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

18 janvier 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

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET DES ABRÉVIATIONS

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

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 ? (intro de la dissertation sur VerbNet)

Chapitre 1

EXTRACTION DES ENTRÉES DU LEXICON *VERBNET*

Dans ce chapitre, nous verrons l'apport que la ressource lexicale *VerbNet* peut offrir à des applications de TAL. Nous avons comme objectif d'extraire l'architecture concernant l'organisation de la classe verbale qu'on retrouve dans le dictionnaire *VerbNet* pour les implémenter dans le dictionnaire de notre système de génération de texte automatique (GAT). Cet objectif provient d'une problématique que nous avions rencontré auparavant. Nous voulions savoir comment organiser la classe verbale au sein de notre système, car les verbes sont si riches et complexes qu'il nous fallait trouver un moyen systématique d'encoder cette partie du discours, et l'encoder pour que notre dictionnaire puisse intéragir correctement avec notre grammaire. Justement, 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 entrées lexicales faisant partie de cette PDD démontre 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 beauoup moins de variétés d'usage quant au nombre de patrons de régime. Cette problématique nous a donc amené à voir comment une ressource comme VN, qui s'est donnée comme mission de combler cette lacune, a fait pour résoudre le problème.

1.1. VerbNet

(Karin Kipper, Hoa Trang Dang, and Martha Palmer,2000) et Dissertation de Kipper Ainsi, tel que mentionné précedemment, VerbNet a été créé dans un contexte où il y avait un réel besoin de réfléchir à la meilleure manière de procédér pour faire un dictionnaire qui saura tenir compte de la complexité que renferment les verbes. 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 dictionnaire computationels existants. Parmi ceux qui ont tenté la chose, nous nommerons : le " generative lexicon" de Pustojevsky WordNet, ComLex, LCS et FrameNet, 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 : le generative lexicon se centre sur les noms, WN ne donne pas de détails concernant les cadres syntaxiques possibles pour un verbe ainsi qu'un vide complet sur la structure prédicat-argument, ComLex offre des cadres syntaxiques mais ne distinguent pas les différents sens qu'un verbe peut prendre, le lexicon LCS essaye de palier à ces lacunes aussi, mais ne couvre pas assez large.

1.1.1. Levin et l'organisation en classes verbales

Les classes verbale sont importantes car elles permettent de "capturer des généralisations " qu'on pourrait faire en regardant le comportement des verbes. Ainsi, au lieu de construire un dictionnaire entrée par entrée, on regroupe les entrée en classe en fonction de leur comportement syntaxique/sémantique. Ces généralisations sont basées sur ...

Les créateurs de VerbNet ont jugé qu'il serait nécessaire d'organiser leur information en classes verbales. Ce qui est un premier pas vers une compréhension de leur hypothèse sur la manière d'organiser un lexicon autour de verbes. Ce que nous avons fait par le fait même puisque c'est cette idée que nous avons recyclé de VerbNet. Ça s'est avéré effectivement tel que VerbNet l'a explicité. L'idée d'organiser l'information ainsi leur est venu de Levin qui avait déjà commencé classer les verbes et avait fait un travail considérable sur ce sujet. Ce qu'ils aimaient du classeur Levin, était (Class-Based Construction of a Verb Lexicon)

(Karin Kipper Schuler 2005,PhD) et Guidelines VerbNet est un lexicon de verbe hiérarchisé. Il renferme des informations sémantiques et syntaxiques quant aux verbes de l'anglais. Les entrées lexicales qui le populent ont été systématiquement construites à partir des classes verbales de Levin (Levin,1993). VerbNet est organisé autour des classes verbales. Autrement dit, chaque verbe introduit dans VerbNet sera associé à une classe verbale. L'avantage d'organiser son information autour des classes verbales de Levin est que ça nous permettent de bien rendre compte du fait que certains verbes se comportent de la même manière (mêmes rôles thématiques, mêmes cadres syntaxiques, même sémantique). Pour revenir au fait que VerbNet est hiérarchique, chaque premier niveau d'une entrée est une classe verbale de Levin, puis de niveau en niveau, on va du large au précis à l'intérieur même de la classe. Ce qui nous donne d'autres noeuds hiérarchiques qui héritent tous des niveau du dessus. Chaque noeud hiérarchique est caractérisé par un ensemble de verbes qui sont unis par le fait qu'ils agissent de la même manière du point de vue de la sémantique et des cadres syntaxiques

et ils partagent les mêmes types d'arguments. Ainsi, chaque classe verbale est organisé hiérarchiquement pour que une classe donnée partage les informations sur les arguments, la syntaxe et la sémantique et les membres (l'ensemble des verbes qui partagent ces trucs).

1.1.2. composantes de VerbNet

VerbNet est composé de classes verbales. Chaque classe contient un ensemble de membres qui lui sont attribués en fonction du caractère commun que les membres possèdent avec la classe, des rôles thématiques pour la structure argument-prédicat et des restrictions sélectionnelles sur ces arguments ainsi que les cadres syntaxiques qui contiennent une courte description du cadre, un exemple le démontrant et une description syntaxique du cadre. Puis finalement, un ensemble des prédicats sémantiques. Une classe peut être divisé en sous-classe si cela est nécessaire. Tel sera le cas lorsque un sous-ensemble des membres de la classe partage des cadres syntaxiques et des prédicats sémantiques spécifiques.

1.1.2.1. *Rôles thématiques*

VerbNet utilise un ensemble de 23 rôles thématiques pour identifier les arguments dans les classes verbales.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. 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 fait 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.

1.1.2.2. *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.

1.1.2.3. *Cadres*

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). Agent V Patient

1.1.2.4. *Prédicats sémantiques*

Est-ce que ça vaut la peine que j'en parle ? Leur manière de voir la sémantique est wierd as fuck

1.1.2.5. *Exemples*

Chaque cadre syntaxique est accompagné d'un exemple pour exemplifier ce que le cadre représente. On peut ainsi mieux comprendre comment la classe verbale fonctionne. Il me faudrait faire des screenshots. Et les rajouter à cette section pour montrer comment les frames fonctionnent.

1.1.3. Origine des classes verbales

1.1.3.1. *Travaux de Levin*

1.1.3.2. *Raffinement de Korhonen et Briscoe*

1.1.4. Mapping de VerbNet à d'autres ressources NLP

FrameNet, WordNet, Xtag

1.1.5. Utilisation de VerbNet dans des applications NLP

** Lire l'article "Extending Verbnet with Novel verb classes" et http ://verbs.colorado.edu/ kipper/Papers/lrec.pdf

Encore du texte... et un tableau

Le tableau 1. I n'est pas très garni.

exemple premier element

second exemple

Voici comment on fait un exemple :

(1) Un exemple.

Tableau 1. I. Un tableau simple dans le premier 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).

faut une ligne vide après

(2) Mon chien dort.

Comme on peut voir en (2), mon chien est endormi.

Paraphrases :

(3) a. Tout est beau.
b. Tout est ben beau.
c. Tout est ben ben beau.

tout est beau, comme en (3), c'est-à-dire (3-a)–(3-c).

1.2. Python

1.2.1. Extraction de données des documents XML provenant de *VerbNet*

Tel que mentionné dans le 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.

1.2.1.1. *Extraction des descriptions des patrons de régime*

(arranger mon info en sous-sous-sous-section)

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 diffèrent. 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, paranthèses 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 caractèrere 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 faisions 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 den 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, nos 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.

[Avant de passer à la prochaine section, il faut parler de la manière que mon dictionnaire fonctionne avec les keys() et values() avant que je le passe à la fonction write pour en extraire des parties]

1.2.2. Création du dictionnaire de patron de régimes (gpcon)

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 créer 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 values dans mon dictionnaire de gpcon.

1.2.2.1. *Script pour faire les actants syntaxiques*

1.2.2.2. *dictionnaire dans Python pour faire correspondre*

1.2.2.3.

1.2.3. Extraction des membres de chaque classe verbale

1.2.3.1.

Chapitre 2

GÉNÉRATION AUTOMATIQUE DE TEXTE

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 le 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 actancielles 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 actancielles

Une fois qu'un gp est sélectionné, on appliquera la règle actancielle qui lui correspond. Nos règles actancielles 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 soutirer 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 actancielles 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 convienne 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 régles 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 a un noeud déjà existant. Ce noeud existe déjà car il a soit été créé par root_standard ou par une des règles actancielles. 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 let 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 lexicalisé, 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 passsage 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 :

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

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

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

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

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

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.

2.6. Référence à une entrée bibliographique

Les documents par Lamport [3], Goossens, Mittelbach et Samarin [1] ainsi que Spivak [5] sont des références en matière de LaTeX. Le manuel par Goossens *et al.* [1] est probablement le plus populaire du lot.

L'article de Martin [4] 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 BibTeX.

Dans ce paragraphe, on teste une cette référence Hastie [2].

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`.

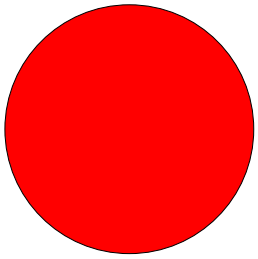
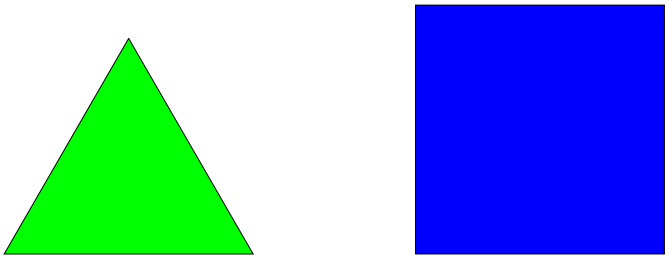


FIGURE 2.1. Un cercle.



(a) Un triangle. (b) Un carré.

FIGURE 2.2. Un carré et un triangle.

Bibliographie

[1] Goossens, M., F. Mittelbach et A. Samarin. 1994, «The LaTeX companion», New-York.

[2] Hastie, D. 2005, «Towards automatic reversible jump markov chain monte carlo», thèse de doctorat, University of Bristol.

[3] Lamport, L. 1986, «LaTeX – a document preparation system», Reading.

[4] Martin, P. 1992, «On Schur-Weyl duality, A_n Hecke algebras and quantum $\mathfrak{sl}(N)$ on $\otimes^{n+1}\mathbb{C}^N$», *Int. J. Mod. Phys. A*, vol. 7, p. 645–673.

[5] Spivak, M. D. 1990, «The joy of TeX», Providence, 2e éd..

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 renseignements, consultez le site web de la FESP.

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

Annexe B

LE TITRE2

Annexe C

LE TITRE3

Annexe D

LE TITRE4