

Université de Montréal

**Extraction des patrons de régime de VerbNet pour une
implémentation dans un système de génération
automatique de texte**

par

Daniel Galarreta-Piquette

Département de traduction et linguistique

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en linguistique

28 février 2018

SOMMAIRE

Ce mémoire explore les patrons de régime en anglais provenant de la ressource lexicographique *VerbNet* et leur implémentation dans un système de génération automatique de texte.

SUMMARY

English summary and keywords. . .

TABLE DES MATIÈRES

Sommaire	iii
Summary	v
Liste des tableaux	xi
Liste des figures	xiii
Liste des sigles et des abréviations	xv
Dédicaces	xvii
Remerciements	xix
Introduction	1
Chapitre 1. Extraction de l’architecture de <i>Verbnet</i>	3
1.1. VerbNet	4
1.1.1. Levin et l’organisation en classes verbales	5
1.1.2. Composantes de VerbNet	8
1.1.2.1. Classes verbales : organisation hiérarchique	8
1.1.2.2. Membres	10
1.1.2.3. Rôles thématiques	10
1.1.2.4. Restrictions sélectionnelles	12
1.1.2.5. Cadres syntaxiques	13
1.1.2.6. Prédicats sémantiques	14
1.1.3. Brèves descriptions de dictionnaires concurrents utilisés en NLP	16

1.1.3.1.	WordNet	16
1.1.3.2.	FrameNet	17
1.1.3.3.	XTAG	18
1.1.3.4.	Dorr's LCS database	19
1.1.3.5.	Comlex	20
1.1.3.6.	DDO	20
1.1.3.7.	LEFFF	20
1.1.4.	textes pertinents ?	20
1.1.5.	Utilisation de VerbNet dans des applications NLP	21
1.1.6.	Pourquoi on n'a pas utilisé les rôles thématiques et les prédicats sémantiques	21
1.1.7.	Évaluation du système	21
1.1.8.	Pourquoi on a décidé d'utiliser VerbNet ?	21
1.2.	Python	21
1.2.1.	Création du dictionnaire de verbe : <code>lexicon.dict</code>	22
1.2.2.	Création du dictionnaire de patron de régimes	25
1.2.3.	Extraction des membres des classes verbales pour enrichir le dictionnaire	27
1.2.4.	Scripts pour faire les tests	28
1.2.4.1.	Extraction des exemples	28
1.2.4.2.	Création des structures qui serviront de tests	28
Chapitre 2.	Génération automatique de texte	29
2.1.	Mécanisme <code>dsynt=created->constrained->OK</code> pour générer un énoncé simple (sujet, verbe, objet)	29
2.1.1.	Application de la règle <code>root_standard</code>	29
2.1.2.	application de la règle <code>lex_standard</code> pour lexicaliser le verbe principal	29
2.1.3.	Application de la règle <code>actant_gp_selection</code>	30

2.1.4.	application des règles actancielles	30
2.1.5.	application de la règle constraints_gp	30
2.1.6.	application des règles de lexicalisation	31
2.1.6.1.	lex_standard	31
2.1.7.	Avantages d'utiliser ces mécanismes	32
2.2.	Énumérations	32
2.3.	Équations mathématiques	32
2.4.	Définitions, théorèmes et preuves	33
2.5.	Construction d'un tableau	33
2.6.	Référence à une entrée bibliographique	34
2.7.	Insertion de figures	34
Bibliographie		-i
Annexe A. Le titre		A-i
A.1.	Section un de l'annexe A	A-i
Annexe B. Le titre2		B-i

LISTE DES TABLEAUX

2. I	Un tableau simple dans le second chapitre.....	33
A. I	Titre alternatif pour la table des matières.....	A-i

LISTE DES FIGURES

1.1	prédicats sémantiques	15
2.1	Un cercle.....	34
2.2	Un carré et un triangle.....	34

LISTE DES SIGLES ET DES ABRÉVIATIONS

GAT	Génération automatique de texte
GP	Patron de régime, de l'anglais <i>Government Pattern</i>
DPOS	Partie du discours profonde, de l'anglais <i>Deep Part of Speech</i>
TST	Théorie Sens-Texte
VN	<i>VerbNet</i>

DÉDICACES

Vos dédicaces.

REMERCIEMENTS

Remerciements. . .

INTRODUCTION

Parler de la génération de texte automatique, de *VerbNet*, de la problématique (pas de consensus quant à l'architecture de la classe verbale en TAL) de VerbNet, des dictionnaires. Pourquoi les verbes sont si importants ? (Kipper, 2005, dissertation) Puisque les verbes sont porteurs du sens principal de la phrase, il faudrait donc créer une ressource qui puisse rendre compte du sens des verbes si on souhaite un bon fonctionnement des applications NLP.

Chapitre 1

EXTRACTION DE L'ARCHITECTURE DE *VERBNET*

Dans ce chapitre, nous verrons l'apport que la ressource lexicale *VerbNet* peut offrir à des applications en traitement automatique du langage (TAL). Nous avons comme objectif d'extraire l'architecture de dictionnaire *VerbNet* pour l'implémenter dans un dictionnaire de type sens-texte qui servira à générer du texte. Cet objectif provient d'une problématique que nous avons rencontré auparavant. Nous voulions savoir comment organiser notre dictionnaire en ce qui concernait les verbes. Car ceux-ci sont si riches et complexes qu'il nous fallait trouver un moyen systématique d'encoder cette partie du discours. De plus, il nous fallait l'encoder pour que notre nouveau dictionnaire puisse interagir correctement avec les règles grammaticales du logiciel. En ce qui concerne les dictionnaires, parmi la communauté TAL, il ne semble pas y avoir de consensus quant à la manière de procéder pour modéliser la classe verbale. La raison est simple, les verbes démontrent des comportements variables, très riches au niveau de l'éventail de patrons de régime possibles pour un même verbe, et assez complexes ce qui nécessite beaucoup plus d'attention que d'autres parties du discours comme les noms qui démontrent beaucoup moins de variétés d'usage quant au nombre de patrons de régime. Ce qui fait en sorte que comme tous les verbes sont des prédicats, et que les prédicats sont les noyaux des énoncés, il faut les traiter avec soin, faute de quoi leur application en NLP sera médiocre. Cette problématique nous a donc amené à constater que *VerbNet* s'était penché sur ce problème et nous voulions vérifier si rouages de leur dictionnaire peuvent s'appliquer en génération automatique de texte. De nos jours, avec les modèles stochastiques où il n'y a pas d'analyse linguistique, on ne construit pas de dictionnaires ou de règles grammaticales, mais on le laisse le soin au système d'apprendre les règles par lui-même et de développer le lexique par lui-même en n'assignant pas de POS et en voyant les langues naturelles comme des suites de caractères. C'est une mode qui fonctionne présentement grâce à la quantité immense d'information qu'on retrouve et ces

nouveaux corpus incroyables combinés avec la puissance des ordinateurs et l'apprentissage machine, certains font complètement fi de l'analyse linguistique dans leurs applications TAL. Toutefois, les utilités principales de ces systèmes sont de divers ordres de NLP, mais en ce qui concerne la génération automatique de texte, le traitement de la langue se doit d'être impeccable (Lareau, Lambrey, Dubinskaite, Galarreta-Piquette et Nejat, 2018). D'où la nécessité de développer des outils puissants et rigoureux qui nécessiteront plus de temps à développer car il faudra créer les règles et enrichir le dictionnaire qui compose la langue. Pour ce faire, on doit recourir à des méthodes linguistiques qui étaient plus populaires dans la fin des années 90 et début 2000 (citer des systèmes). Toutefois, en étant bien conscient du changement de cap, nous pensons qu'il est encore primordial de développer de bons outils linguistiques computationnels car c'est de cette manière qu'on pourra le mieux représenter le fonctionnement de la langue. De plus, nombreux chercheurs combinent l'apprentissage machine à des règles et des dictionnaires pour ainsi développer le lexique partiellement en annotant automatiquement des lexèmes. (trouver une citation dans les INLG de 2014 à 2017)

1.1. VERBNET

Ainsi, tel que mentionné précédemment, VerbNet a été créé dans un contexte où il y avait un réel besoin de réfléchir à la meilleure manière de procéder pour construire un dictionnaire qui saura tenir compte de la richesse et la complexité que renferment les verbes (Kipper, Dang et Palmer, 2000). C'était en réponse au manque de lignes directrices sur l'organisation des verbes dans les dictionnaires destinés à des applications TAL et ce malgré la quantité impressionnante de dictionnaires computationnels existants déjà. Parmi ceux qui ont tenté la chose, nous nommerons : le generative lexicon de Pustojevsky (Pustejovsky, 1991) WordNet de Fillmore (Miller, 1995), ComLex (Grishman, Macleod et Meyers, 1994), LCS (Ayan et Dorr, 2002) et FrameNet (Baker, Fillmore et Lowe, 1998). Ceux-ci comportaient des lacunes que du point de vue du traitement des entrées lexicales verbales, selon les auteurs de VerbNet. Les lacunes étaient diverses mais importantes, par exemple, ils trouvaient que le Generative Lexicon ne focusait que sur les noms, que WordNet ne donnait pas de détails concernant les cadres syntaxiques possibles et aucune mention de la structure prédicat-argument. Quant à ComLex, ils décrivaient effectivement les cadres syntaxiques possibles, mais ne faisait pas de désambiguïsation sémantique. Finalement, ils jugeaient que le lexicon LCS a essayé de palier à ces lacunes aussi, mais l'étendue lexicale qu'il offrait n'était pas assez large à leur goût (Schuler, 2005). C'est donc dans ce contexte qu'est né le projet VerbNet.

1.1.1. Levin et l'organisation en classes verbales

Levin a bâti un lexicon verbal de l'anglais. Ses recherches portaient sur les propriétés sémantiques et syntaxiques des verbes. Selon elle, les composantes sémantiques des verbes déterminent la manière dont ils se combinent avec leurs arguments. Elle a tenté de répertorier et systématiser les divers cadres syntaxiques des verbes. Puisque les verbes forment les noyaux des énoncés et qu'ils sont prédicats, Levin y a vu un intérêt académique. De plus, les verbes démontrent des propriétés extrêmement complexes contrairement aux autres prédicats se trouvant dans les autres PDD. Il existe un éventail de combinaisons possibles pour un verbe avec ses arguments et tout locuteur natif d'une langue sait quels sont les diathèses possibles et les arguments possibles pour un verbe. Et ce, sans avoir de connaissances linguistiques au préalable, il s'agit d'une intuition de locuteur natif. Pour cette raison, Levin voulait regrouper les verbes en classes verbales en se basant sur son intuition de locutrice native. Ainsi, en testant divers patrons de régime des groupes de verbes, elle a délimiter un nombre impressionnant de classes verbales où chaque verbe compris dans cette classe partage des cadres syntaxiques avec les autres membres de sa classe.

Tel que Levin le dit ([Levin, 1993](#)), son travail a été guidé par l'idée que les comportements syntaxiques des verbes sont motivés par les composantes sémantiques qui les représente. À ces fins, Levin a observé les comportements des verbes en anglais pour y tester cette hypothèse linguistique car il pensait que c'était une bonne manière de démontrer la chose. Si des verbes qui partagent le même type de comportement syntaxique, ils doivent aussi probablement partager des caractéristiques sémantiques. Son travail a été de classer les verbes systématiquement. Déjà, Levin soulignait l'importance d'encadrer les verbes dans les lexicons car les verbes qui sélectionne des arguments démontrent un ensemble complexes de propriétés. Ainsi, au lieu de construire un dictionnaire où chaque entrée verbale est prise individuellement, on regroupe les entrées lexicales en classe en fonction de leur comportement syntaxique et sémantique.

À la base le traitement est fait en fonction du comportement syntaxique, puis selon Levin, les verbes qui partagent le même type d'alternations syntaxiques (patrons de régime) partagent aussi la sémantique sous-jacente à ces constructions, donc cela fait en sorte qu'ils partagent généralement les mêmes caractéristiques sémantiques aussi. Le fait d'organiser le dictionnaire en classe verbale accélère le processus de création du dictionnaire, car une fois qu'on détermine le nombre de verbes qui fonctionnent de la même manière, il ne reste plus qu'à faire une entrée qui retient l'information syntaxique et sémantique partagée par l'ensemble de la classe, et passer à la classe suivante. Après, lorsqu'on a fait le tour des verbes, on a juste à les ajouter à l'une des classes pré-existantes. Les créateurs de VerbNet

sont conscients du fait qu'on va parfois tenter de faire rentrer un verbe dans une classe et que c'est un brin tirer par les cheveux, mais il s'agit d'une infime partie du dictionnaire. (Schuler, 2005).

Ici, on voit que les verbes spray/load peuvent exprimer leurs arguments d'au moins deux manières différentes. La seconde manière que Levin a appelé "locative alternation".

(1) Cadre syntaxique : Spray

- a. Sharon sprayed water on the plants.
- b. Sharon sprayed the plants with water.

(2) Cadre syntaxique : Load

- a. The farmer loaded apples into the cart.
- b. The farmer loaded the cart with apples.

Ces mêmes locuteurs natifs savent que des verbes qui s'apparentent à Spray/Load (car ils semblent correspondre au même type d'évènement) ne permettent pas les 2 alternances démontrées en (1) et en (2). Les verbes fill/cover permettent l'option b) et les verbes dump/pour permettent l'option a). Ainsi, à partir de tests de ce type, pour voir quels verbes permettaient le même type de construction, elle a bâti un lexicon qui regroupe les verbes en fonctions des alternances de diathèses qu'ils permettent.

(3) Alternances de diathèses : cover

- a. *Monica covered a blanket over the baby.
- b. Monica covered the baby with a blanket.

(4) Alternances de diathèses : fill

- a. *Gina filled lemonade into the pitcher.
- b. Gina filled the pitcher with lemonade.

(5) alternances de diathèses : pour

- a. Carla poured lemonade into the pitcher.
- b. *Carla poured the pitcher with lemonade

(6) alternances de diathèses : dump

- a. The farmer dumped apples into the cart.
- b. *The farmer dumped the cart with apples.

On en vient ainsi à la conclusion, que ces verbes ont beau ressembler à Spray/Load, mais il ne permettent pas les mêmes cadres syntaxiques, donc il y a une composante sémantique qui fait défaut et on ne peut pas avoir les mêmes frames. Ainsi, en opposant diverses diathèses possibles, Levin a fait un travail colossal où elle a repertorié des verbes se combinant de la même manière. C'est pourquoi des verbes qui ont beau être des synonymes, ne se combinent pas de la même manière. Elle avait ainsi un éventail de diathèses à tester et elle regroupait en classe les verbes qui partageaient les mêmes alternances. Se justifiant ainsi en disant que les raisons qui sous-tendent un partage des diathèses proviennent d'une sémantique similaire.

Remettre l'exemple de VerbNet (plus clair) :

(cette partie manque l'exemple venant de verbnet)Ainsi, tel que démontré en ?? et ?? les verbes appartenant à la classe break et à la classe cut se ressemblent car ils peuvent tous les 2 prendre ces cadres syntaxiques (ce genre de construction syntaxique). Toutefois, en ?? et en ??, on voit très bien qu'il ne partagent pas ces constructions syntaxiques. *Break* prend seulement la construction intransitive et exclut l'autre, tandis que *cut* prend la construction conative et exclut l'intransitive. La raison que Levin donne est la suivante. Le verbe *cut* décrit une série d'actions ciblant la complétion d'un but (séparer un objet en morceau). Toutefois, il est possible de faire ces séries d'actions sans que l'objectif final ne soit atteint, mais l'action de couper peut quand même être perçue. En ce qui concerne *break*, la seule chose qui importe dans l'évènement, c'est le changement d'état d'un objet (qui devient séparés en morceaux). Si on n'arrive pas au résultat, une tentative de briser quelque chose ne peut être perçue. Ce qu'on peut tirer de cet exemple, est que les classes verbales regroupent des verbes qui partagent des comportements syntaxiques similaires, les membres des classes ne sont donc pas nécessairement des synonymes, il ne s'agit que de verbes qui s'utilisent de la même façon. Ainsi, certaines classes vont effectivement regrouper des verbes qui signifient à peu près la même chose, mais aussi des verbes qui en surface, ne partagent pas bcp avec une majorité des verbes de cette classe, mais syntaxiquement, ces membres se comportent comme le reste de la classe. Ainsi, selon Levin, il y a quelque chose derrière les composantes sémantiques de ces membres qui les unirait avec les autres qui se ressemblent.

Ainsi, les créateurs de VerbNet ont repris les concepts de Levin, mais les ont retravaillés. Toutefois, l'idée de construire le lexicon verbal en classes est directement hérité de Levin. L'organisation hiérarchique est aussi directement inspiré de la hiérarchie que Levin avait pensé. Toutefois, VerbNet a aussi ajouté des classes à ce que Levin avait fait. Ce qu'ils appellent intersective Levin classes, ce sont des sous-ensembles de classes qui s'entrecoupent et forment des classes à part. (Schuler, 2005)et (Kipper, Korhonen, Ryant et Palmer, 2006).

1.1.2. Composantes de VerbNet

VerbNet est composé de classes verbales. Chaque classe contient un ensemble de membres, une liste de rôles thématiques (accompagnés de restrictions) utilisés pour les cadres syntaxiques et sémantiques, puis un ensemble de cadres possibles pour une classe. Chaque cadre est composé d'une brève description, d'un exemple, d'une description syntaxique et d'un ensemble de prédicats décrivant le cadre (Schuler, 2005).

1.1.2.1. *Classes verbales : organisation hiérarchique*

Les raisons qui ont motivé VerbNet à organiser son information en hiérarchie. D'abord, il se sont fortement inspiré de Acquilex Lexical Knowledge Base (Copestake, 1992). Ensuite, pour des raisons pratiques, ils ont trouvé que c'était la meilleure manière d'organiser cette montagne d'information lexicale car ils ont revu l'information fournie par Levin et ont ajouté des sous-classes aux classes ainsi que des sous-sous-classes aux sous-classes, allant jusqu'à un niveau de 3 steps de profondeur. Donc, des sous-ensembles des sous-ensembles. Ce qu'ils considèrent comme un raffinement des classes originales de Levin (Schuler, 2005). En gros, c'est l'ajout de sous-classes aux classes originales de Levin (et aux sous-sous-classes) qui ont rendu VerbNet hiérarchique.

Le fonctionnement est assez simple. Une sous-classe fille hérite de toute l'information de sa classe mère. Les sous-classes sont à l'origine de vouloir spécifier le comportement qui rassemble un sous-ensemble des membres d'une classe. Ainsi, une sous-classe est un ajout d'information par rapport à des restrictions d'usages de rôles thématiques, de cadres syntaxiques ou de prédicats sémantiques. (guidelines, (Schuler, 2005))

Par la suite, on se sert encore de la numérotation pour expliciter la hiérarchie à l'intérieur même d'une classe (guidelines). Chaque classe peut inclure des classes filles, qui sont des classes sœurs entre elles, et qui peuvent avoir des classes filles à leur tour. La classe verbale Spray en démontre bien la chose.

D'abord, il y a la classe supérieure, qui est la plus haute de la hiérarchie, toutes les caractéristiques de cette classe sont partagées par tous les verbes de la classe. Dans la top classe nous avons les constructions syntaxiques et les prédicats sémantiques partagés par la classe, ainsi que les rôles thématiques.

Une classe mère domine une sous-classe, toutes ses caractéristiques sont partagées avec les classes subordonnées à celle-ci.

Une sous-classe : Les sous-classes dans VerbNet héritent des caractéristiques provenant de la classe dominante, mais elles spécifient particulièrement des constructions syntaxiques

et une sémantique entre les verbes membres de cette sous-classe. Ce qui est spécifié dans une sous-classe peut être de différents ordres : ajouter des constructions syntaxiques propres à ce sous-groupe, ajouter des restrictions sélectionnelles sur les étiquettes des rôles sémantiques.

Des classes sœurs, ne partagent pas de caractéristiques hormis celle héritées par leur classe mère. Le reste de l'information encodée dans la classes sœurs n'est valide qu'à l'intérieur de leur classe respectivement.

Démontrer la chose comme dans Guidelines :

- Spray-9.7
 - Spray-9.7-1
 - * Spray-9.7-1-1
 - Spray-9.7-2

Les classes verbales de VerbNet sont numérotées par des chiffres allant de 9-109 (guidelines). Le système de numérotation est directement hérité de Levin ([Levin, 1993](#)) Ainsi, le numéro apparaissant devant une classe verbale est associé à des caractéristiques sémantiques et syntaxiques communes. Par Exemple, les classes verbales associées à des verbes de type "mettre quelque chose" commenceront par le chiffre 9. Ce qui nous donne quelque chose comme :

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

Certains numéros n'impliquent qu'une seule classe, car il n'y a pas d'autres classes qui partagent ce genre de traits sémantiques ou syntaxiques communs.

1.1.2.2. Membres

L'ensemble des membres d'une classe :

LISTING 1.1. les membres

```
<VNCLASS ID="give-13.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaces="true">
  <MEMBERS>
    <MEMBER name="deal" wn="deal%2:40:01 deal%2:40:02 deal%2:40:07 deal%2:40:06" grouping="deal.01"/>
    <MEMBER name="lend" wn="lend%2:40:00" grouping="lend.02"/>
    <MEMBER name="loan" wn="loan%2:40:00" grouping="loan.03"/>
    <MEMBER name="pass" wn="pass%2:40:00 pass%2:40:01 pass%2:40:13 pass%2:38:04" grouping="pass.04"/>
    <MEMBER name="peddle" wn="peddle%2:40:00" grouping="peddle.01"/>
    <MEMBER name="refund" wn="refund%2:40:00" grouping="refund.01"/>
    <MEMBER name="render" wn="render%2:40:02 render%2:40:01 render%2:40:00 render%2:40:03" grouping="render.05"/>
    <!--removed "trade" from class because doesn't take "to-PP"-->
    <!--removed "volunteer" from class because doesn't fit dative or-->
    <!--PP recipient PP frames-->
  </MEMBERS>
</VNCLASS>
```

1.1.2.3. Rôles thématiques

(ce que je veux, ce sont des beamer arrows qui permettent de pointer vers qqch à l'intérieur de listings, tikz style)

VerbNet utilise un ensemble de 23 rôles thématiques pour identifier les arguments dans les classes verbales (vérifier de quels rôles thématiques ils s'inspirent). On étiquette les arguments dans les classes verbales en leur associant un rôle thématique. La raison pour laquelle VerbNet a opté pour les rôles thématiques est que, contrairement à un étiquetage générique où on énumère les arguments en procédant comme "Argument 1" Verbe "Argument 2" pour illustrer un cadre syntaxique est parce que les rôles thématiques peuvent offrir de l'information sémantique de plus que juste un argument numéroté. Effectivement, à la base, les rôles thématiques ont été pensés dans les années 60 pour assigner une fonction aux arguments sélectionnés par des prédicats. On les avait créés pour typer les participants des prédicats. La spécification du rôle fournit de l'information sémantique sur le type d'argument en jeu, tandis que numéroté ne fournit rien du tout. Chaque argument se voit donné un rôle thématique unique, et ces rôles thématiques sont partagés par tous les membres d'une classe. Donc, ils doivent être assez large pour que ce soit cohérent avec tous les membres, mais pas trop imprécis non plus. Les rôles thématiques correspondent à la relation sémantique qui existe entre un prédicat et ses arguments.

VerbNet utilise les rôles thématiques pour étiquetter les arguments figurant dans les cadres syntaxiques (et sémantiques) (Schuler, 2005). Elle puise dans une banque de 23 rôles thématiques pour associer le bon rôle à l'argument en question. Ils ont choisi ces rôles car ils

étaient assez généraux pour se prêter à toutes les évènements que dégagent les verbes dans le dictionnaire. Ils voulaient capturer l'essence des verbes et démontrer encore une fois le caractère général des verbes en démontrant qu'une poignée d'argument peut bel et bien rendre compte des arguments sélectionnés par les verbes en anglais. À l'intérieur d'une même classe verbale on y retrouve un nombre x de rôles thématiques qui seront mappés aux arguments sélectionnés dans les cadres syntaxiques et sémantiques fournis par VerbNet. Ils ont choisi les rôles thématiques au lieu d'autres moyens d'étiquetage des arguments car ils trouvaient que ça ajoutait de l'information sémantique sur une classe verbale.

Les rôles thématiques choisis par VerbNet sont de divers ordres, certains proviennent des premiers pensés par (Fillmore, 1968) d'autres de Jackendoff (Jackendoff, 1972), et des ajouts pour compenser pour les classes qui nécessiteraient des rôles plus spécifiques à leur classes, tout en étant assez généraux pour pouvoir se prêter à d'autres classes. Encore une fois, tel que mentionné dans le background de Dissertation, certains critiquent les rôles thématiques car ils ont un caractère assez arbitraires et il n'est pas facile de tout justifier. De cette manière, les rôles thématiques utilisés par VerbNet n'ont pas été créés systématiquement, mais plus en fonction de ce qui cadrerait le plus avec les classes verbales et les arguments nécessaires qui apporteraient le plus d'information sémantique pertinente. Ils ne pensent pas qu'un ensemble plus petits de rôles thématiques aurait pu être suffisant, mais ils ne pensent pas non plus que ceux qu'ils ont choisi sont exactement ce dont on aurait besoin, il n'ont pas nécessairement été au bout de tous les rôles thématiques possibles. Mais, ils considèrent que ce qu'ils avaient était suffisant pour rendre compte des comportements des arguments envers leur prédicats respectifs, et ce pour toutes les classes verbales.

À l'intérieur d'une classe verbale (ou sous-classe, etc.), chaque argument se fait assigner un rôle thématique

Les rôles thématiques nous donne de l'information sur la sémantique des participants d'une phrase dans le but que les différents cadres syntaxiques (alternances) n'influencent pas les rôles assignés. On nous fourni un exemple que nous réutiliserons pour illustrer le propos (guidelines) :

- (7) a. Sandy shattered the glass.
- b. The glass shattered

D'un point de vue de la syntaxe, dans la première phrase, Sandy est le sujet du verbe et the glass en est l'objet direct. Tandis que dans la seconde phrase, the glass est le sujet du même verbe, et il n'y a pas d'objets directs. En leur assignant un rôle thématique, on notera que d'un point de vue de la sémantique rien n'a changé, même si la structure a changé en

apparence. Ainsi, si on assigne un rôle d'Agent à Sandy et un rôle de Patient à the glass, on remarque que les rôles sémantiques sont cohérents avec la phrase, malgré le changement de sujet et d'objet direct.

Les rôles thématiques sont explicités au début d'une classe verbale. (exemples venant de break-45.1.xml)

LISTING 1.2. les rôles thématiques

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

Finalement, les rôles thématiques sont domain-general, voulant dire qu'on veut les réutiliser dans le plus de classes là où c'est pertinent. Ils ne sont pas spécifiques à une classe ou un verbe, on les retrouve partout dans le lexicon. C'est pourquoi on veut qu'ils soient assez général pour rendre compte de divers type d'arguments.

Pour les besoins de notre travail, nous n'utilisons pas les rôles thématiques dans notre travail, mais nous voulions souligner qu'ils étaient importants pour les créateurs de VerbNet. Voir les raisons de Melcuk p.230

1.1.2.4. Restrictions sélectionnelles

Les restriction sélectionnelles vont sur les rôles thématiques. Est-ce que c'est pertinent d'en parler ? Probablement, pas, on ne s'en sert pas du tout. En gros, il s'agit de restrictions imposées aux rôles thématiques pour que ça sélectionne un certain type d'argument. Ainsi, parmi ces restrictions, parfois, certains verbes requièrent que l'Agent soit nécessairement de type x, tandis que le patient peut être n'importe quoi. OU l'inverse, etc. Ça donne encore plus d'informations sur le participant de la phrase, de plus, VerbNet considère que ces restrictions sélectionnelles offrent encore plus d'information sémantique, car on précise le type d'argument requis.

LISTING 1.3. les restrictions sélectionnelles

```

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

```

1.1.2.5. Cadres syntaxiques

Pour une classe donnée, on y retrouve soit un ou des cadres syntaxiques qui servent à représenter le type de réalisation de surface possible pour cette classe verbale. D'ailleurs, ces cadres syntaxiques sont partagés par l'ensemble des membres d'une classe ou d'une sous-classe. Chaque cadre syntaxique décrit une construction verbale de type transitives directes/indirectes, des intransitives, des phrases prépositionnelles, etc. Un cadre syntaxique est composé de rôles thématiques dans leur position argumentale ainsi que le verbe qui les régie (ainsi que d'autres unités lexicales nécessaires au bon fonctionnement d'une construction).

À l'intérieur de chaque classe verbale, on retrouve la liste des membres, suivie de la liste des rôles thématiques qu'on retrouve dans les cadres syntaxiques de cette classe, puis les cadres syntaxiques. Ainsi, on liste tous les cadres syntaxiques possibles pour une classe. Ce qui nous donne de l'information de nature syntaxique et explicite les liens qui unissent les rôles thématiques au verbe en question. Ça nous donne aussi beaucoup d'information concernant le verbe, car on voit comment il peut être utilisé, quel genre d'actant il sélectionne, de quel type de patron de régime il s'agit. C'est ce qui nous intéresse le plus à la base. C'est de savoir comment ce verbe se combine, peut-il prendre un verbe comme complément d'objet direct, est-ce qu'il sélectionne telle ou telle préposition. Bref, l'information contenue dans les cadres syntaxiques est très pertinente pour notre travail, car il existe très peu de dictionnaires qui ont voulu énumérer toutes les alternances syntaxiques d'un verbe. Puis, les cadres syntaxiques respectent aussi la caractéristique hiérarchique omniprésente dans VerbNet. Ainsi, tous les membres d'une classe partagent ces patrons de régime, et une sous-classe hérite aussi des patrons de régime de la classe qui la gouverne.

guidelines : Les cadres syntaxiques sont compris dans la section FRAMES de VerbNet qui contient, les cadres syntaxiques ainsi que les prédicats sémantiques. Cela nous donne une description des différentes réalisations en syntaxe de surface et des alternances de diathèses permises pour les verbes représentés par cette classe. On a ainsi une vue d'ensemble sur les constructions syntaxiques possibles par classe verbale.

Dans la figure ci-bas, on voit le début de FRAMES, qui est la balise comprenant tous les FRAME qui eux sont des cadres. À l'intérieur de FRAME, on a SYNTAX et SEMANTICS. Commençons d'abord par SYNTAX, et faisons abstraction de DESCRIPTION et EXAMPLES pour l'instant.

LISTING 1.4. cadres syntaxiques

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

1.1.2.6. *Prédicats sémantiques*

VerbNet est fier de pouvoir offrir de l'information sémantique en plus d'information syntaxique. Voici comment ils mettent leur information sémantique. Les prédicats sémantiques = dénoter les relations entre participants d'un événement et l'événement en soi. Ils sont utilisés pour transmettre les key components of meaning d'une classe verbale.

Dissertation : Les prédicats sémantiques sont utilisés pour véhiculer des composantes sémantiques importantes de chaque classe verbale. Ces prédicats sémantiques dénotent les relations entre les participants et les événements. Dans verbnet, l'information sémantique est exprimée par une conjonction de prédicats sémantiques. Ces prédicats sémantiques peuvent ...

Revoir cette section, peut-être demandé à François de la lire : c'est fait, voir la photo prise dans mon phone et essayer de la retranscrire en graphique LaTeX.

LISTING 1.5. Les prédicats sémantiques

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

Pour mieux exposer leur sémantique, nous avons fait un tableau qui exemplifie les prédicats sémantiques pour l'énoncé : give

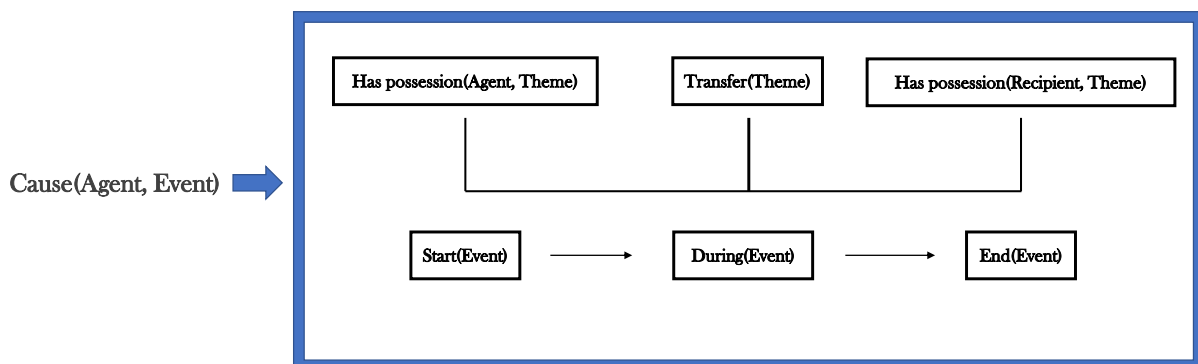


FIGURE 1.1. prédicats sémantiques

1.1.3. Brèves descriptions de dictionnaires concurrents utilisés en NLP

1.1.3.1. *WordNet*

: Wordnet est une base de données lexicale en ligne pour les verbes, noms, adjectifs et ad-
verbes de la langue anglaise. Cette base de donnée est organisée en ensemble de synonymes,
où chaque *synset* contient des mots référant à un concept donné ainsi qu'une glose défini-
tion et une phrase exemple. L'aspect hiérarchique est construit via des liens d'hyperonymie
et d'hyponymie entre les *synsets*. Dans WordNet, les verbes sont classés parmi les classes
suivantes s'il s'agit de verbes dénotant des actions ou des événements (motion, perception,
contact, communication, compétition, change, cognition, consommation, création, émotion,
perception, possession, bodily care and functions, social behavior and interactions) et des
verbes dénotant des états (des types être :resemble, belong, suffice, des verbes contrôles
(want, fail, prevent, succeed), et des verbes aspectuels (begin)) ([Fellbaum, 1998](#))

Les verbes sont groupés par ensemble de synonymes et non par classes verbales contrai-
rement à VerbNet. Le concept de synonyme n'est pas vu dans son sens typique, il s'agit
plus de : items lexicaux qu'on pourrait utiliser dans un même contexte sont vus comme des
synonymes. À la base WordNet a été conçu pour agir comme réseau sémantique et contient
ainsi peu d'information syntaxique. La ressource fournit des définitions, des exemples et des
synsets, mais ne nous donne pas d'information sur la structure sémantique ou syntaxique
des verbes, elle est sous-entendue, ce que fait que contrairement à VerbNet qui explicite en
détail chaque patron de régime et structure syntaxique. les liens entre chaque items lexicaux
sont de divers ordres : synonymes, antonymes, troponymes, entailment et causation. ([Schuler, 2005](#))

"English verbs as a semantic net" ([Fellbaum, 1998](#)) : Dans cet article, on expose le liens
sémantiques entre les verbes, ce que wordnet fait : il tisse un lien entre 2 verbes qui partagent
des composantes sémantiques : ces liens sont de divers ordres et justifié par le concept
d'entailment. Ainsi, on a les relations de hyponymie, hyperonymie, méronymie, troponymie,
etc.

D'ailleurs, VerbNet a décidé de faire un mapping de ses entrées lexicales aux entrées
lexicales de WordNet pour compléter la ressource. Chaque verbe dans VerbNet est mappé
à WordNet si c'est possible et qu'il y existe un équivalent. On peut aussi confirmer le choix
d'associer un mot à plusieurs classes verbales dans VerbNet. Ainsi, le verbe leave est associé
à Escape, future having, keep, leave et fulfill car leave peut impliquer différentes composantes
sémantiques en fonction des prédicats décrits dans ces classes. On retrouve l'équivalent dans
les synset qui sont attribués à Leave (toutefois, WordNet attribue 14 différents sens à Leave)

et VerbNet en attribue 5. Ainsi, lorsque c'est possible, les membres des classes se font assigné un WordNet ID avec.

1.1.3.2. *FrameNet*

: Le projet de Berkeley FrameNet contient de l'information sur les noms, adjectifs et verbes le tout basé sur un corpus annoté manuellement. Les sens des mots sont groupés dans des structures conceptuelles appelées des Frames qui partagent des propriétés sémantiques similaires et qui sont des représentations de concepts cognitifs. La base de données FrameNet contient des descriptions des cadres sémantiques et des items lexicaux en association avec leur représentation sémantique et syntaxique en contexte (la représentation des valences). Le projet inclut aussi des phrases exemples pour les frames décrits. Les semantic frames sont définis comme des représentations schématiques de situations impliquant des participants, propositions et d'autres rôles conceptuels. Prenons le frame de JUDGEMENT qui possède les rôles : judge, evaluatee, reason et le STATEMENT Frame avec des rôles comme SPEAKER, ADDRESSEE et MESSAGE, tel que démontré en (2) et en (3). FrameNet utilise aussi le concept des rôles thématiques mais qu'ils appellent des Frames Elements qui agissent comme des étiquettes pour les divers arguments des verbes. Toutefois, dans FrameNet ces éléments sont spécifiques aux concepts du frame et non domain-independent, ainsi il en découlent de nombreux frame elements. Ça montre ici qu'on a plein de frame elements différents et ça n'apporte pas grand chose de les différencier, car en termes de fonction ils opèrent la même chose syntaxiquement. suivi d'un exemple avec tableau pour montrer la différence entre un synset X et un frame X (prendre l'exemple de l'article de FrameNet)

court article : FrameNet (Baker *et al.*, 1998) : Les unités lexicales sont décrites en termes de frame semantics. Puis les unités lexicales ainsi que leurs frame semantics sont confirmées par des phrases exemples manuellement anotées sémantiquement provenant de corpus de la langue anglaise. Le but était encoder les connaissances sémantiques des unités lexicales dans une forme que les machines peuvent lire. Ils ont couvert les domaines sémantiques suivant : santé, chance, perception, communication, transaction, temps, espace, corps, motion, étapes de la vie, contexte sociaux, émotion et cognition. Ça nous donne une base de données lexicales. Qu'on décompose en 3 modules : 1 lexicon qui contient toutes les unités lexicales, une base de données de frames et les exemples annotées correspondant aux frames. La base de données de frames contient les descriptions des frames qui sont encodées en structures conceptuelles. Et les phrases exeples manuellement annotées sont des preuves empiriques que les frames ont lieu d'être. Les frames veulent décrire la structure argumentielle des unités lexicales. Ces arguments sont décrits par des étiquettes similaires aux rôles thématiques,

mais beaucoup plus vastes car il y a différentes étiquettes en fonction du frame en question. On les appelle des frames elements et ceux-ci sont en lien avec le frame dont ils font partie. À noter qu'il y existe aussi une organisation hiérarchique où on a des sous-frames qui héritent de traits des frames parents. En frame semantics, un frame correspond à un scénario qui implique une interaction et des participants (ceux-ci jouent un rôle dans l'interaction)(Shi et Mihalcea, 2005)

utilisation en NLP : On a combiné VN et FN pour en faire un semantic parser.(Shi et Mihalcea, 2005). Semantic parser : Au lieu de faire un parse tree syntaxique, ça fait un parse tree sémantique et ça nous recrée un arbre en tenant compte des participants et de leur relation avec le verbe et entre eux.

mapping à VerbNet :(Shi et Mihalcea, 2005) Le mapping s'est fait en deux étapes. d'abord, il fallait mapper les entrées lexicales de VN avec des frames appropriés de FN. Ensuite, il fallait mapper les rôles sémantiques de VN avec les frames elements de FN.

1.1.3.3. XTAG

: Les chercheurs du projet XTAG ont construit une grammaire de la langue anglaise en se basant sur le formalisme de TAG (Tree Adjoining Grammar). Elle offre des descriptions riches des verbes en anglais. Chaque item lexical se fait assigner un ensemble d'arbres décrivant ses propriétés syntaxiques de base. Les arbres reflètent la structure prédicat-argument de ces items. Les arbres peuvent se combiner en deux opérations : substitution et adjonction. En adjoignant de nouvelles branches ou en substituant des branches, la grammaire TAG permet de rendre compte des divers phénomènes linguistiques de la langue anglaise. La grammaire de XTAG inclut des descriptions syntaxiques pour 33 000 items lexicaux dont 9000 verbes et un éventail de 1300 arbres décrits et organisés en 70 famille d'arbres. (parler du mapping)

a lexicalized tree adjoining grammar for English (Research Group, 2001) : Les arbres dans des familles d'arbres sont liés par le fait qu'ils représentent les mêmes type de cadres de sous-catégorisation. Ce qui distingue 2 arbres dans une même famille est des alternances syntaxiques. Les classes verbales sont organisées ainsi : Chaque verbe dans le lexicon syntaxique est associé à plusieurs familles d'arbres, où chaque famille regorge d'arbres individuels issues de différentes transformation syntaxiques mais ayant un cadre de sous-catégorisation les unissant. Ainsi, de cette manière un même verbe peut pointer vers différents arbres individuels, en passant par les familles d'arbres, et en étant associé à diverses familles d'arbres. Ainsi, ils n'ont pas à spécifier tous les arbres individuels qu'un verbe peut prendre, mais des familles d'arbres. Les familles d'arbres sont ainsi créées en prenant une forme canonique d'un cadre de sous-catégorisation et de lui faire toutes les transformations que permettent

la langue anglaise, puis de générer les arbres en découplant avec les 2 opérations et storer le tout dans la base de données d'arbres de XTAG.

XTAG system (Doran, Egedi, Hockey, Srinivas et Zaidel, 1994) : Ce que fait XTAG, une phrase en input, puis on la fait passer par deux filtres : un analyseur morphologique et un POS tagger. Maintenant qu'on a des unités lexicales étiquetées et morphologiquement annotées. Ce filtrage augmente la rapidité du parsing. Puis le parser poursuit avec les unités qui lui sont fournis pour faire sa job de parser. Le parser consulte la base de données syntaxiques et la base de données d'arbres pour trouver l'arbre approprié pour chaque item lexical. Puis le parser combine les arbres élémentaires avec les 2 opérations adjonctions/substitution ce qui donne en output le parsing de la phrase donnée en input. Le travail similaire qui a été fait est la base de données d'arbres qui sont regroupés en famille d'arbres. La base de données syntaxiques pointe vers les arbres associées à chaque entrée lexicale, ou les familles d'arbres. On distingue arbre individuel (généralement non-verbaux) vs famille d'arbres (généralement des verbes) car les familles d'arbres représentent des cadres de sous-catégorisation. Les arbres dans une famille d'arbre sont liés entre eux par des transformation (changement de diathèse)

Assigning XTAG trees to VerbNet(Ryant et Kipper, 2004) : Les auteurs de VerbNet ont ainsi voulu mapper leurs cadres syntaxiques aux arbres de XTAG. Ça a permis d'étendre leur coverage à d'autres constructions syntaxiques. Comme XTAG a tenté de répertorier toutes les transformations possibles qu'une cadre de sous-catégorisation peut subir, il y avait bcp de matière. VerbNet couvre surtout la voix déclarative, tandis que XTAG explore toutes les transformations possibles. Mais ne fait pas de distinctions pour les différents sens des verbes et ne traite pas de sémantique. (juste dire qu'il y existe un mapping) et qu'ils sont allés étendre leur coverage un peu.

1.1.3.4. *Dorr's LCS database*

: La base de données de Dorr provient des théories de Jackendoff qui argumente en faveur d'une approche de décomposition sémantique pour la sémantique des verbes en termes de leur structure conceptuelle lexicale. L'élément principal d'un LCS sont des constituents conceptuels, les primitifs conceptuels et les champs sémantiques. Les constituents conceptuels appartiennent à un ensemble de catégories : chose, évènement, état, lieu, chemin, propriété, but, manière, montant, temps. Les champs sémantiques : +temp, +loc, +poss qui agissent comme des restrictions sélectionnelles. Les primitifs conceptuels : ÊTRE, ALLER, RESTER, CAUSER, INCHOATIF, EXTENSION. Les prépositions sont aussi des primitifs conceptuels : AT, FROM, IN, ON, et on peut ajouter des restrictions sélectionnelles à des primitifs conceptuels. Une décomposition sémantique des verbes en termes de structures

lexicales conceptuelles explique leur propriété syntaxiques. Dans des systèmes qui utilisent ces concepts, la sémantique des verbes est explicitée en le décomposant en structure lexicales conceptuelles. Avec ce formalisme, on pense que les verbes avec des LCS similaires partagent aussi des comportement syntaxiques comme des alternances de diathèses. Ils utilisent aussi des rôles thématiques pour montrer la structure argumentale.

Ainsi, Bonnie Dorr a fait un système de machine translation à partir de cela.

La base de donnée de Dorr LCS, inclue une grande quantité de verbes organisés en classes sémantiques dérivés des classifications de Levin. Les auteurs de VerbNet trouvent qu'il y a un manque du côté des structures syntaxiques qui sont beaucoup trop simplifiées, et du fait que le lien entre la sémantique et la syntaxe n'est pas clair. Et une grande partie de ces données avaient été cherchées automatiquement menant à des incohérences, c'est pourquoi, après avoir fait une analyse des données ils n'ont pris que très peu d'information provenant de LCS.

Application en NLP : text generation ([Traum et Habash, 2000](#))

1.1.3.5. *Comlex*

: Base de données en anglais, développée à NYU. C'est une ressource syntaxique riche, mais qui n'est pas accessible au public. Chaque item lexical a des traits syntaxiques et des compléments. Il existe 92 sous-catégorisation disponibles pour les verbes. Contient des descriptions syntaxiques pour 6000 verbes. Selon les auteurs de VerbNet, il ne fait pas de distinction entre les différents sens d'un verbe (ce qui est problématique), et ne traite pas de sémantique explicitement. Mais il s'agit d'une base de données sur les constructions syntaxiques des unités lexicales qu'ils traitent. Application en NLP

voir l'article sur Comlex ([Grishman et al., 1994](#))

1.1.3.6. *DDO*

: dictionnaire danois utilisant LFG pour faire son lexicon Application en NLP

1.1.3.7. *LEFFF*

: un dictionnaire morpho-syntaxique pour le français Application en NLP

1.1.4. textes pertinents ?

Messiant : Korhonen : Czech :

1.1.5. Utilisation de VerbNet dans des applications NLP

automatically building conceptual graphs using VerbNet and WordNet expanding verb coverage with VerbNet Putting Pieces Together : Combining FrameNet, VerbNet and WordNet for Robust Semantic Parsing Question Answering with Lexical Chains Propagating Verb Arguments

1.1.6. Pourquoi on n'a pas utilisé les rôles thématiques et les prédicats sémantiques

Parler de ça dans la section python où on crée les patrons de régime, ça s'y prête plus. p.210 dans le livre Melcuk on pourrait garder les rôles pis les mettre dans MATE aussi

1.1.7. Évaluation du système

chapitre 5, p.75 C'est important de parler de l'évaluation du système car c'est celui que nous utiliserons, et nous voulons voir les problèmes qu'ils ont rencontré. Pour ensuite comparer avec l'utilisation que nous en aurons fait et à quel point les bons points selon eux ne sont peut-être pas les mêmes bons points que nous avons vu. Est-ce que VerbNet contient des coquilles car il a été créé par un processus automatique ? Voir la dissertation. (automatic techniques for extending coverage)

1.1.8. Pourquoi on a décidé d'utiliser VerbNet ?

1.2. PYTHON

Pour cette section : faire des commentaires directement dans le code, et après, sous le code, expliquer ce que le code fait ensemble.

Tel que mentionné dans la section précédente concernant l'architecture de VerbNet. Nous avons vu comment les documents XML sont arrangés. Ainsi, à l'aide du module *xml.etree.cElementTree* nous avons pu faire des opérations sur l'ensemble des données de VN.

Pour cette partie, découper une page de script en différents blocs de script et expliquer en dessous chaque bloc. Ce sera plus clean que de mettre pleins de commentaires dans le code et expliquer tout d'un coup.

1.2.1. Création du dictionnaire de verbe : lexicon.dict

LISTING 1.6. code pour lexicon.dict

```
def supers(t, i):
    ID = t.get('ID')
    sc = {ID:i}
    subclasses = t.findall('SUBCLASSES/VNSUBCLASS')
    if len(subclasses) > 0:
        for sub in subclasses:
            sc = {**sc, **supers(sub, ID)}
    return sc

def treeframes(t):
    ID = t.get('ID')
    z = []
    for frame in t.findall('FRAMES/FRAME'):
        description = re.sub(r"\s*[\s\.\-\ +\\\\/\\(\)]\s*", '_ ',
                             frame.find('DESCRIPTION').get('primary'))
        if description in exclude:
            continue
        description = re.sub('PP', 'PP_{}', description)
        preps = [p.get('value') or p.find('SELRESTRS/SELRESTR').get('type').upper() for p in f
        preps = [sorted(p.split()) for p in preps]
        examples = [e.text for e in frame.findall('EXAMPLES/EXAMPLE')]
        if len(preps)==1:
            description = description.format('_', '.join(preps[0]))
        elif len(preps)==2:
            description = description.format('_', '.join(preps[0]), '_'.join(preps[1]))
        elif len(preps)==3:
            description = description.format('_', '.join(preps[0]), '_'.join(preps[1]), '_'.join(
            z.append((description, examples))

    subclasses = t.findall('SUBCLASSES/VNSUBCLASS')
    subframes = [treeframes(subclass) for subclass in subclasses]
    subframes = sum(subframes, []) # flatten list of lists
    return [(ID, z)] + subframes

with open('lexicon.dict', 'w') as f:
    f.write('lexicon {\n')
    for file in [f for f in os.listdir('verbnet') if f[-4:] == '.xml']:
        root = ET.parse('verbnet/'+file).getroot()
        d = dict(treeframes(root))
        sc = supers(root, 'verb')
        for c in d.keys():
            f.write('"' + c + '"')
            if sc[c] == 'verb':
                f.write(': ' + sc[c] + ' {')
            else :
                f.write(': ' + '"' + sc[c] + '"' + ' {')
            [f.write('\n gp = { id=' + gp[0] + (max(len(gp[0]), 30)-len(gp[0]))*' ' + ' dia=x
            f.write('\n}\n')
    f.write('\n}
```

Dans les feuilles XML de VerbNet, l'information que nous cherchions pour améliorer notre système MATE était : tous les patrons de régimes possibles pour une classe de verbe. Au départ, nous ne savions pas encore ce que nous voulions extraire de ces feuilles XML, elles abondent en information. Toutefois, il semblait évident que nous pourrions vraiment en tirer parti. Nous avons donc commencé par extraire l'information se trouvant sous SYNTAX en pensant que le syntactic frame était ce que nous utiliserions pour construire notre lexicon. Toutefois, nous nous sommes vite rendu compte dans le processus que ce n'était pas exactement ce que nous voulions. Ça nous est apparu évident lorsque nous avons extrait les descriptions des syntactic frames, les exemples, puis les données sous SYNTAX. En effet, *NP V S INF ['NP', 'VERB', 'NP'] I loved to write.* un exemple comme celui-ci nous montre que les informations sous SYNTAX ne corresponde pas à ce que nous cherchions, car la description du patron de régime et le patron de régime en soi différent. Ils mettent que "to write" correspond à un NP. Ce qui n'est pas le cas quand on y pense bien. De plus, lorsqu'on extrait les balises sous SYNTAX, on a décidé de ne pas extraire les attributs contenus sous les balises car c'était de l'information soit sur les rôles thématiques ou sur les prépositions.

Nous avons donc utilisé VerbNet de mieux que nous le pouvions. Ainsi, les descriptions des FRAMES SYNTACTIC étaient "accurates" et allait nous donner les brèves descriptions des patrons de régime que nous ajouterons à notre lexicon. Ainsi, pour l'exemple mentionné NP V S INF, ce que nous en retenions, c'est que pour cette classe de verbe, il existe un patron de régime où on a un sujet, puis un verbe, et ensuite une proposition infinitive comme complément d'objet direct. Cette information était suffisante pour créer le dictionnaire de patron de régime. Car, nous pensions au départ que nous pourrions tout prendre de verbnet pour créer notre lexicon. Nous voulions chercher les descriptions des patrons de régime ainsi que l'information sous SYNTAX sous FRAME pour ainsi créer les patrons de régime en soi en les traduisant dans notre langage (TST). Toutefois, leur manière d'encore les patrons de régime ne correpond pas à la notre sur beaucoup trop d'aspect. Nous le verrons plus tard, il y a de la redondance à certains moments et notre théorie rend mieux compte des patrons de régime. Mais pour l'instant, ce qui est important de noter c'est que nous notons les patrons de régime en attribuant des actants I et des noms de relation pour signifier que est le rôle de cet actant lors du passage de la sémantique à la syntaxe. VerbNet ne fait pas cela de la même manière que nous. Ils ont aussi un volet sémantique, mais qui ne se branche pas à notre modèle théorique. Nous voulions donc nous inspirer quand même de leur méthode pour nos patrons de régime, mais le fait qu'ils utilisent les rôles thématiques nous posait problème. Il était difficile d'associer un rôle thématique à un actant syntaxique (bien que nous ayons tenté [montrer le graphique de F.L]). Donc, nous en sommes venus à la conclusion qu'il nous faudrait créer les patrons de régime en Python pour ensuite les exporter dans un

format adéquat pour MATE. Pour ce faire, nous devions utilisé les descriptions des patrons de régime qui se retrouve dans *VerbNet* et à partir de la description, créer une fonction qui nous permettrait de générer rapidement des patrons de régime adéquat pour MATE en ayant simplement à remplir les trous.

Pour extraire les descriptions des patrons de régime nous avons utilisé deux fonctions. D’abord la fonction *treeframe* qui s’occupe de récupérer la description de chaque frame syntaxique à travers tout VN. Nous passons à travers tous les frames existant dans les XML de VN. Autant dans les classes que les sous-classes. Toutefois, cette fonction ne fait pas que récupérer la description du syntactic frame de VN et nous la recrache tel quel. Nous faisons quelques opérations sur les descriptions que nous extrayons. Notamment, nous remplaçons tout espace, tiret, point, barre oblique, parenthèses qui pourraient se trouver dans les descriptions par des `_` à l’aide d’expressions régulières. Puis, nous retirons certaines descriptions de gp à cause de leur caractère problématique à encoder (pour des raisons théoriques, logiques – à expliquer ailleurs). Puis une fois qu’un clean up a été fait des descriptions et que nous avons uniquement celles que nous voulions, nous procédons à une seconde étape de raffinement des descriptions. Nous utiliserons une seconde fois une expression régulière pour trouver tous les occurrences de PP car nous ajouterons les prépositions impliquées dans les PP comme tel. Pour ce faire, nous faisons un search de tous les prépositions existant dans les patrons de régime de *verbnet* (et non dans la description, c’est là que les patrons de régime de VN nous ont été utiles) et nous mettons les prépositions que nous soutirons des gp directement dans le nom de la description des gp à la suite du mot PP à chaque fois qu’on retrouve le mot PP dans une description. On obtient ainsi les descriptions nettoyées de celle qu’on ne veut pas, avec uniquement des underscore pour séparer les syntagmes puis les prépositions (lorsqu’il y a lieu) dans les noms des descriptions de gp.

Puis, une fois que nous faisons ces opérations sur les classes de *VerbNet*, nous avons scripté une méthode pour que la fonction s’applique aussi aux sous-classes.

Par la suite, nous avons créé une fonction *super* afin que la création du lexicon s’agence bien avec la manière que MATE fonctionne. Cette fonction nous permet d’utiliser le mécanisme d’héritage qui existe dans MATE. Ainsi, on peut renvoyer une sous-classe à la classe (ou sous-classe) qui la domine. De plus, cette fonction limite le nombre de descriptions de gp. *VerbNet* a aussi ce mécanisme d’intégrer autrement. Ainsi, si une classe X a 10 descriptions et qu’une sous-classe Y en a 5, mais que les 10 descriptions de la classe s’appliquent aussi à la sous-classe, on n’aura pas 15 descriptions mais juste 5 dans le sous-classe, car le mécanisme d’héritage s’occupe de faire ça. On a aussi programmé la création de notre lexicon pour que si il n’y a pas de sous-classes, que la classe hérite de la classe verb afin d’avoir les attributs

de base de cette classe (la dpos, la spos, voir MATE). De cette manière, tous les classes héritent de la classe verb si on remonte aux classes mères, ainsi, on n'a pas besoin de préciser à chaque fois la dpos/spos.

Puis finalement, nous utilisons la fonction write qui écrit le tout dans un fichier .dict. Nous loopons à travers tous les fichiers compris dans le dossier VN. Puis nous allons chercher les keys et values du dictionnaire créé à l'aide de la fonction treeframe et du dictionnaire super.(À revoir)

Nous avons aussi pris la peine d'extraire les exemples pour chaque description car il s'agit d'une information utile pour voir dans quel contexte s'utilise ce patron de régime. Et c'est une information pertinente à avoir dans un dictionnaire, donc nous l'ajouterons à notre lexicon pour notre système de GAT. Pour ce faire, nous opérons de la même manière que pour les descriptions de gp dans le sens où nous faisons l'extractiond exemples en passant par les feuilles XML et le module d'extraction xml.etree. Pour chaque frame, nous passons la fonction à l'entièreté de VN et nous nous arrêtons à la balise EXAMPLES puis à tout texte se trouvant dans cette balise. Nous avons voulu extirper les exemples alongside des descriptions car c'est de l'information pertinente à avoir dans un dictionnaire.

1.2.2. Création du dictionnaire de patron de régimes

LISTING 1.7. code pour gpcon.dict

```
def roman(n):
    return ['I', 'II', 'III', 'IV', 'V', 'VI'][n-1]

def gp(name, real_actant):
    s = name + ' {\n'
    i=0
    for actant in real_actant:
        i = i+1
        if type(actant) == list:
            for y in actant:
                s = s + " " + roman(i) + "={" + y + "}\n"
        else:
            s = s + " " + roman(i) + "={" + actant + "}\n"
    s = s + '}\n'
    return s

#SUBJECTIVE
subj = 'rel=subjective dpos=N'

#DIRECT OBJECTIVE
dir_N = 'rel=dir_objective dpos=N'
dir_V_ING = 'rel=dir_objective dpos=V finiteness=GER'
dir_V_INF = 'rel=dir_objective dpos=V finiteness=INF'
```

```

#INDIRECT OBJECTIVE
to_N = 'rel=indir_objective dpos=N prep=to'
indir_N = 'rel = indir_objective dpos = N'

#OBLIQUE
on_V = 'rel=oblique dpos=V prep=on'
to_obl_N = 'rel=oblique dpos=N prep=to'
for_obl_N = 'rel=oblique dpos=N prep=for'
as_N = 'rel=oblique dpos=N prep=as'
against_N = 'rel=oblique dpos=N prep=against'
at_N = 'rel=oblique dpos=N prep=at'

# LOC

locab = 'rel=oblique dpos=N prep=locab'
locad = 'rel=oblique dpos=N prep=locad'
locin = 'rel=oblique dpos=N prep=locin'

descriptions = {
'NP_agent_V' : [subj],
'NP_agent_V_NP' : [subj, dir_N],
'NP_asset_V_NP_PP_from_out_of' : [subj, dir_N, [from_N, out_of_N]],
'NP_attribute_V' : [subj],
'NP_attribute_V_NP_extent' : [subj, dir_N],
'NP_attribute_V_PP_by_extent' : [subj, by_N],
'NP_cause_V_NP' : [subj, dir_N ],
'NP_instrument_V_NP' : [subj, dir_N],
'NP_location_V' : [subj],
'NP_location_V_NP_theme' : [subj, dir_N],
'NP_location_V_PP_with_agent' : [subj, with_N],
'NP_location_V_PP_with_theme' : [subj, with_N],
'NP_material_V_NP' : [subj, dir_N],
'NP_material_V_PP_into_product' : [subj, into_N],

# CR\{e}ATION DU GPCON
with open('gpcon.dict', 'w') as f:
    f.write('gpcon {\n')
    for d in descriptions.keys():
        f.write(gp(d, descriptions[d]))
    f.write('}')

```

Rajouter aussi en lstlistings les résultats qui sont des documents .dict (à la fin de l'explication)

Soit le mentionner ici, ou ailleurs, mais il a fallu faire un dictionnaire de patron de régime. D'abord, parce qu'on s'est rendu compte que du à toute l'information qu'on allait chercher et la différence dans le type d'information, on a jugé bon de créer un second dictionnaire qui ne contiendrait que les gps, autrement dit un gpcon. Celui l'information sur les patrons de régime (les actants syntaxiques). Il existe x nombre de gps répertoriés. Nous les avons trouvé en faisant un ensemble à partir de tous les descriptions que nous avons obtenus avec

le script précédent. Une fois que nous avons l'ensemble des gps différents. Il nous fallait les créer, car tel que mentionné, nous ne pouvions pas extraire les gps de VerbNet dû à une différence trop grande (cadre théorique et application). Notre système de GAT fonctionne avec la théorie Sens-Texte et nous pensons que c'est la théorie qui s'y prête le plus pour faire ce type d'opérations et qui tient le mieux compte de la manière dont le langage fonctionne. Ainsi, nous avons créé le gpcon à partir de Python car un bon nombre d'opérations peuvent être automatisés (éviter les fautes, et c'est plus transparent). Pour la création du gpcon, notre dictionnaire en Python ressemblait à ça. Nos keys() étaient la description du gp et les valeurs étaient les actants syntaxiques impliqués dans ce gp (avec de l'information sur les actants syntaxiques nécessitant une préposition à réaliser). Selon l'ordre dans lequel figure nos objets dans la liste qui est ce qu'on retrouve dans values(), notre fonction va assigner le bon actant syntaxique(I, II, III, etc.) ainsi, cette partie est automatisée grâce à cette fonction. Après, pour l'objet "subj" on va lui assigner une string 'rel=subjective dpos=N' ce qui est encodé dans une autre cellule. Ainsi à chaque fois qu'un gp a un subj, on n'a pas à écrire ce que subj contient. Alors pour l'objet subj, on aura I et 'rel=subjective dpos=N'. C'est l'union de la fonction gp et de la fonction roman qui nous permettent d'assigner les bons actants syntaxiques aux objets dans la liste qui représente les valeurs dans mon dictionnaire de gpcon.

1.2.3. Extraction des membres des classes verbales pour enrichir le dictionnaire

LISTING 1.8. code pour ajouter des lexèmes à lexicon.dict

```
def treemember(t):
    ID = t.get('ID')
    members = [m.get('name') for m in t.findall('MEMBERS/MEMBER')]
    subclasses = t.findall('SUBCLASSES/VNSUBCLASS')
    submembers = []
    if len(subclasses) > 0:
        for sub in subclasses:
            submembers = submembers + treemember(sub)
    return [(ID, members)] + submembers

files = [f for f in os.listdir('verbnet') if f[-4:] == '.xml']

members = dict(sum([treemember(ET.parse('verbnet/'+file).getroot()) for file in files], [])) #
values = sum(list(members.values()), []) # ici c'est uniquement les membres, sans infos sur le
dups = {m:[ID for ID in members.keys() if m in members[ID]] for m in values if values.count(m)
unique_member = {m:ID for ID in members.keys() for m in values if m in members[ID] and values.
lexemes = {d[0]+'_'+str(n+1):d[1][n] for d in dups.items() for n in range(len(d[1]))}

# Ici, j'ai fusionné les dictionnaires ensemble
unified_dict = {**unique_member, **lexemes}

with open('members.dict', 'w') as f:
```

```

f.write('members {\n')
for key in sorted(unified_dict.keys()):
    f.write(key)
    f.write(' : '),
    f.write(''+str(unified_dict[key])+''')
    f.write('\n')
f.write('\n}\n')

```

1.2.4. Scripts pour faire les tests

1.2.4.1. *Extraction des exemples*

LISTING 1.9. code pour créer phrases.txt

```

def treeframes(t):
    z = []
    for frame in t.findall('FRAMES/FRAME'):
        description = re.sub(r"\s*[\s\.\-\\ +\\\\/\\(\)]\s*", '_', frame.find('DESCRIPTION').get(
        if description in exclude:
            continue
        examples = [e.text for e in frame.findall('EXAMPLES/EXAMPLE')]
        z = z + examples
        subclasses = t.findall('SUBCLASSES/VNSUBCLASS')
        subframes = [treeframes(subclass) for subclass in subclasses]
        subframes = sum(subframes, []) # flatten list of lists
    return z + subframes

liste=[]
with open('phrases.txt','w') as f:
    for file in [f for f in os.listdir('verbnnet') if f[-4:] == '.xml']:
        root = ET.parse('verbnnet/'+file).getroot()
        d = (treeframes(root))
        final_liste = liste + d
        [f.write(x+'\n') for x in final_liste]

```

1.2.4.2. *Création des structures qui serviront de tests*

Dans la figure ci-bas, on explique comment on a créer les documents .str qui serviront d'input à notre système MATE qui prend ce genre de document en entrée.

LISTING 1.10. code pour créer des structures .str

```

phrases = open('phrases.txt','r')

with open('structures.str','w') as f:
    for (i,p) in enumerate(phrases):
        with open('s'+str(i)+'.str','w') as g:
            structure = 'structure Sem S'+str(i)+'{\n S {text="'+p.strip()+'"\n\n main-> \n }\'
            f.write(structure)
            g.write(structure)

```

Chapitre 2

GÉNÉRATION AUTOMATIQUE DE TEXTE

Pour cette partie, prendre l'article en 8 pages de François, et observer comment il décrit le déroulement de MATE. Reprendre l'idée, mais approfondir chaque point

Ce sera une section du mémoire sur le mécanisme `dsynt=X`

2.1. MÉCANISME `DSYNT=CREATED->CONSTRAINED->OK` POUR GÉNÉRER UN ÉNONCÉ SIMPLE (SUJET, VERBE, OBJET)

2.1.1. Application de la règle `root_standard`

Cette règle crée un noeud qui sera la racine de l'arbre syntaxique. Ce noeud se fait imposer des contraintes. Notamment, on demande à ce que ce soit un lexème appartenant à la partie du discours : verbe et que sa finitude soit de type : fini. On impose à ce noeud le trait `dsynt=constrained` pour que ça s'harmonise avec la règle `lex_standard`, mais c'est à revoir.

2.1.2. application de la règle `lex_standard` pour lexicaliser le verbe principal

On assigne un lexème à un noeud créé en syntaxe profonde. Dans le contexte actuel, on l'utilise pour sélectionner l'unité lexicale qui matche les contraintes énoncées sur le noeud vide créé par la règle `root_standard`. Ce lexème provient du lexicon. Lorsque la règle s'applique et qu'elle consomme le noeud en y mettant la bonne lexicalisation (qui respecte les contraintes sur le noeud), on ajoute un trait `dsynt=OK` pour signifier que le sémantème a été réalisé en syntaxe profonde et qu'on ne fasse plus d'opérations sur ce noeud.

2.1.3. Application de la règle `actant_gp_selection`

Cette règle s'applique lorsque nous avons un prédicat. On crée une variable[?GP] qui nous fournit un chemin vers l'information encodée sous l'attribut *gp* d'un lexème [?X]. Puis, on extirpe les traits *id* et *dia* pour chaque attribut *gp* de notre [?X]. Une fois qu'on a récupéré ces informations, on les appose au lexème en question car on se servira de ces informations pour l'application de règles subséquentes. Le trait *id* représente la description du patron de régime (chaque description se retrouve dans notre *gpcon* qui est un dictionnaire de *gp*) et le trait *dia* nous renseigne sur la diathèse de ce patron de régime, c'est-à-dire combien d'actants sont en jeu, et dans quel ordre sont-ils placés? Il est essentiel qu'un lexème verbal aille chercher ces traits car il en a besoin pour appliquer les règles actanciennes qui en découlent. Il faut que le système sache quel patron de régime utilisé pour un prédicat donné, et dans quel ordre les actants seront réalisés en syntaxe.

2.1.4. application des règles actanciennes

Une fois qu'un *gp* est sélectionné, on appliquera la règle actancielle qui lui correspond. Nos règles actanciennes ressemblent à : `actant_gp_ijk`. Les règles prennent en input les arcs sémantiques liant les actants à leur prédicat. Ces règles génèrent des noeuds vides auxquels on appose un trait `dsynt=created` pour signifier qu'on vient de créer des noeuds vides (on dit qu'ils sont vides car ils n'ont pas encore été consommés par une unité lexicale) en syntaxe profonde. On obtient ainsi des arcs syntaxiques au bout desquels se trouve un noeud vide. Ces règles se font imposer des conditions bien strictes. Il faut d'abord que le prédicat qui les gouverne soit lexicalisé. Ce qui se traduisait par l'ajout d'un trait `dsynt=OK` au lexème lexicalisé (avec la règle `lex_tandard`). La règle `actant_gp_selection` nous permettait de soustraire les traits *id* et *dia* et c'est ici que le trait *dia* entre en jeu. On s'en sert pour illustrer la diathèse du verbe. [à revoir]

2.1.5. application de la règle `constraints_gp`

Cette règle applique des contraintes à des noeuds nouvellement créés (autrement dit, des noeuds qui ont le trait `dsynt=created`). C'est à partir de cette règle que le mécanisme de *dsynt=created*-> *constrained*-> *OK* prend vie. Il s'agit d'une étape intermédiaire entre la création du noeud avec les règles actanciennes et la lexicalisation qui consommera le noeud en octroyant un lexème suivant un trait `dsynt=OK` (signifiant que la réalisation en syntaxe profonde est terminée). Ainsi, la règle procède de cette manière. On prend un X (un prédicat) qui lie un Y (un actant) puis ce dernier se fait imposer des contraintes. On cherche à lui

imposer ces contraintes car on souhaite qu'il respecte le patron de régime de X. Ainsi, s'il ne convient pas comme actant syntaxique, il ne pourra pas passer à la lexicalisation. Car on souhaite une correspondance entre les traits naturels contenus dans le dictionnaire pour le sémantème en question. Si ses traits ne conviennent pas aux contraintes imposées au noeud, provenant du patron de régime de X, alors l'arbre sera incomplet. Lorsqu'on met les contraintes sur le noeud, on lui met aussi le trait `dsynt=constrained` pour montrer qu'il a été contraint et donc qu'il remplit les critères pour passer à la lexicalisation.

2.1.6. application des règles de lexicalisation

Il y a différentes règles de lexicalisation afin de donner un peu de latitude au système. Ainsi, si quelques informations sont manquantes dans le lexicon ou le semanticon, nous voulons que le système arrive quand même à effectuer une génération de texte. C'est pourquoi il existe la règle standard qui fonctionne lorsque nous avons accès à tous les éléments nécessaires. Les règles de types *guess* opèrent lorsqu'il nous manque certaines infos.

2.1.6.1. *lex_standard*

Nous avons vu cette règle un peu plus haut, car il fallait l'appliquer dès le début pour lexicaliser le noyau principal afin de chercher le bon lexème qui pouvait remplir cette fonction. Une fois que nous l'avons lexicalisé, nous avons pu aller chercher les informations sur la nature de son gp etc. Nous sommes maintenant rendus au moment où il existe des arcs syntaxiques au bout desquels nous avons des noeuds contraints par des traits comme : la dpos, la finitude, la définitude, etc. Nous allons donc lexicaliser ces noeuds afin de poursuivre la construction de l'arbre, du haut vers le bas. En gros, comment ça fonctionne. On cherche la lexicalisation d'un sémantème donné dans le semanticon, en allant chercher le trait *lex* du sémantème. Puis, on colle cette lexicalisation à un noeud déjà existant. Ce noeud existe déjà car il a soit été créé par *root_standard* ou par une des règles actanciennes. On va octroyer au noeud 3 traits : un trait *dlex*, un trait *dpos*, et finalement un trait *dsynt*. Le trait *dlex* est la lexicalisation profonde, celle qu'on retrouve dans le lexicon, le trait *dpos* est la partie du discours profonde qui doit correspondre à la dpos demandée par *constraints_gp*. Et finalement, un trait *dsynt=OK* qui s'ajoute à la lexicalisation du sémantème pour signifier au système que le noeud a été consommé et qu'il a été réalisé en syntaxe profonde. Donc, qu'on ne fasse plus d'opération sur ce noeud. Finalement, on peut uniquement faire des opérations sur des noeuds qui ont le trait *dsynt=constrained* afin de lexicaliser seulement les noeuds qui se sont fait attribués des contraintes. Ainsi, s'ils respectent les contraintes, ils pourront être lexicalisés, sinon, l'arbre sera incomplet et la génération échouera.

2.1.7. Avantages d'utiliser ces mécanismes

"Dsynt=OK" indique que le noeud a été consommé. Autrement dit, il existe maintenant une unité lexicale réalisé en syntaxe profonde, là où il y avait un noeud vide. "Dsynt=constrained" indique qu'un noeud vide s'est fait imposé des contraintes. Ces contraintes peuvent être de plusieurs ordres. Notamment, on impose une dpos, une finitude, un mood, etc. Il s'agit du passage intermédiaire entre la création d'un noeud et sa lexicalisation. On veut s'assurer qu'il respecte certaines contraintes pour ne pas lexicaliser n'importe quoi lors de la génération de notre arbre syntaxique. Finalement, le trait dsynt=created

2.2. ÉNUMÉRATIONS

Voici une énumération avec numérotation :

1. item 1 ;
2. item 2 ;
3. item 3.

Maintenant, une énumération sans numérotation avec des marqueurs différents :

- Marqueur par défaut ;
- \bullet ;
- ★ \star .

2.3. ÉQUATIONS MATHÉMATIQUES

Une équation :

$$\otimes^n \mathbb{C}^2 \cong \bigoplus_{m=-n/2}^{n/2} W_m.$$

Une autre équation, cette fois-ci numérotée :

$$\frac{\partial \mathcal{L}}{\partial \phi^a} - \partial_\mu \frac{\partial \mathcal{L}}{\partial (\partial_\mu \phi^a)} = 0, \quad \mu = 0, 1, 2, 3. \quad (2.3.1)$$

Les équations (2.3.1) précédentes sont appelées *équations d'Euler-Lagrange* ou encore *équations du mouvement*. Dans les calculs suivants,

$$\begin{aligned} \delta S &= \int_\Omega d^d x \mathcal{L}(\phi'^a(x), \partial_\mu \phi'^a(x)) - \int_\Omega d^d x \mathcal{L}(\phi^a(x), \partial_\mu \phi^a(x)) \\ &= \int_\Omega d^d x \left[\delta \phi^a \frac{\partial \mathcal{L}}{\partial \phi^a} + \partial_\mu \delta \phi^a \frac{\partial \mathcal{L}}{\partial (\partial_\mu \phi^a)} \right] \end{aligned}$$

$$\begin{aligned}
&= \int_{\Omega} d^d x \left[(\delta \phi^a \frac{\partial \mathcal{L}}{\partial \phi^a} + \partial_{\mu} \left(\delta \phi^a \frac{\partial \mathcal{L}}{\partial (\partial_{\mu} \phi^a)} \right) - \delta \phi^a \partial_{\mu} \frac{\partial \mathcal{L}}{\partial (\partial_{\mu} \phi^a)} \right] \\
&= 0,
\end{aligned}$$

aucune ligne n'est numérotée. Alors que dans ce qui suit, la dernière ligne l'est :

$$\begin{aligned}
\delta S &= \int_{\Omega'} d^d x' \mathcal{L}(\phi'^a(x'), \partial'_{\mu} \phi'^a(x')) - \int_{\Omega} d^d x \mathcal{L}(\phi^a(x), \partial_{\mu} \phi^a(x)) \\
&= \int_{\Omega} d^d x \left[\bar{\delta} \phi^a \frac{\partial \mathcal{L}}{\partial \phi^a} + \partial_{\mu} \bar{\delta} \phi^a \frac{\partial \mathcal{L}}{\partial (\partial_{\mu} \phi^a)} \right] + \int_{\partial \Omega} d\sigma_{\mu} \mathcal{L}(\phi^a, \partial_{\mu} \phi^a) \delta x^{\mu} \\
&= \int_{\Omega} d^d x \partial_{\mu} \mathcal{J}^{\mu}(x).
\end{aligned} \tag{2.3.2}$$

2.4. DÉFINITIONS, THÉORÈMES ET PREUVES

Voici une définition.

Définition 2.4.1 (La définition). *La définition.*

Voici un théorème.

Théorème 2.4.1 (Titre). *Ceci est vrai !*

DÉMONSTRATION. Voici la preuve. □

Démonstration. Voici la preuve en gras. □

2.5. CONSTRUCTION D'UN TABLEAU

TABLEAU 2. I. Un tableau simple dans le second chapitre.

Option	g	c	d	p{0.4\textwidth}
Effet	À gauche	Au centre	À droite	Le texte de cette colonne est justifié et la largeur de la colonne est fixée à 40 % de la zone de texte (hors tableau).

Le tableau 2. I n'est pas très garni.

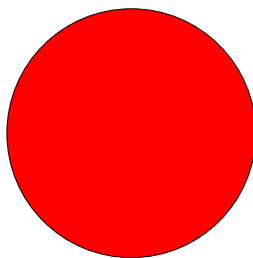
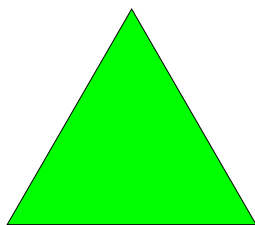
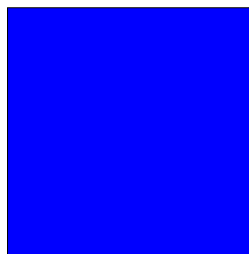


FIGURE 2.1. Un cercle.



(a) Un triangle.



(b) Un carré.

FIGURE 2.2. Un carré et un triangle.

2.6. RÉFÉRENCE À UNE ENTRÉE BIBLIOGRAPHIQUE

Les documents par ?? ainsi que ? sont des références en matière de L^AT_EX. Le manuel par ? est probablement le plus populaire du lot.

L'article de ? est, manifestement, très riche en rebondissements.

Les entrées du fichier `.bib` qui ne sont pas référencées dans le texte ne sont pas ajoutées à la bibliographie : un avantage de plus en faveur de BibT_EX.

Dans ce paragraphe, on teste une cette référence ?.

2.7. INSERTION DE FIGURES

La figure 2.1 est un *cercle*. À la figure 2.2, le triangle (a) et le carré (b) ont été placés côtes-à-côtes grâce à la commande `\subfigure`.

Bibliographie

Ayan, N. et B. J. Dorr. 2002, «Generating A Parsing Lexicon from an LCS-Based Lexicon», *LREC 2002 Workshop Proceedings : Linguistic Knowledge Acquisition and Representation : Bootstrapping Annotated Language Data*.

Baker, C. F., C. J. Fillmore et J. B. Lowe. 1998, «The Berkeley FrameNet Project», dans *Proceedings of the 17th International Conference on Computational Linguistics - Volume 1*, COLING '98, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 86–90, doi : 10.3115/980451.980860.

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

Doran, C., D. Egedi, B. A. Hockey, B. Srinivas et M. Zaidel. 1994, «XTAG System : A Wide Coverage Grammar for English», dans *Proceedings of the 15th Conference on Computational Linguistics - Volume 2*, COLING '94, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 922–928, doi :10.3115/991250.991297.

Fellbaum, C., éd.. 1998, «WordNet : An Electronic Lexical Database», Language, Speech, and Communication, MIT Press, Cambridge, MA, ISBN 978-0-262-06197-1.

Grishman, R., C. Macleod et A. Meyers. 1994, «Complex Syntax : Building a Computational Lexicon», dans *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*, COLING '94, Association for Computational Linguistics, Kyoto, Japan, p. 268–272, doi :10.3115/991886.991931.

Kipper, K., H. T. Dang et M. Palmer. 2000, «Class-Based Construction of a Verb Lexicon», dans *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, AAAI Press, ISBN 978-0-262-51112-4, p. 691–696.

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

- Lareau, F., F. Lambrey, I. Dubinskaite, D. Galarreta-Piquette et M. Nejat. 2018, «GenDR : A Generic Deep Realizer with Complex Lexicalization», dans *Proceedings of 11th Edition of the Language Resources and Evaluation Conference (LREC)*, Miyazaki.
- Levin, B. 1993, «English verb classes and alternations : A preliminary investigation», .
- Miller, G. A. 1995, «WordNet : A Lexical Database for English», *Commun. ACM*, vol. 38, n° 11, doi :10.1145/219717.219748, p. 39–41, ISSN 0001-0782.
- Pustejovsky, J. 1991, «The Generative Lexicon», *Comput. Linguist.*, vol. 17, n° 4, p. 409–441, ISSN 0891-2017.
- Research Group, T. X. 2001, «A Lexicalized Tree Adjoining Grammar for English», *IRCS Technical Reports Series*.
- Ryant, N. et K. Kipper. 2004, «Assigning XTAG Trees to VerbNet», dans *Proceedings of the 7th International Workshop on Tree Adjoining Grammar and Related Formalisms*, Vancouver, Canada, p. 194–198.
- Schuler, K. K. 2005, «Verbnet : A Broad-coverage, Comprehensive Verb Lexicon», PhD Thesis, University of Pennsylvania, Philadelphia, PA, USA.
- Shi, L. et R. Mihalcea. 2005, «Putting Pieces Together : Combining FrameNet, VerbNet and WordNet for Robust Semantic Parsing», dans *Proceedings of the 6th International Conference on Computational Linguistics and Intelligent Text Processing, CICLing'05*, Springer-Verlag, Mexico City, Mexico, ISBN 3-540-24523-5 978-3-540-24523-0, p. 100–111, doi :10.1007/978-3-540-30586-6_9.
- Traum, D. et N. Habash. 2000, «Generation from Lexical Conceptual Structures», dans *Proceedings of the 2000 NAACL-ANLP Workshop on Applied Interlinguas : Practical Applications of Interlingual Approaches to NLP - Volume 2*, NAACL-ANLP-Interlinguas '00, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 52–59, doi :10.3115/1117554.1117561.

Annexe A

LE TITRE

A.1. SECTION UN DE L'ANNEXE A

La première annexe du document.

Pour plus de renseignements, consultez le site [web de la FESP](#). Pour plus de renseigne-

TABLEAU A. I. Liste des parties

Les couvertures conformes	obligatoires
Les pages de garde	obligatoires
La page de titre	obligatoire
Le résumé en français et les mots clés français	obligatoires
Le résumé en anglais et les mots clés anglais	obligatoires
Le résumé de vulgarisation	facultatif
La table des matières, la liste des tableaux, la liste des figures	obligatoires
La liste des sigles, la liste des abréviations	obligatoires
La dédicace	facultative
Les remerciements	facultatifs
L'avant-propos	facultatif
Le corps de l'ouvrage	obligatoire
L'index analytique	facultatif
Les sources documentaires	obligatoires
Les appendices (annexes)	facultatifs
Le curriculum vitæ	facultatif
Les documents spéciaux	facultatifs

ments, consultez le site [web de la FESP](#).

Annexe B

LE TITRE2

texte annexe B

