir. Lors du passage de ParCoTrain à SUD, nous avons converti 46 étiquettes. Nous avons trouvé les équivalences pour *Cit* et *PredRap* mais l'écriture du script n'a pas abouti pour ces étiquettes. Nous aborderons la complexité de l'opération dans la partie suivante (cf. Chapitre 4, partie 2). Lors du passage de SUD à UD, nous avons converti 44 étiquettes. *Cit*, *PredRap*, *PredicNom* et *PredicOpt* n'ont pas été converties. En effet, pour *PredicNom*, nous n'avons pas trouvé comment définir un token comme racine de la phrase (*root*). Pour *PredicOpt*, nous avons rencontré des difficultés (cf. Chapitre 4, partie 2) suite à la conversion vers le format SUD.

Certaines conversions de relations sont à améliorer. C'est le cas pour les relations *Coord, ConjCoord* et *Polylex*. Cette dernière sera traitée dans le chapitre suivant (cf. Chapitre 5, partie 1) car nous avons eu recours à une stratégie différente pour la conversion.

La relation *Coord* dans ParCoTrain se fait du premier coordonné vers le second coordonné puis du second coordonné vers le second coordonné et ainsi de suite. La relation *ConjCoord* relie le coordonné précédant la conjonction de coordination à celle-ci. Dans les figures 14 et 15, on peut voir le sens de ces relations.

Or dans SUD et UD, dans la relation *cc*, qui est l'équivalent de *ConjCoord*, la conjonction de coordination est gouvernée par le dernier coordonné. Et dans la relation *conj*, qui est l'équivalent de *Coord*, tous les coordonnés sont gouvernés par le premier. Les figures 16 et 17 permettent d'illustrer ces relations.

1) Više **nisam** bunovan , samo mi hladnoća svežeg jutra **pirka** oko nosa **i** ja se zimogrožljivo **pripijam** uz majku .

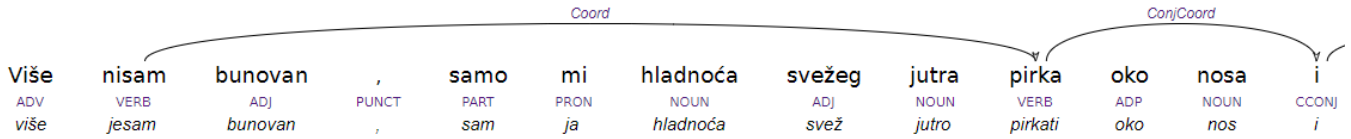

Fig. 14 : graphe de l'exemple 1 représentant les relations Coord et ConjCoord

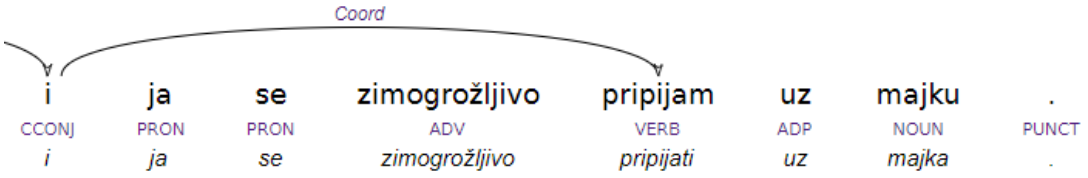

Fig. 15 : graphe de l'exemple 1 représentant la relation Coord

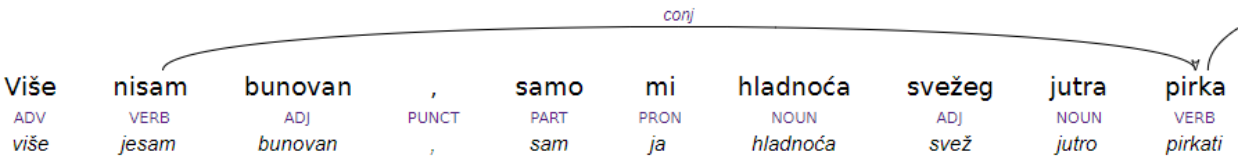

Fig. 16 : graphe de l'exemple 1 représentant la relation conj

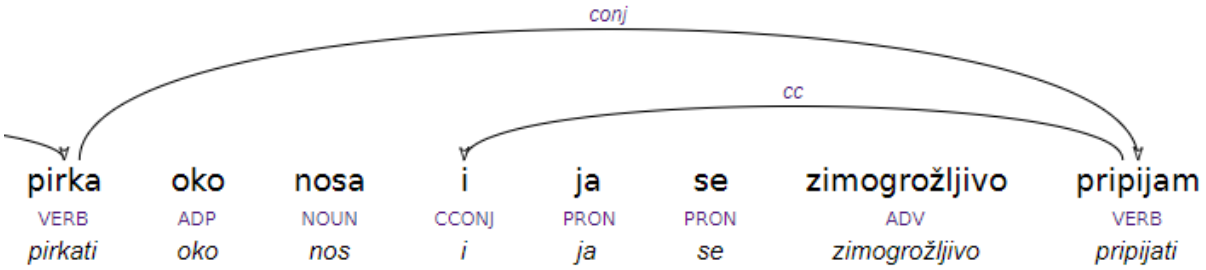

Fig. 17 : graphe de l'exemple 1 représentant les relations conj et cc

Cependant, la conversion des relations *Coord* et *ConjCoord* en *conj* et *cc* que nous avons faite n'est pas optimale. En effet, nous avons écrit la requête Grew pour un, deux ou trois coordonnés. Nous ne sommes pas sûrs que la conversion de la relation fonctionne pour quatre coordonnés ou plus. Une conversion plus optimale serait donc judicieuse.

2. Conversions non effectuées

2.1. PredRap

L'une des relations syntaxiques qui se sont montrées problématiques pour la conversion était *PredRap*. Cette étiquette désigne dans ParCoTrain la relation entre le verbe introductif et le verbe principal du discours rapport. En effet, il n'y a pas d'équivalence directe avec cette étiquette et celles de SUD et d'UD. Si le verbe du discours rapporté interrompt le contenu du discours rapporté, il faut convertir *PredRap* en *parataxis* sinon il faut convertir en *comp:obj*. Ainsi, il faut regarder en contexte gauche et droit pour choisir entre *comp:obj* et *parataxis*, ce qui est difficile à faire automatiquement.

2) "**Napominjem** Poštovanom Komesarijatu" , **pisalo** je u toj žalbi , "u vezi tačke A - dva [...]".

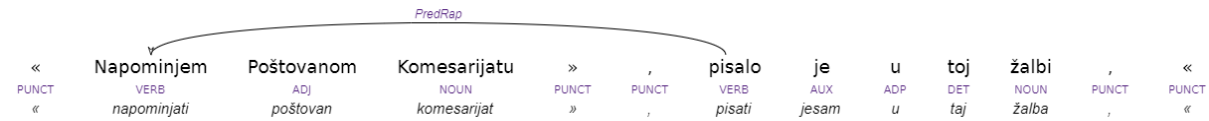

Fig. 18 : graphe de l'exemple 2 représentant la relation PredRap

3) I on **reče** : “Vi **ste** moji”.

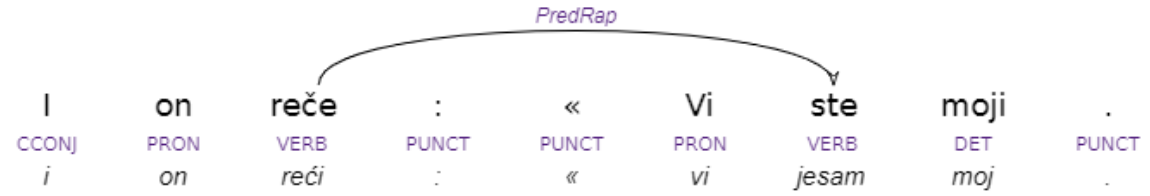

Fig. 19 : *graphe de l’exemple 2 représentant la relation PredRap*

Dans l’exemple 2 extrait du corpus, le verbe du discours rapporté interrompt le discours rapporté. Dans ce cas, il faut remplacer *PredRap* par *parataxis*. Dans l’exemple 3 également extrait du corpus, le verbe du discours rapporté n’interrompt pas le discours rapporté. Dans ce cas, il faut remplacer *PredRap* par *comp:obj*.

2.2. Cit

L’étiquette *Cit* de ParCoTrain n’a pas été convertie par manque de temps. En effet, comme nous l’expliquons dans cette partie, les équivalences pour cette étiquette ne sont pas directes et requièrent, de ce fait, plus de temps que les autres relations. *Cit* est destinée à marquer les emplois métalinguistiques d’un mot. Cependant dans le corpus, cette étiquette couvre des cas de figures très disparates comme, par exemple, des segments de discours direct rapporté. Afin d’essayer de distinguer tous les cas de figures du corpus, nous proposons plusieurs équivalences en fonction de la catégorie morpho-syntaxique des mots entrant en jeu dans cette relation. Ainsi, si le gouverneur de *Cit* est un verbe et porte donc l’étiquette morpho-syntaxique *VERB*, il s’agit d’un segment de discours direct rapporté. *Cit* devra donc être converti en *comp* en SUD et *ccomp* en UD. Si le gouverneur est un nom, l’étiquette de modifieur de type approprié est attendue en fonction de l’étiquette morpho-syntaxique du mot qui porte la fonction *Cit*. Nous obtenons donc :

- en SUD : *udep* pour un nom, *mod* pour un adjectif, un adverbe ou une particule
- en UD : *nmod* pour un nom, *amod* pour un adjectif, *advmod* pour un adverbe ou une particule

2.3. PredicOpt

La conversion de la relation *PredictOpt* a posé des problèmes différents de ceux vus précédemment. En effet, cette étiquette qui désigne le prédicatif optionnel c'est-à-dire un complément nominal ou adjectival d'un verbe optionnellement attributif, a été convertie en *mod* en SUD. Cependant, d'autres relations du corpus ont été transformées en *mod* comme *Neg* ou *ExtraPred* (cf. Annexe 1). Celles-ci possèdent des caractéristiques spécifiques, comme la catégorie morpho-syntaxique du dépendant, qui permettent leur conversion de SUD à UD. Or, pour *PredictOpt*, il n'y a pas d'éléments qui permettent de faire la distinction entre les autres relations *mod* et celle résultant de la conversion de *PredictOpt*. Dans l'état actuel, nous avons une règle de conversion finale qui transforme les mod restant en advmod. La conversion de cette étiquette n'a donc pas été traitée spécifiquement et cela entraîne une perte d'information par rapport au corpus initial.

Chapitre 5 - Analyse linguistique de la perte d'information liée à la conversion

Comme nous l'avons abordé dans les résultats, certaines annotations ont subi un syncrétisme lors de leur conversion. Dans ce chapitre, nous nous appliquons à analyser les répercussions de la transformation des étiquettes. Nous présentons également une stratégie que nous avons mise en place afin de pallier les potentielles pertes d'informations entraînées par la conversion.

1. Perte d'informations

1.1. Les étiquettes morpho-syntaxiques

Lors de la conversion des étiquettes morpho-syntaxiques, nous avons perdu beaucoup de précisions par rapport au corpus initial. En effet, comme nous l'avons vu dans le chapitre 4 partie 1, nous sommes passés de 35 étiquettes dans ParCoTrain à seulement 17 dans UD. Les sous-catégories des pronoms, des adverbes, des déterminants (adjectifs autre que qualificatifs dans ParCoTrain), des noms (communs et collectifs) et des numéraux peuvent être utiles lors d'analyses linguistiques en corpus notamment lors de recherches en contexte pour étudier les structures dans la langue. En effet, les variations au sein d'une structure, avec un étiquetage fin comme celui de ParCoTrain, permettent de détecter plus facilement les structures identiques et au contraire, celles qui sont proches mais non identiques. Dans sa thèse *Un treebank pour le serbe : constitution et exploitations*, Miletic (2018) explique les intérêts d'une annotation morpho-syntaxique fine : "Le décodage du fonctionnement syntaxique du serbe s'appuie fortement sur les traits morphosyntaxiques fins comme le cas, le genre, le nombre ou la personne. Un parser doit donc disposer de ce type d'information pour obtenir des performances solides sur cette langue. Par conséquent, un treebank doit en être doté." (Miletic, 2018). Il faut donc garder les informations initiales des catégories grammaticales qui se trouvent dans la colonne *xpos* du fichier CONLL-U au moment de la conversion. Lors de recherches linguistiques avec les étiquettes converties, il faudra utiliser la colonne *xpos* et non *pos* pour avoir accès à ces informations.

1.2. Les relations syntaxiques

Trouver les équivalences de ParCoTrain en SUD et UD (cf. Annexe 1) s'est parfois avéré délicat pour les relations syntaxiques. En effet, les étiquettes SUD (23) sont moins nombreuses que celles de ParCoTrain (48). Le nombre de relations finales en UD (28) est également inférieur au nombre initial d'étiquettes de ParCoTrain. Il est donc difficile de faire la différence entre certaines relations avec les étiquettes UD. C'est le cas pour *PredicComplObj* qui désigne le prédicatif complémentaire lié à l'objet direct et *PredicComplSuj* qui désigne le prédicatif complémentaire lié au sujet. Ces deux étiquettes qui ne représentent pas la même relation sont convertis en xcomp en UD et n'ont pas de traits distinctifs. De ce fait, une fois converties, seule une étude en contexte permet de distinguer ces étiquettes, l'accès à l'information n'est donc plus direct.

3) I sve ponovo beše isto kao nekada .

Fig. 22 : *: graphe de l'exemple 3 représentant la relation DepEx_DepVAdv*

La différence de granularité entre les étiquettes de ParCoTrain et celles d'UD est la source principale de la perte d'information. En effet, un autre exemple de cette différence est l'annotation de la relation des ellipses qui est désignée, dans ParCoTrain par le préfixe *DepEx_* ajouté à l'étiquette de l'élément dont le gouverneur est élidé. Cette étiquette est remplacée par *orphan* en UD. Dans l'exemple 3 (Fig. 22), la relation *DepEx_DepVAdv* est convertie en UD en *orphan*. De ce fait, nous perdons les informations de l'étiquette de l'élément dont le gouverneur est élidé.

2. Stratégie pour éviter la perte d'information : ajout d'une deep feature

Le dépendant de la racine de la proposition exprimant l'emphase, privé de fonction syntaxique, soit Emph dans ParCoTrain, a posé problème lors de la conversion. En effet, dans SUD, nous devrions le convertir en mod. Cependant, une autre étiquette de PraCoTrain est convertie en mod dans SUD, ExtraPred qui correspond au dépendant de la racine de la proposition sous forme d'un adverbe extra-prédicatif. Aucun élément syntaxique ou morpho-syntaxique ne permet de différencier ces deux étiquettes. En revanche, dans UD il existe l'étiquette *advmod:emph*. Il est donc important de pouvoir identifier les occurrences de Emph dans la conversion SUD.

La solution que nous avons trouvée est d'ajouter une deep feature *emph* à *mod* intégrée par une arobase dans SUD (*mod@emph*) pour l'étiquette de l'emphase dans ParCoTrain. Cette deep feature ne fait pas partie de la liste initiale de SUD mais elle est indispensable dans notre conversion car elle nous permet de faire la distinction entre l'étiquette *mod* représentant *ExtraPred* et l'étiquette *mod* représentant *Emph*. Cette distinction permet de convertir les étiquettes *mod@emph* en *advmod:emph* dans UD et *mod* en *advmod* dans UD. Ainsi on peut résumer les conversions dans le tableau suivant.

	ParCoTrain		SUD		UD	
Définition	Étiquette	Structure	Étiquette	Structure	Étiquette	Structure
dépendant de la racine de la proposition exprimant l'emphase, privé de fonction syntaxique	Emph	token-[Emph] → PART	mod@emph	tok-[mod@emph] →PART	advmod:emph	token-[advmod:emph] → PART
dépendant de la racine de la proposition sous forme d'un adverbe extra-prédicatif	ExtraPred	token-[ExtraPred] → PART	mod	tok-[mod] →PART	advmod	token-[advmod] → PART

Fig. 20 : Tableau d'équivalence des relations Emph et Extrapred

Nous avons également utilisé cette stratégie d'ajout d'une deep feature pour la relation Polylex qui réunit les éléments d'une locution prépositionnelle ou adverbiale ou d'une conjonction complexe. En effet, la deep feature *@fixed* n'existe pas dans SUD mais dans UD, il existe l'étiquette *fixed* qui permet de décrire cette relation *Polylex*.

	ParCoTrain		SUD		UD	
Définition	Étiquette	Structure	Étiquette	Structure	Étiquette	Structure
relation réunissant les éléments d'une locution prépositionnelle ou adverbiale ou d'une conjonction complexe	Polylex	X1, X2, X3 X1-[Polylex] →X2-[Polylex] → X3	comp@fixed	X1, X2, X3 X1-[comp@fixed] → X2 X1-[comp@fixed] → X3	fixed	X1, X2, X3 X1-[comp@fixed]→ X2 X1-[comp@fixed]→ X3

Fig. 21 : Tableau d'équivalence de la relation Polylex

Conclusion et perspectives

Dans ce mémoire, nous avons cherché à convertir les étiquettes morpho-syntaxiques, les traits morphologiques et les relations syntaxiques du corpus ParCoTrain issu du projet ParCoLab vers le format Universal Dependencies.

Il nous a fallu tout d'abord établir un état de l'art, avant de réaliser un travail de préparation des données du corpus ParCoTrain par des programmes python. Nous souhaitions utiliser Arborator-Grew pour la conversion de nos annotations mais celui-ci n'est pas adapté au traitement de corpus de grande envergure. Nous avons donc eu recours à l'outil Grew qui nous a permis, à travers des scripts de réécritures de graphes, d'effectuer nos conversions.

Nous avons converti les 35 étiquettes morpho-syntaxiques et les 29 traits morphologiques de ParCoTrain. Au niveau des relations syntaxiques, nous avons converti en Universal Dependencies 44 étiquettes des 48 étiquettes de ParCoTrain. Cependant, nous avons utilisé le format Surface-syntax Universal Dependencies qui suit les mêmes critères de distributions que ParCoTrain (privilégiant les têtes fonctionnelles) et qui nous a permis de convertir 46 étiquettes des 48 étiquettes. Les conversions restantes demandent des recherches approfondies quant à la formulation des requêtes de réécriture mais sont réalisables. Le tableau d'équivalence ParCoTrain-SUD-UD que nous avons dressé (cf. Annexe 1) pourra servir de base de travail à l'amélioration de nos conversions qui n'ont pas été vérifiées en contexte.

Nous avons également étudié les pertes d'informations liées à la conversion des étiquettes du corpus ParCoTrain. Celles-ci restent moindre quant à la quantité d'étiquettes converties et peuvent être évitées en améliorant nos propositions de conversions par l'ajout, par exemple, d'une deep feature lors de la conversion vers SUD.

Ainsi, notre travail présente une méthodologie claire basée sur la recherche d'équivalences d'annotations et la conversion pratique de celle-ci afin de permettre la favorisation de la réutilisation de corpus annotés.

Références

Agić, Ž., & Ljubešić, N. (2015). Universal Dependencies for Croatian (that work for Serbian, too). *The 5th Workshop on Balto-Slavic Natural Language Processing*, 1-8.

Buchholz, S., & Marsi, E. (2006). CoNLL-X Shared Task on Multilingual Dependency Parsing. *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, 149-164.

Davies M., The Corpus of Contemporary American English as the first reliable monitor corpus of English, *Literary and Linguistic Computing*, Volume 25, Issue 4, December 2010, Pages 447–464.

De Marneffe, M.-C., Manning, C. D., Nivre, J., & Zeman, D. (2021). Universal Dependencies. *Computational Linguistics*, 1-54.

Gerdes, K., Guillaume, B., Kahane, S., & Perrier, G. (2018). *SUD or Surface-Syntactic Universal Dependencies : An annotation scheme near-isomorphic to UD*. Universal Dependencies Workshop 2018.

Gerdes, K., Guillaume, B., Kahane, S., & Perrier, G. (2019). *Improving Surface-syntactic Universal Dependencies (SUD) : Surface-syntactic relations and deep syntactic features*. TLT 2019 - 18th International Workshop on Treebanks and Linguistic Theories.

Guibon, G., Courtin, M., Gerdes, K., & Guillaume, B. (2020). When Collaborative Treebank Curation Meets Graph Grammars. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 5291–5300, Marseille, France. European Language Resources Association.

Guillaume, B., De Marneffe, M.-C., & Perrier, G. (2019). *Conversion et améliorations de corpus du français annotés en Universal Dependencies*. Revue TAL, ATALA (Association pour le Traitement Automatique des Langues), 2019, 60 (2), pp.71-95.

Guillaume, B. (2021). Graph Matching and Graph Rewriting : GREW tools for corpus exploration, maintenance and conversion. *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, 168‑175.

Guillaume, B., Bonfante, G., Masson, P., Morey, M., & Perrier, G. (2012). Grew : Un outil de réécriture de graphes pour le TAL (Grew: a Graph Rewriting Tool for NLP) [in French]. *Proceedings of the Joint Conference JEP-TALN-RECITAL 2012, volume 5: Software Demonstrations*, 1‑2.

Jakovljević, B., Kovacevic, A., Sečujski, M., & Markovic, M. (2014). *A Dependency Treebank for Serbian : Initial Experiments* (Vol. 8773, p. 49).

Ljubešić, N., & Klubička, F. (2014). Bs,hr,srWaC - Web Corpora of Bosnian, Croatian and Serbian. *Proceedings of the 9th Web as Corpus Workshop (WaC-9)*, 29‑35.

Martin, É. (1988). Frantext, la base de données textuelles du français de l'INaLF. *Bulletin de l'EPI (Enseignement Public et Informatique)*, (52), 184-200.

Miletic, A. (2015). *Création des ressources de TAL pour une langue peu dotée : Le cas du serbe*. 80. Université Toulouse-Jean Jaurès.

Miletic, A. (2018a). *Un treebank pour le serbe : Constitution et exploitations Tome 1*. Université Toulouse-Jean Jaurès.

Miletic, A. (2018b). *Un treebank pour le serbe : Constitution et exploitations Tome 2 : Annexes*. Université Toulouse-Jean Jaurès.

Miletic, A., Fabre, C., & Stosic, D. (2015). Construction du jeu d'étiquettes pour le parsing du serbe. In *22e journées du Traitement Automatique des Langues Naturelles*.

Miletic, A., Fabre, C., & Stosic, D. (2019). De la constitution d'un corpus arboré à l'analyse syntaxique du serbe. *Traitement Automatique des Langues*.

Miletic, A., & Thuilier, J. (2021). Minimisation de la longueur des dépendances et position de l'adjectif épithète en français et en serbe. *Langages*, *223*(3), 81‑102.

Nivre, J., de Marneffe, M.-C., Ginter, F., Hajič, J., Manning, C. D., Pyysalo, S., Schuster, S., Tyers, F., & Zeman, D. (2020). Universal Dependencies v2 : An Evergrowing Multilingual Treebank Collection.

Samardžić, T., Starović, M., Agić, Ž., & Ljubešić, N. (2017, avril 4). Universal Dependencies for Serbian in Comparison with Croatian and Other Slavic Languages. *Samardžić, Tanja; Starović, Mirjana; Agić, Željko; Ljubešić, Nikola (2017). Universal Dependencies for Serbian in Comparison with Croatian and Other Slavic Languages. In: Proceedings of the 6th Workshop on Balto-Slavic Natural Language Processing, Valencia, Spain, 4 April 2017, Association for Computational Linguistic.* Proceedings of the 6th Workshop on Balto-Slavic Natural Language Processing, Valencia, Spain.

Sečujski, M., Obradović, R., Pekar, D., Jovanov, L., & Delić, V. (2002, September). AlfaNum system for speech synthesis in Serbian language. In *International Conference on Text, Speech and Dialogue* (pp. 237-244). Springer, Berlin, Heidelberg.

Thomas, P.-L. (1994). Serbo-croate, serbe, croate..., bosniaque, monténégrin : Une, deux..., trois, quatre langues ? *Revue des Études Slaves*, *66*(1), 237‑259.

Utvić, M. (2011). Annotating the corpus of contemporary Serbian. In *Proceedings of the INFOtheca '12 Conference* (pp. 36-47).

Vitas, D., & Krstev, C. (2004). Intex and Slavonic morphology. *INTEX pour la linguistique et le traitement automatique des langues, Presses Universitaires de Franche-Comté*, 19-33.

Annexes

Annexe 1 : Tableau d'équivalence des relations syntaxiques

Légende du tableau :

- éléments en vert : conversions effectuées dans le script ParCoTrain→SUD (cf. Annexe 2.3)
- eléments en bleu : conversions effectuées dans le script SUD→UD (cf. Annexe 2.4)
- éléments en jaune : conversions à améliorer
- éléments en rouge : conversions non effectuées

	ParCoTrain		SUD		UD	
Définition	Étiquette	Structure	Étiquette	Structure	Étiquette	Structure
Apposition	Ap	NOUN-[Ap] →NOUN	appos	NOUN-[appos] →NOUN	appos	NOUN-[appos] →NOUN
Verbe auxiliaire dans une forme verbale composée	AuxV	VERB-[AuxV] →AUX	comp:aux	AUX= root AUX-[comp:aux] →VERB	aux	VERB-[aux] →AUX
élément métalinguistique	Cit	gouverneur-[Cit] →dépendant	Si le gouverneur est un VERB : **comp** Si le gouverneur est un autre nom : **udep** pour un nom **mod** pour un adjectif **mod** pour un adverbe ou une particule	VERB-[comp] →dep NOUN-[udep] → dep ADJ-[mod] → dep ADV-[mod] → dep	Si le gouverneur est un VERB : **ccomp** Si le gouverneur est un autre nom : **nmod** pour un nom **amod** pour un adjectif **advmod** pour un adverbe ou une particule	VERB-[ccomp] →dep NOUN-[nomod] → dep ADJ-[amod] → dep ADV-[advmod] → dep
complément d'un numéral sous forme du paucal	ComplNum	NUM-[ComplNum] → NOUN	nummod	NOUN-[nummod] → NUM	nummod:gov	NOUN-[nummod:gov] → NUM

	ParCoTrain		SUD		UD	
Définition	Étiquette	Structure	Étiquette	Structure	Étiquette	Structure
complément de préposition	ComplPrep	ADP-[ComplPrep] → NOUN	comp:obj	ADP-[comp:obj] → NOUN	case	NOUN-[case] → ADP
relation entre le coordonné précédant immédiatement la conjonction de coordination et la conjonction elle-même	ConjCoord	*A et B* A-[ConjCoord] → et	cc	*A et B* B–[cc]→ et	cc	*A et B* B–[cc]→ et
relation entre la conjonction de coordination et le dernier coordonné *(+ relation entre les coordonnés)*	Coord	*A, B et C* A-[Coord] → B et-[Coord] → C ——— *A, B, C* A-[Coord] → B B-[Coord] → C	conj	*A, B et C* A-[conj]→B A-[conj]→C	conj	*A, B et C* A-[conj]→B A-[conj]→C
relation entre deux éléments d'une structure corrélative	Correl	gouverneur-[Correl] → dépendant	mod	gouverneur-[mod] →dépendant	Si dépendant = SCONJ : **advcl**	mot-[advcl] → SCONJ mot-[dep]→ mot

	ParCoTrain		SUD		UD	
Définition	Étiquette	Structure	Étiquette	Structure	Étiquette	Structure
					Si autre : **dep**	
dépendant d'un adjectif sous forme d'un adverbe	DepAdjAdv	ADJ-[DepAdjAdv] → ADV	mod	ADJ-[mod] → ADV	advmod	ADJ-[advmod] → ADV
dépendant d'un adjectif sous forme d'un nom fléchi	DepAdjCas	ADJ-[DepAdjCas] → NOUN	mod	ADJ-[mod] → NOUN	obl	ADJ-[obl] → NOUN
dépendant d'un adjectif sous forme d'un groupe prépositionnel	DepAdjPrep	ADJ-[DepAdjPrep] → ADP	mod	ADJ-[mod] → ADP	obl	ADJ-[obl] → NOUN puis NOUN-[case] → ADP
dépendant d'un adverbe sous forme d'un adverbe	DepAdvAdv	ADV-[DepAdvAdv] → ADV	mod	ADV-[mod] → ADV	advmod	ADV-[advmod] → ADV
dépendant d'un adverbe sous forme d'un nom fléchi	DepAdvCas	ADV-[DepAdvCas] → NOUN	Si dépendant Case = Gen : **det@numgov** Sinon : **mod**	NOUN-[det@numgov] → Adv ADV-[mod]→ NOUN	Si dep Case =Gen : **det:numgov** Sinon : **obl**	NOUN-[det:numgov] → ADV ADV–[obl] → NOUN
dépendant d'un adverbe sous forme d'une	DepAdvPrep	ADV-[DepAdvPrep]	mod	ADV-[mod] → ADP	obl	ADV-[obl] → NOUN

	ParCoTrain		SUD		UD	
Définition	Étiquette	Structure	Étiquette	Structure	Étiquette	Structure
préposition		→ ADP				puis NOUN-[case] → ADP
ellipse : préfixe ajouté à l'éti-quette de l'élément dont le gouverneur est élidé	DepEx_	gouverneur-[DepEx_relation] → dépendant	orphan	gouverneur-[orphan] → dépendant	orphan	gouverneur-[orphan] → dépendant
dépendant de nom sous forme d'un adjectif	DepNAdj	NOUN-[DepNAdj] → ADJ	mod	NOUN-[mod] → ADJ	amod	NOUN-[amod] → ADJ
dépendant de nom sous forme d'un nom fléchi	DepNCas	NOUN-[DepNCas] → NOUN	mod	NOUN-[mod] → NOUN	obl	NOUN-[obl] → NOUN
dépendant de nom sous forme d'un groupe prépositionnel	DepNPrep	NOUN 1-[DepNPrep] →ADP→ NOUN2	mod	NOUN 1-[mod] → ADP→ NOUN2	obl	NOUN 1-[obl] → NOUN 2-[case] → ADP
dépendant d'un verbe sous forme d'un nom fléchi et qui n'est pas un ObjDir, ObjIndir ou prédicatif	DepVCas	VERB-[DepVCas] → NOUN	mod	VERB-[mod] → NOUN	obl	VERB-[obl] → NOUN
dépendant d'un verbe sous forme d'un adverbe	DepVAdv	VERB-[DepVAdv]	mod	VERB-[mod] → ADV	advmod	VERB-[advmod] → NOUN

	ParCoTrain		SUD		UD	
Définition	Étiquette	Structure	Étiquette	Structure	Étiquette	Structure
		→ ADV				
dépendant infinitif d'un verbe	DepVInf	VERB-[DepVInf] → VERB_infinitif	comp :obj@x	VERB-[comp :obj@x] → VERB_infinitif	xcomp	VERB-[xcomp] → VERB_infinitif
dépendant d'un verbe introduit par un participe présent ou passé	DepVPart	VERB-[DepVPart] → PART	mod	VERB-[mod] → PART	advcl	VERB-[advcl] → PART
dépendant d'un verbe sous forme d'un groupe prépositionnel qui n'est pas un ObjIndir ou un prédicatif	DepVPrep	VERB-[DepVPrep] → ADP → NOUN	mod	VERB-[mod] → ADP → NOUN	obl	VERB-[obl] → NOUN-[case] → ADP
dépendant de la racine de la proposition exprimant l'emphase, privé de fonction syntaxique	Emph	gouverneur-[Emph] → PART	mod@emph	gouverneur-[mod@emph] →PART	advmod:emph	gouverneur -[advmod:emph] → PART
dépendant de la racine de la proposition sous forme d'un adverbe extra-prédicatif	ExtraPred	gouverneur-[ExtraPred] → PART	mod	gouverneur-[mod] →PART	advmod	gouverneur-[advmod] → PART
dépendant de la racine de la proposition sous forme	Interrog	VERB-[Interrog]	discourse	VERB-[discourse]	discourse	VERB-[discourse] → marqueur

	ParCoTrain		SUD		UD	
Définition	Étiquette	Structure	Étiquette	Structure	Étiquette	Structure
d'un marqueur d'interrogation		→ marqueur d'interrogation		→ marqueur d'interrogation		d'interrogation
juxtaposition de deux éléments de haut niveau où aucune autre relation ne s'applique	Juxt	gouverneur-[Juxt] → dépendant	parataxis	gouverneur-[parataxis] → dépendant	parataxis	gouverneur-[parataxis] → dépendant
négation (verbale ou nominale)	Neg	VERB-[Neg] → ADV | PART	mod	VERB-[mod] → ADV | PART	advmod	VERB-[advmod] → ADV | PART
objet direct, à l'accusatif ou au génitif	ObjDir	VERB-[ObjDir] → dépendant	comp:obj	VERB-[comp:obj] → dépendant	obj	VERB-[obj] → dépendant
objet indirect au datif	ObjIndirCas	VERB-[ObjIndirCas] → NOUN	comp:obl	VERB-[comp:obl] → NOUN	iobj	VERB-[iobj] → NOUN
objet indirect réalisé comme o 'de' + N_locatif	ObjIndirPrep	VERB-[ObjIndirPrep] → ADP	comp:obl	VERB-[ObjIndirPrep] → ADP	obl:arg	VERB-[obl:arg] → NOUN
relation réunissant les éléments d'une locution prépositionnelle ou adverbiale ou d'une conjonction complexe	Polylex	X1, X2, X3 X1-[Polylex] →X2-[Polylex] →X3	comp@fixed	X1, X2, X3 X1-[comp@fixed] → X2 X1-[comp@fixed]→ X3	fixed	X1, X2, X3 X1-[comp@fixed] → X2 X1-[comp@fixed] → X3

	ParCoTrain		SUD		UD	
Définition	Étiquette	Structure	Étiquette	Structure	Étiquette	Structure
relation entre le subordonnant et le prédicat de la complétive	PredCompletive	VERB1-[sub] →SCONJ SCONJ-[PredCompletive] →VERB2	comp:obj	VERB1-[comp :obj] →SCONJ SCONJ-[comp:obj] →VERB2	ccomp	V1—ccomp V2, V2—mark C_sub VERB1-[ccompj] → VERB2 VERB2-[mark] →SCONJ
relation entre le prédicat de la principale et le prédicat de la percontative	PredPercont	VERB 1-[PredPercont] → VERB 2	comp:obj	VERB 1-[comp:obj] → VERB 2	ccomp	VERB 1-[ccomp] → VERB 2
relation entre le verbe introductif et le verbe principal du discours rapport	PredRap	VERB 1-[PredRap] → VERB 2 (ou autre tête du segment qui représente le discours rapporté)	**comp:obj** Si le verbe du discours rapporté interromps le contenu du discours rapporté: **parataxis**		**ccomp** Si le verbe du discours rapporté interromps le contenu du discours rapporté: **parataxis**	
relation entre l'antécédent d'une relative et son prédicat	PredRel	VERB 1-[PredRel] → VERB 2	mod:relcl	VERB 1-[mod:relcl] → VERB 2	acl	VERB 1-[acl] → VERB 2
relation entre le subordonnant et le prédicat de la subordonnée	PredSub	SCONJ-[PredSub] → VERB	comp:obj	SCONJ-[comp :obj] → VERB	advcl	VERB 1-[advcl] → VERB 2 -[mark] → SCONJ

	ParCoTrain		SUD		UD	
Définition	Étiquette	Structure	Étiquette	Structure	Étiquette	Structure
prédicatif adverbial : complément adverbial du verbe biti ‘être’	PredicAdv	biti ‘être’-[PredicAdv] → ADV	comp:pred	biti ‘être’-[comp:pred] → ADV	root	ADV = root, ADV-[cop]→ biti ‘être’, ADV-[nsubj] → NOUN
prédicatif complémentaire lié à l’objet direct : complément nominal, adjectival ou prépositionnel d’un verbe obligatoirement attributif autre que le verbe biti ‘être’	PredicComplObj	VERB-[PredicComplObj] → prédicatif	comp:pred	VERB-[comp:pred] → prédicatif	xcomp	VERB-[xcomp] → prédicatif
prédicatif complémentaire lié au sujet : complément nominal, adjectival ou prépositionnel d’un verbe obligatoirement attributif autre que le verbe biti ‘être’	PredicComplSuj	VERB-[PredicComplSuj] → prédicatif	comp:pred	VERB-[comp:pred] → prédicatif	xcomp	VERB-[xcomp] → prédicatif
prédicatif nominal : complément nominal du verbe biti ‘être’	PredicNom	biti ‘être’-[PredicNom] → prédicatif	comp:pred	biti ‘être’-[comp:pred] → prédicatif	root	prédicatif = root, prédicatif -[cop] → biti ‘être’, prédicatif-[nsubj] → sujet
prédicatif optionnel : complément nominal ou	PredicOpt	VERB-[PredicOpt]	mod	VERB-[mod] → prédicatif	amod/ nmod/ advmod	VERB-[relation] → prédicatif

	ParCoTrain		SUD		UD	
Définition	Étiquette	Structure	Étiquette	Structure	Étiquette	Structure
adjectival d'un verbe optionnellement attributif		→ prédicatif			en fonction du POS du prédicatif	
relie le verbe au pronom réflexif	Ref	VERB-[Ref] → PRON	comp@pass	VERB-[comp@pass] → PRON	expl	VERB-[expl] → PRON
relie la racine externe et la tête de la phrase	Root		root		root	
relation entre le verbe principal et le subordonnant introduisant une proposition subordonnée	Sub	VERB-[Sub] → SCONJ – (+ SCONJ-[Pred(Sub|Comp]) → VERB 2)	Non suivi de PredCompletive : **mod** – suivi de PredCompletive : **comp:obj**	VERB-[mod] → SCONJ – SCONJ-[comp :obj] → VERB	advcl – ccomp	VERB 1-[advcl|ccomp] → VERB 2-[mark] → SCONJ
sujet grammatical, exprimé par le nominatif	Suj	VERB-[Suj] → NOUN	subj	VERB-[subj] → NOUN	Sujet avec POS=SCONJ + lemma=da|što|kad → csubj Sujet = verbe à l'infinitif autres : nsubj	VERB-[relation] → NOUN

	ParCoTrain		SUD		UD	
Définition	Étiquette	Structure	Étiquette	Structure	Étiquette	Structure
sujet logique, exprimé par le datif, génitif ou accusatif	SujLog	VERB-[SujLog] → NOUN	subj (subj@expl = syntactic position, no semantic value)	VERB-[subj] → NOUN	nsubj	VERB-[nsubj] → NOUN
relie la ponctuation au premier token précédent qui n'en est pas une	Ponct	token-[Ponct] → PUNCT	punct	token-[punct] → PUNCT	punct	token-[punct] → PUNCT

Annexe 2 : Scripts Grew

Annexe 2.1 : Conversions POS

```
package POS {
    rule noun {
        pattern {N [upos=N]}
        without {N [xpos=re"N_prop.*"]}
        commands { N.upos = NOUN} }

    rule nounp {
        pattern {N [upos=N, xpos=re"N_prop.*"]}
        commands { N.upos = PROPN }}

    rule num {
        pattern {N [upos=Num]}
        commands { N.upos = NUM }}

    rule det {
        pattern {N [upos=A]}
        without {N [xpos=re"A_qual.*"]}
        commands { N.upos = DET}}


    rule adj {
        pattern {N [upos=A, xpos=re"A_qual.*"]}
        commands { N.upos = ADJ}}

    rule verb {
        pattern {N [upos=V_main]}
        commands { N.upos = VERB}}

    rule aux{
        pattern {N [upos=V_aux]}
        commands { N.upos = AUX}}
```

```
rule pron{
        pattern {N [upos=P]}
        commands { N.upos = PRON}}

rule sconj {
        pattern {N [upos=C_sub]}
        commands { N.upos = SCONJ}}

rule cconj {
        pattern {N [upos=C_coord]}
        commands { N.upos = CCONJ}}

rule adv {
        pattern {N [upos=Adv]}
        commands { N.upos = ADV}}

rule particule {
        pattern {N [upos=Part]}
        commands { N.upos = PART}}

rule ponctuation {
        pattern {N [upos=Z]}
        commands { N.upos = PUNCT}}


rule prep {
        pattern {N [upos=Prep]}
        commands { N.upos = ADP}}

rule interjection {
        pattern {N [upos=I]}
        commands { N.upos = INTJ}}

rule lettresiso{
        pattern {N [upos=L]}
        commands { N.upos = NOUN}}
```

```
}

strat pct2udpos {
        Onf(POS)}
```

Annexe 2.2 : Conversions traits morphologiques

```
package Morph {
        %Genre
         rule genreF {
                pattern { N [g=f, Gender=_] }
                commands { N.Gender = Fem}}

        rule genreM {
                pattern { N [g=m, Gender=_] }
                commands { N.Gender = Masc }}

        rule genreN {
                pattern { N [g=n, Gender=_] }
                commands { N.Gender = Neut }}

        %Nombre
         rule singulier {
                pattern { N [n=sg, Number=_] }
                commands { N.Number = Sing }}

        rule pluriel {
                pattern { N [n=pl, Number=_] }
                commands { N.Number = Plur }}

        %Cas
        rule CasIns {
                pattern { N[ c=ins, Case=_]}
                commands {N.Case=Ins}}
```

```
rule CasNom {
        pattern { N[ c=nom, Case=_]}
        commands {N.Case=Nom}}

rule CasGen {
        pattern { N[ c=gen, Case=_]}
        commands {N.Case=Gen}}

rule CasDat {
        pattern { N[ c=dat, Case=_]}
        commands {N.Case=Dat}}

rule CasAcc {
        pattern { N[ c=acc, Case=_]}
        commands {N.Case=Acc}}

rule CasVoc {
        pattern { N[ c=voc, Case=_]}
        commands {N.Case=Voc}}

rule CasLoc {
        pattern { N[ c=loc, Case=_]}
        commands {N.Case=Loc}}




%personne
rule PersUn {
        pattern { N[ r=1, Person=_]}
        commands {N.Person=1}}

rule PersDeux {
        pattern { N[ r=2, Person=_]}
        commands {N.Person=2}}

rule PersTrois {
```

```
        pattern { N[ r=3, Person=_]}
        commands {N.Person=3}}

%verbe
rule imper {
        pattern { N[ t=imper, VerbForm=_,Mood=_] }
        commands {N.VerbForm=Fin ; N.Mood=Imp}}

rule pres {
        pattern { N[ t=pres, VerbForm=_, Mood=_, Tense=_] }
        commands {N.VerbForm=Fin ; N.Mood=Ind ; N.Tense=Pres}}

rule aor {
        pattern { N[ t=aor, VerbForm=_, Mood=_, Tense=_] }
        commands {N.VerbForm=Fin ; N.Mood=Ind ; N.Tense=Past}}

rule fut {
        pattern { N[ t=fut, VerbForm=_, Mood=_, Tense=_] }
        commands {N.VerbForm=Fin ; N.Mood=Ind ; N.Tense=Fut}}

rule impf {
        pattern { N[ t=impf, VerbForm=_, Mood=_, Tense=_] }
        commands {N.VerbForm=Fin ; N.Mood=Ind ; N.Tense=Imp}}

rule partpres {
        pattern { N[ t=partpres, VerbForm=_, Tense=_] }
        commands {N.VerbForm=Conv ; N.Tense=Pres}}

rule partpass {
        pattern { N[ t=partpass, VerbForm=_, Tense=_] }
        commands {N.VerbForm=Conv ; N.Tense=Past}}

rule partact {
        pattern { N[ t=partact, VerbForm=_, Voice=_] }
        commands {N.VerbForm=Part ; N.Voice=Act}}

rule partpast {
```

```
            pattern { N[ t=partpast, VerbForm=_, Voice=_] }
            commands {N.VerbForm=Part ; N.Voice=Pass}}

        rule inf {
            pattern { N[ t=inf, VerbForm=_]}
            commands {N.VerbForm=Inf}}

        %Degré de comparaison
        rule positif {
            pattern { N[xpos=re".*pos.*", Degree=_] }
            commands {N.Degree=Pos}}

        rule comparatif {
            pattern { N[xpos=re".*comp.*", Degree=_] }
            commands {N.Degree=Cmp}}

        rule superlatif {
            pattern { N[xpos=re".*sup.*", Degree=_] }
            commands {N.Degree=Sup}}

        %négation
        rule neg {
            pattern { N[xpos=re".*neg.*", Polarity=_] }
            commands {N.Polarity=Neg}}
}

strat pct2udmorph {Onf(Morph)}
```

Annexe 2.3 : Conversions relations syntaxiques ParCoTrain→SUD

```
package ParSUD {
        rule DepVAdv {
            pattern { e: A-[DepVAdv]-> B}
            commands { del_edge e ; add_edge A-[mod]->B}}

        rule apposition {
```

```
        pattern { e: A-[Ap]->B}
        commands { del_edge e ; add_edge A-[appos]->B}}

    rule ComplNum {
        pattern { e: NUM-[ComplNum]->NOUN}
        commands {del_edge e; shift NUM ==> NOUN; add_edge
NOUN-[nummod]->NUM}}

    rule AuxV{
        pattern { e: VERB-[AuxV]->AUX}
        commands {del_edge e; shift VERB==> AUX; add_edge
AUX-[comp:aux]->VERB}}

    rule ComplPrep {
        pattern { e: A-[ComplPrep]-> B}
        commands { del_edge e ; add_edge A-[comp:obj]-> B}}

    rule Ponct {
        pattern { e: A-[Ponct]->PUNCT}
        commands { del_edge e ; add_edge A-[punct]-> PUNCT}}

    rule DepAdjAdv {
        pattern { e: A-[DepAdjAdv]-> B}
        commands { del_edge e ; add_edge A-[mod]-> B}}

    rule DepAdjCas {
        pattern { e: A-[DepAdjCas]-> B}
        commands { del_edge e ; add_edge A-[mod]-> B}}

    rule DepAdjPrep {
        pattern { e: A-[DepAdjPrep]-> B}
        commands { del_edge e ; add_edge A-[mod]-> B}}

    rule DepAdvAdv {
```

```
        pattern { e: A-[DepAdvAdv]-> B}
        commands { del_edge e ; add_edge A-[mod]-> B}}

rule DepAdvCas {
        pattern { e: A-[DepAdvCas]-> B ; B [Case=Gen]}
        commands { del_edge e ; shift A==>B ; add_edge B-[det@numgov]-> A}}

rule DepAdvCa2s {
        pattern { e: A-[DepAdvCas]-> B }
        commands { del_edge e ; add_edge A-[mod]-> B}}

rule DepAdvPrep {
        pattern { e: A-[DepAdvPrep]-> B}
        commands { del_edge e ; add_edge A-[mod]-> B}}

rule DepNAdj {
        pattern { e: A-[DepNAdj]-> B}
        commands { del_edge e ; add_edge A-[mod]-> B}}

rule DepNCas {
        pattern { e: A-[DepNCas]-> B}
        commands { del_edge e ; add_edge A-[mod]-> B}}



rule DepNPrep {
        pattern { e: A-[DepNPrep]-> B}
        commands { del_edge e ; add_edge A-[mod]-> B}}

rule DepVCas {
        pattern { e: A-[DepVCas]-> B}
        commands { del_edge e ; add_edge A-[mod]-> B}}

rule DepVPart{
```

```
        pattern { e: A-[DepVPart]-> B}
        commands { del_edge e ; add_edge A-[mod]-> B}}

rule DepVInf{
        pattern { e: A-[DepVInf]-> B}
        commands { del_edge e ; add_edge A-[comp:obj@x]-> B}}

rule DepVPrep{
        pattern { e: A-[DepVPrep]-> B}
        commands { del_edge e ; add_edge A-[mod]-> B}}

rule Emph{
        pattern { e: A-[Emph]-> B}
        commands { del_edge e ; add_edge A-[mod@emph]-> B}}

rule ExtraPred{
        pattern { e: A-[ExtraPred]-> B}
        commands { del_edge e ; add_edge A-[mod]-> B}}

rule Interrog{
        pattern { e: A-[Interrog]-> B}
        commands { del_edge e ; add_edge A-[discourse]-> B}}



rule Juxt{
        pattern { e: A-[Juxt]-> B}
        commands { del_edge e ; add_edge A-[parataxis]-> B}}

rule Neg{
        pattern { e: A-[Neg]-> B}
        commands { del_edge e ; add_edge A-[mod]-> B}}

rule ObjDir{
```

```
        pattern { e: A-[ObjDir]-> B}
        commands { del_edge e ; add_edge A-[comp:obj]-> B}}

    rule ObjIndirCas{
        pattern { e: A-[ObjIndirCas]-> B}
        commands { del_edge e ; add_edge A-[comp:obl]-> B}}

    rule ObjIndirPrep{
        pattern { e: A-[ObjIndirPrep]-> B}
        commands { del_edge e ; add_edge A-[comp:obl]-> B}}

    rule Sub {
        pattern {e : A -[Sub]-> B ; A[upos=VERB] ;
B-[PredSub|PredCompletive]->C}
      commands { del_edge e ; shift A ==> B ; add_edge B-[comp:obj]->A}}

    rule Sub2 {
        pattern {e : A -[Sub]-> B }
      commands { del_edge e ; add_edge A-[mod]->B}}

    rule PredCompletive{
        pattern { e: A-[PredCompletive]-> B}
        commands { del_edge e ; add_edge A-[comp:obj]-> B}}


    rule PredRel{
        pattern { e: A-[PredRel]-> B}
        commands { del_edge e ; add_edge A-[mod:relcl]-> B}}

    rule PredSub{
        pattern { e: A-[PredSub]-> B}
        commands { del_edge e ; add_edge A-[comp:obj]-> B}}

    rule PredicAdv{
```

```
        pattern { e: A-[PredicAdv]-> B}
        commands { del_edge e ; add_edge A-[comp:pred]-> B}}

rule PredicComplObj{
        pattern { e: A-[PredicComplObj]-> B}
        commands { del_edge e ; add_edge A-[comp:pred]-> B}}

rule PredicComplSuj{
        pattern { e: A-[PredicComplSuj]-> B}
        commands { del_edge e ; add_edge A-[comp:pred]-> B}}

rule PredicNom{
        pattern { e: A-[PredicNom]-> B}
        commands { del_edge e ; add_edge A-[comp:pred]-> B}}

rule PredicOpt{
        pattern { e: A-[PredicOpt]-> B}
        commands { del_edge e ; add_edge A-[mod]-> B}}

rule Ref{
        pattern { e: A-[Ref]-> B}
        commands { del_edge e ; add_edge A-[comp@pass]-> B}}



rule Root {
        pattern { e: A-[Root]-> B}
        commands { del_edge e ; add_edge A-[root]-> B}}

rule Suj {
        pattern { e: A-[Suj]-> B}
        commands { del_edge e ; add_edge A-[subj]-> B}}

rule SujLog{
```

```
        pattern { e: A-[SujLog]-> B}
        commands { del_edge e ; add_edge A-[subj]-> B}}

    rule ConjCoord{
        pattern { e: A-[ConjCoord]->B;  f : B-[Coord]->C }
        commands {del_edge e;del_edge f; add_edge A-[conj]->C; shift B ==> C;
add_edge C-[cc]->B}}

    rule Coord2{
        pattern { e: A-[Coord]->B;  f : B-[Coord]->C }
        commands {del_edge e; del_edge f; add_edge A-[conj]->B; add_edge
A-[conj]->C}}

    rule Coord{
        pattern { e: A-[Coord]->B }
        commands {del_edge e; add_edge A-[conj]->B}}

    rule Correl{
        pattern { e: A-[Correl]->B }
        commands {del_edge e; add_edge A-[mod]->B}}

    rule DepExOrphan {
        pattern { e : N -[re"DepEx_.*"]-> M}
      commands {del_edge e ; add_edge N-[orphan]->M}}

    rule Polylex {
        pattern {e : A -[Polylex]-> B ; f : B-[Polylex]-> C}
      commands { del_edge e ; del_edge f ; add_edge A-[comp@fixed]->B ;
add_edge A-[comp@fixed]->C}}

    rule Polylex2 {
        pattern {e : A -[Polylex]-> B }
      commands { del_edge e ; add_edge A-[comp@fixed]->B}}
}
```

strat pct2ParSUD {Onf(ParSUD)}

Annexe 2.4 : Conversions relations syntaxiques SUD→UD

```
package SUDUD {

    rule compaux {
        pattern { e: V_aux-[comp:aux]->V}
        commands {del_edge e; shift V_aux ==> V; add_edge V-[aux]->V_aux}}

    rule nummod {
        pattern { e: A-[nummod]-> B}
        commands { del_edge e ; add_edge A-[nummod:gov]-> B}}

    rule compobjcase {
        pattern { e: A-[comp:obj]->B; A [upos=ADP]}
        commands {del_edge e; shift A ==> B; add_edge B-[case]->A}}

    rule correlMODsconj {
        pattern { e: A-[correl]-> B; B [upos=SCONJ]}
        commands { del_edge e ; add_edge A-[advcl]-> B}}



    rule correlMODdep {
        pattern { e: A-[correl]-> B}
        commands { del_edge e ; add_edge A-[dep]-> B}}

    rule DepAdjAdvMOD {
        pattern { e: A-[mod]->B; A [upos=ADJ] ; B[upos=ADV]}
        commands { del_edge e; add_edge A-[advmod]->B}}

    rule DepAdjCasMOD {
        pattern { e: A-[mod]->B; A [upos=ADJ] ; B[upos=NOUN]}
```

```
        commands { del_edge e; add_edge A-[obl]->B}}

    rule DepAdjPrepMOD {
        pattern { e: A-[mod]->B; A [upos=ADJ]; B [upos=ADP] ; f: C-[case]->B; C
[upos=NOUN]}
        commands { del_edge e; add_edge A-[obl]->C}}

    rule DepAdvAdvMOD {
        pattern { e: A-[mod]->B; A [upos=ADV] ; B[upos=ADV]}
        commands { del_edge e; add_edge A-[advmod]->B}}

    rule DapAdvPrepMODOBL{
        pattern { e: A-[mod]->B; A [upos=ADV]; B [upos=ADP]; f : C-[case]->B; C
[upos=NOUN]}
        commands {del_edge e; add_edge A-[obl]->C}}

    rule DepNAdjMOD {
        pattern { e: A-[mod]->B; A [upos=NOUN] ; B[upos=ADJ]}
        commands { del_edge e; add_edge A-[amod]->B}}

    rule DepNCasMOD {
        pattern { e: A-[mod]->B; A [upos=NOUN] ; B[upos=NOUN]}
        commands { del_edge e; add_edge A-[obl]->B}}

    rule DepNPrepMOD {
        pattern { e: A-[mod]-> B; A[upos=NOUN]; B[upos=ADP]; f: C-[case]->B ;
C[upos=NOUN]}
        commands { del_edge e ; add_edge A-[obl]-> C}}

    rule DepVCasMOD {
        pattern { e: A-[mod]->B; A [upos=VERB] ; B[upos=NOUN]}
        commands { del_edge e; add_edge A-[obl]->B}}

    rule DepVAdvMOD {
        pattern { e: A-[mod]->B; A [upos=VERB] ; B[upos=ADV]}
        commands { del_edge e; add_edge A-[advmod]->B}}
```

```
rule DepVinfCOMP {
        pattern { e: A-[comp:obj@x]->B }
        commands { del_edge e; add_edge A-[xcomp]->B}}

rule DepVPartMOD {
        pattern { e: A-[mod]->B ; A[upos=VERB]; B[upos=VERB]}
        commands { del_edge e; add_edge A-[advcl]->B}}

rule DepVPrepMOD {
        pattern { e: A-[mod]-> B; A[upos=VERB]; f: C-[case]->B}
        commands { del_edge e ; add_edge A-[obl]-> C}}

rule NegEtExtraPredMOD {
        pattern { e: A-[mod]->B; B[upos=PART|ADV]}
        commands { del_edge e; add_edge A-[advmod]->B}}

rule ObjIndirPrepCOMPOBL {
        pattern { e: A-[comp:obl]->B; A [upos=VERB] ; B[upos=ADP]; f: C-[case]->B}
        commands { del_edge e; add_edge A-[iobj]->C}}

rule ObjIndirCasCOMPOBL {
        pattern { e: A-[comp:obl]->B; A [upos=VERB] ; B[upos=NOUN]}
        commands { del_edge e; add_edge A-[iobj]->B}}

rule PredRelMODRELCL{
        pattern { e: A-[mod:relcl]-> B}
        commands { del_edge e; add_edge A-[acl]->B}}

rule RefCompPass {
        pattern { e: A-[comp@pass]-> B}
        commands { del_edge e; add_edge A-[expl]->B}}

rule Polylex {
        pattern {e : A -[comp@fixed]-> B }
        commands { del_edge e ; add_edge A-[fixed]->B}}

rule COMPPRED {
```

```
        pattern {e: A-[comp:pred]-> B}
        without {A [lemma = "biti"]}
        commands {del_edge e ; add_edge A-[xcomp]->B}}

rule PredictAdvCOMP {
    pattern {e: A-[comp:pred]->B; f: R-[root]->A ; A[lemma="biti"]}
    commands {del_edge e; add_edge B-[cop]->A; del_edge f; add_edge R-[root]->B}}

rule PredicNomCOMP {
    pattern {e: A-[comp:pred]->B; f: R-[root]->A ; A[lemma = "biti"]}
    commands {del_edge e; add_edge B-[cop]->A; del_edge f; add_edge R-[root]->B}}

rule DepAdvCasNUMGOV {
        pattern { e: A-[det@numgov]-> B }
        commands { del_edge e ; add_edge A-[det:numgov]-> B}}

rule DepAdvCasMOD {
        pattern { e: A-[mod]-> B }
        commands { del_edge e ; add_edge A-[obl]-> B}}

rule SubjSCONJ {
        pattern { e: A-[subj]-> B ; B [upos=SCONJ , lemma =  "da"| "što" | "kad" ] }
        commands { del_edge e ; add_edge A-[csubj]-> B}}

rule SubjVINF {
        pattern { e: A-[subj]-> B ; B [VerbForm=Inf] }
        commands { del_edge e ; add_edge A-[csubj]-> B}}

rule SubMOD {
        pattern { e: A-[mod]-> B ; A [upos=VERB] ; B [upos=SCONJ] }
        commands { del_edge e ; add_edge A-[advcl]-> B}}

rule SubMOD2 {
        pattern { e: A-[comp:obj]-> B ; A [upos=VERB] ; B [upos=SCONJ] }
        commands { del_edge e ; add_edge A-[ccomp]-> B}}

rule PreSubCOMP{
```

```
            pattern { e: A-[comp:obj]-> B ; B [upos=VERB] ; A [upos=SCONJ] }
            commands { del_edge e ; add_edge A-[advcl]-> B}}

        rule PredPercontCOMP{
            pattern {e : A -[comp:obj]-> B ; A[upos=VERB] ; B [upos=VERB] }
            commands { del_edge e ; add_edge A-[ccomp]->B} }

        rule PredCompletiveCOMP{
            pattern {e : A -[comp:obj]-> B ; A[upos=SCONJ] ; B [upos=VERB] }
            commands { del_edge e ; add_edge A-[ccomp]->B}}

        rule ObjDirCOMP{
            pattern {e : A -[comp:obj]-> B ; A[upos=VERB] }
            commands { del_edge e ; add_edge A-[ccomp]->B}}

        rule EmphMOD {
            pattern { e: A-[mod@emph]-> B}
            commands { del_edge e ; add_edge A-[advmod:emph]-> B}}

        rule subjNsubj {
            pattern { e: A-[subj]-> B}
            commands { del_edge e ; add_edge A-[nsubj]-> B}}

        rule modADVMOD {
            pattern {e: A-[mod]->B }
            commands {del_edge e; add_edge A-[advmod]->B}}
}
strat pct2SUDUD {Onf(SUDUD)}
```

Annexe 3 : Scripts python

Annexe 3.1 : Nettoyage des fichiers

```python
import sys, re

liste = []


for ligne in sys.stdin:
    ligne = ligne.rstrip("\n")
    if "\t" in ligne :
        liste = ligne.split("\t")
        if len(liste) == 10:
            liste[8] = re.sub(r"\d+", r"_", liste[8])
            #"\t".join(liste)
            print("\t".join(liste))

    else :
        print(ligne)
```

Annexe 3.2 : Découpage des fichiers

```python
import sys

compte = 0

for ligne in sys.stdin:
    ligne = ligne.rstrip("\n")
    if "\t" in ligne :
        liste = ligne.split("\t")
        if len(liste) == 10:
            if compte >= 0 and compte <= 3400 :
                print("\t".join(liste))
    else :
        if compte >= 0 and compte <= 3400 :
            print(ligne)
        compte += 1
        truc = []
```

Annexe 3.3 : Ajout des features

```python
import sys, re

liste = []

for ligne in sys.stdin:
    ligne = ligne.rstrip("\n")
    if "\t" in ligne :
        liste = ligne.split("\t")
        if len(liste) == 10:
            liste[5] = re.sub(r"_", r"Gender=_|Case=_
            |Number=_|Degree=_|Polarity=_|Person=_
            |VerbForm=_|Mood=_|Tense=_|Voice=_", liste[8])
            print("\t".join(liste))

    else :
        print(ligne)
```

Annexe 4 : Lignes de commande dans le terminal linux

Annexe 4.1 : Conversion des POS

```
m1tal@LITL31:~/Documents/memoire$ grew transform -i dev2.conll  -o dev2POS.conll
 -grs POS.grs -strat pct2udpos
```

Annexe 4.2 : Ajouts des features

```
m1tal@LITL31:~/Documents/memoire/finFeatures$ python3 Features.py < dev2POS.conl
l > dev2Featfin.conll
```

Annexe 4.3 : Conversion des traits morphologiques

```
m1tal@LITL31:~/Documents/memoire/finFeatures$ grew transform -i dev2Featfin.conl
l  -o dev2Morph.conll -grs Traitsmorpho.grs -strat pct2udmorph
```

Annexe 4.4 : Conversion des relations ParCoTrain → SUD

```
m1tal@LITL31:~/Documents/memoire/finSUD$ grew transform -i dev2Morph.conll  -o d
ev2SUD.conll -grs Par_SUD.grs -strat pct2ParSUD
```

Annexe 4.5 : Conversion des relations SUD → UD

```
m1tal@LITL31:~/Documents/memoire/finSUD$ grew transform -i dev2SUD.conll  -o dev
2UD.conll -grs SUD_UD.grs -strat pct2SUDUD
```

Déclaration sur l'honneur de non-plagiat

Je soussigné.e,

Nom, Prénom : Caule, Clémence
Régulièrement inscrit.e à l'Université de Toulouse II Jean Jaurès
N° étudiant : 22100792

Année universitaire : 2021-2022

certifie que le document joint à la présente déclaration est un travail original, que je n'ai ni recopié ni utilisé des idées ou des formulations tirées d'un ouvrage, article ou mémoire, en version imprimée ou électronique, sans mentionner précisément leur origine et que les citations intégrales sont signalées entre guillemets.

Fait à : Mimizan
Le : 10/06/2022

Signature :