CHAPITRE 5

ANALYSE SYNTAXIQUE

LES LANGUES

Analyser *un texte syntaxiquement veut dire mettre en évidence sa structure. Prenons la phase "Much to learn, you still have." comme exemple d'analyse. Il est clair que le vocabulaire est d'origine anglaise, mais la phrase semble un peu étrange. En anglais, les phrases sont souvent structurées comme sujet-verbe-objet. Mais dans cette phrase, l'ordre est différent : objet-sujet-verbe ; une structure similaire aux langues amérindiennes d'Amazonie. Bien sûr, dans ce cas c'est la syntaxe utilisée par Yoda (Star Wars). Il y a plusieurs façons de voir la structure syntaxique : une composition des structures jusqu'à arriver à une structure élémentaire, un ensemble de relations entre les mots, etc. Dans ce chapitre, nous allons présenter deux structures syntaxiques ainsi que quelques méthodes d'analyse syntaxique pour les deux.*

La syntaxe étudie la manière dont les mots sont combinés afin de former une phrase. Elle cherche à définir une structure standard des phrases d'un langage (naturel ou artificiel). L'analyse syntaxique aide à décider si une phrase est grammaticalement correcte ou non. Si oui, on essaye de trouver sa structure afin d'aider d'autres tâches comme :

- Détection des erreurs grammaticales
- Compréhension de la langue
- Extraction d'information

1 Structures syntaxiques

Une langue doit avoir une structure syntaxique ; des règles qui aident la formation des phrases. On parle ici d'une phrase grammaticalement correcte ; ça ne veut pas dire que la phrase doit avoir un sens. Donc, le fait qu'une phrase est bien formée syntaxiquement n'est pas une garantie qu'elle est sémantiquement correcte. Par exemple, la phrase "Les idées vertes et non colorées dorment furieusement." est bien formée syntaxiquement, mais n'a aucun sens. Il existe plusieurs théories pour représenter la structure syntaxique :

- Grammaire générative et transformationnelle
- Grammaire de dépendance
- Grammaire catégorique
- Grammaires stochastiques
- Approches fonctionnelles de la grammaire

1.1 Annotation constituante

Nous avons vu dans le chapitre précédent que chaque mot d'une langue a une catégorie grammaticale (Ex. "Le/DET cours/NOM est/VP intéressant/ADJ"). Si on revient au chapitre des modèles de langage, on peut déduire que le déterminant possède une grande probabilité d'être suivi par un nom ou un adjectif suivi par un nom (En RegEx : /DET ADJ* NOM/). On appelle cette structure un **syntagme** nominal. Il est appelé "nominal" et pas "adjectival" ou "déterminant" puisque le nom ici est le centre de la structure. L'adjectif modifie souvent un nom et un déterminant est toujours dépendant d'un nom. Il existe plusieurs **syntagmes** selon la catégorie noyau : nominal, verbal, adjectival, prépositionnel, etc. Par exemple, [Le/-DET cours/NOM]$_{NP}$ [est/V intéressant/ADJ VP]$_{VP}$. La composition de plusieurs **syntagmes** forment un autre syntagme jusqu'à arriver à la phrase. Techniquement, une phrase peut être structurée comme un arbre où la racine est la phrase, les syntagmes sont représentés par des nœuds internes et les mots avec leurs catégories grammaticales sont représentés par des feuilles.

Rappelons la classification de Chomsky ; un langage peut être : régulier, hors-contexte, contextuel ou récursivement énumérable. Plusieurs langues peuvent être formalisées en utilisant une grammaires à contexte libre ; en anglais Context Free Grammar. La grammaire $G < \Sigma, N, P, S >$ d'un langage est composée de :

- Σ : ensemble des symboles terminaux ; dans ce cas, le vocabulaire. Exemple, Σ = {le, petit, chat, mange, un, poisson, ...}.
- N : ensemble des symboles non terminaux ; les variables (syntagmes et catégories grammaticales plus d'autres variables supplémentaires). Par exemple, N = {S, NP, VP, DET, N, ADJ, ...}.
- $S \in N$: axiome ; c'est le point de départ pour former la phrase.
- P : ensemble des règles de production. Les règles sont de la forme $A \rightarrow \beta$ avec $A \in N$, $\beta \in (\Sigma \cup N)^*$. Par exemple, P={S → NP VP, NP → DET ADJ N | DET N, VP → V NP}

Parmi les problèmes trouvés lors de l'analyse syntaxique, l'ambigüité syntaxique. Prenons, à titre d'exemple, la grammaire suivante (une grammaire pas bien définie) :

- S → NP VP (1)
- NP → DT NN (2) | DT NN PP (3)
- VP → VB NP (4) | VB NP PP (5)| VB PP (6) | AU VP (7)
- PP → PR NP (8)
- DT → l' | une | son
- NN → élève | solution | stylo | explication
- VB → écrit
- AU → a
- PR → avec

Selon cette grammaire, la phrase "L'élève a écrit une solution avec son explication" aura deux interprétations. Les deux arbres syntaxiques sont illustrées dans la figure 5.1. La première considère la préposition "avec" dépendante du nom "solution", et donc on aura la dérivation suivante (Figure 5.1(1)) :

$$
\begin{aligned}
S &\overset{(1)}{\vdash} NP\ VP \overset{(2)}{\vdash} DT\ NN\ VP \overset{(7)}{\vdash} DT\ NN\ AU\ VP \\
&\overset{(4)}{\vdash} DT\ NN\ AU\ VB\ NP \overset{(3)}{\vdash} DT\ NN\ AU\ VB\ DT\ NN\ PP \\
&\overset{(8)}{\vdash} DT\ NN\ AU\ VB\ DT\ NN\ PR\ NP \overset{(2)}{\vdash} DT\ NN\ AU\ VB\ DT\ NN\ PR\ DT\ NN
\end{aligned}
$$

La deuxième considère la préposition "avec" dépendante du verbe "écrit", et donc on aura la dérivation

suivante (Figure 5.1(2)) :

$$S \overset{(1)}{\vdash} NP\ VP \overset{(2)}{\vdash} DT\ NN\ VP \overset{(7)}{\vdash} DT\ NN\ AU\ VP$$
$$\overset{(5)}{\vdash} DT\ NN\ AU\ VB\ NP\ PP \overset{(2)}{\vdash} DT\ NN\ AU\ VB\ DT\ NN\ PP$$
$$\overset{(8)}{\vdash} DT\ NN\ AU\ VB\ DT\ NN\ PR\ NP \overset{(2)}{\vdash} DT\ NN\ AU\ VB\ DT\ NN\ PR\ DT\ NN$$

Clairement il y a une ambigüité ici ; donc, essayons de reformuler les deux dérivations. La première veut dire : la solution et son explication ont été écrites par l'élève. La deuxième veut dire : la solution a été écrite par l'élève en utilisant l'explication de ce même élève. Clairement, l'explication n'est pas un outil d'écriture ; donc la première est la plus juste. Afin de régler l'ambigüité, on a besoin du niveau sémantique qui peut guider l'analyse. Dans notre exemple, on peut ajouter l'information que les outils d'écriture doivent être des entités concrètes et l'objet de l'écriture doit être un entité abstraite.

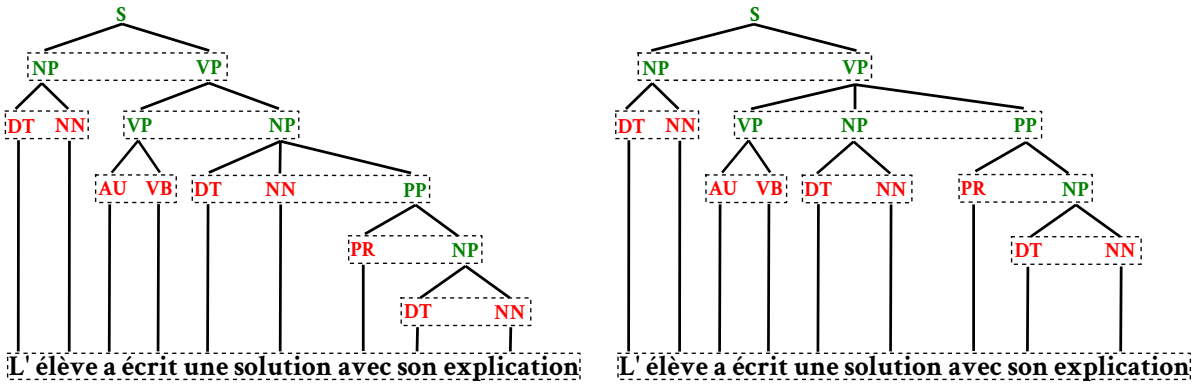


Figure 5.1 : *Exemple de deux arbres syntaxiques de la phrase "L'élève a écrit une solution avec son explication"*

Une autre méthode pour faire face à ce problème est d'utiliser les CFG probabilistes : Probabilistic Context Free Grammar (PCFG). La grammaire est similaire à celle d'un CFG : $G < \Sigma, N, P, S >$. La seule différence est que les règles sont de la forme : $A \rightarrow \beta\,[p]$ avec $A \in N$, $\beta \in (\Sigma \cup N)^*$. Seulement, nous avons ajouté la probabilité p d'occurrence de la règle. Afin d'estimer cette probabilité, on utilise un corpus annoté (**TreeBank**). Étant donné la fonction C qui calcule le nombre d'occurrence, la probabilité d'une règle $A \rightarrow \beta$ est estimée en utilisant l'équation 5.1. Dans le cas d'ambigüité entre plusieurs règles, on choisit celle avec le maximum de probabilité.

$$P(A \rightarrow \beta | A) = \frac{C(A \rightarrow \beta)}{C(A)} \tag{5.1}$$

1.2 Annotation fonctionnelle

Un mot ou plus remplissent une fonction syntaxique (Ex. sujet, objet, etc.). Par exemple, dans la phrase "Le chat mange un poisson", le mot "chat" est le sujet du verbe "mange". Dans l'annotation fonctionnelle, la fonction syntaxique est une relation binaire appelée **dépendance**. Une relation de dépendance relie un mot appelé **tête syntaxique** avec un autre appelé **dépendant**. Un mot ne peut être un dépendant qu'une seule fois, mais il peut être une tête syntaxique plusieurs fois. Un exemple de l'annotation fonctionnelle des deux phrases "L'élève a écrit une solution et son explication" et "L'élève a écrit une solution avec son stylo" est illustré dans la figure 5.2.

La structure de dépendance est formalisée comme un graphe orienté $G = (V, A)$ où les mots sont représentés par des sommets V et les relations sont représentées par des arcs A. L'arbre de dépendances d'une phrase satisfait les conditions suivantes :

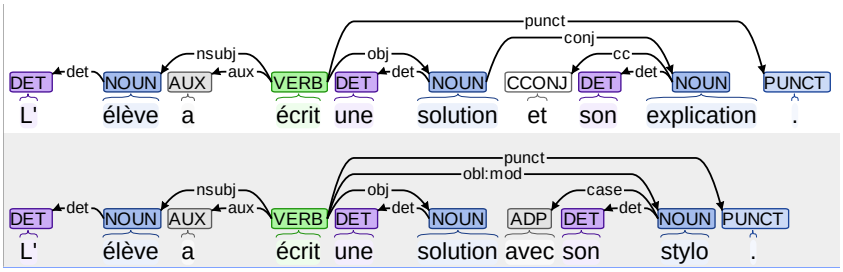


Figure 5.2 : *Exemple d'une analyse fonctionnelle de deux phrases en utilisant* `https://corenlp.run/` *[visité le 2021-09-25]*

— Il existe un seul nœud racine désigné qui n'a pas d'arcs entrants. En général c'est un verbe.
— À l'exception du nœud racine, chaque sommet a exactement un arc entrant.
— Il existe un chemin unique du nœud racine à chaque sommet de V.

Les relations de dépendance sont des relations entre deux entités. Un exemple des relations de dépendance est donnée dans le tableau 5.1.

Dép. de base	Description	Exemple
nsubj	sujet nominal	Le people **gagne**
obj	objet direct	On **présente** le cours
iobj	objet indirect	Il m'**envoie**
csubj	sujet propositionnel	Suivre le cours **permet** ...

Dép. des noms	Description	Exemple
amod	modificateur adjectival	La **fille** modeste
det	déterminant	La **fille**
nmod	modificateur nominal	Le résultat de la **course**
nummod	modificateur numérique	J'ai mangé 3 **bonbons**

Tableau 5.1 : *Quelques relations de dépendances universelles de Stanford (de Marneffe et al., 2014),* `https://universaldependencies.org/u/dep/index.html` *[visité le 2021-09-11]*

2 Analyse des constituants

La grammaire à contexte libre est le système formel le plus utilisé pour modéliser la structure constituante. Il existe deux types de méthodes pour analyser un texte :

— **ascendante** : à partir des mots de la phrase, on essaye de trouver les catégories grammaticales. Ensuite, les syntagmes qui génèrent une combinaison des catégories et des **syntagmes**. On fait ça jusqu'à arriver à l'axiome "S". Exemple, LR.
— **descendante** : à partir de l'axiome "S", on cherche les règles qui génèrent la phrase. Dans cette approche, on se base sur les mots pour guider la génération. Exemple, Descente récursive, LL, Early.

Les algorithmes mentionnées dans les exemples sont conçus pour traiter les langages de programmation. Ces derniers ont des syntaxes bien définies qui peuvent être traitées d'une manière déterministe. Les langages naturels contiennent beaucoup d'ambigüité et les règles sont plus complexes. Un algorithme conçu pour l'analyse syntaxique des langages naturels est **CKY**.

2.1 Algorithme CKY

L'algorithme de **Cocke-Kasami-Younger** (**CKY**) utilise la programmation dynamique afin d'appliquer une analyse syntaxique ascendante. La seule condition pour appliquer cet algorithme est de transformer la grammaire $G < \Sigma, N, P, S >$ sous la forme normale de Chomsky. Un petit rappel de cette forme (N est l'ensemble des variables et Σ est le vocabulaire) :

$$A \rightarrow BC \text{ où } A, B, C \in N$$
$$A \rightarrow w \text{ où } w \in \Sigma$$

Après transformation de la grammaire $G < \Sigma, N, P, S >$ sous FNC, on peut l'utiliser pour la reconnaissance des phrases. Étant donné une phrase $w = w_1 \dots w_n$, on crée un tableau triangulaire T de taille $n*n/2$. L'algorithme 5.1 décrit comment remplir ce tableau afin d'analyser le mot w suivant la grammaire G en FNC. Je l'ai modifié un peu pour pouvoir revenir en arrière et créer l'arbre syntaxique : chaque cellule contient un ensemble des quadruplets (variable, index des fils, position du 1ier fils, position du deuxième fils). L'algorithme prend un temps $O(n^3 * |P|)$ où P est l'ensemble des productions. On commence par remplir le diagonal du tableau T par les variables A qui génèrent les mots w_i de la phrase w ($A \rightarrow w_i$). On continue le remplissage du bas vers le haut et du gauche vers le droit. Chaque cellule est remplie par une variable qui génère une des variables à gauche (colonne k) suivie par une des variables en bas (ligne k) où $i \leq k \leq j$. Lorsqu'on arrive à la dernière cellule de la première ligne, on doit trouver l'axiome "S" ; sinon, la phrase n'appartient pas au langage.

Algorithme 5.1 : *Reconnaissance d'une phrase en utilisant la méthode CKY*

Données : une grammaire $G < \Sigma, N, P, S >$ en FNC ; une phrase $w = w_1 \dots w_n$
Résultat : $T[n, n]$
pour $i = 1 \dots n$ **faire**
    $T[i, i] \leftarrow \{(A, 0, 0, 0) / (A \rightarrow w_i) \in P\}$;
fin
pour $i = (n-1) \dots 1$ **faire**
    pour $j = (i+1) \dots n$ **faire**
        pour $k = i \dots (j-1)$ **faire**
            pour tous les A *tel que* $(A \rightarrow BC) \in P$ *et* $B \in T[i, k]$ *et* $C \in T[k+1, j]$ **faire**
                $iB \leftarrow index(B, T[i, k])$;
                $iC \leftarrow index(C, T[k+1, j])$;
                $T[i, j] \leftarrow T[i, j] \cup \{(A, k, iB, iC)\}$;
            fin
        fin
    fin
fin
si "S" $\notin T[1, n]$ **alors**
    Erreur "La phrase n'a pas été reconnue" ;
fin

Une fois la phrase w acceptée et le tableau rempli, on utilise ce denier pour générer l'arbre syntaxique. On commence par choisir les deux fils de l'axiome S en se basant sur T. On cherche les fils de chaque nœud

récursivement jusqu'à arriver à un nœud sans fils. L'algorithme 5.2 décrit l'opération de construction d'un arbre syntaxique en détail.

Algorithme 5.2 : *Construction de l'arbre syntaxique en utilisant CKY*

Données : $T[n,n]$
Résultat : Racine de l'arbre syntaxique : $r \leftarrow \varnothing$
si "S" $\in T[1,n]$ **alors**
    $r \leftarrow$ `Construire`$(1, n, index(\text{“}S\text{”}, T[1,n]))$;
fin
Fonction `Construire`(i, j, pos)
Début
    $(A, k, iB, iC) \leftarrow T[i, j][pos]$;
    Créer un nouveau nœud : nœud ;
    nœud.valeur $\leftarrow A$;
    si $k > 0$ **alors**
        nœud.gauche $\leftarrow$ `Construire`(i, k, iB) ;
        nœud.droit $\leftarrow$ `Construire`$(k+1, j, iC)$;
    fin
    retourner nœud ;
Fin

L'arbre syntaxique résultant est binaire (à cause de la forme normale de Chomsky). Mais, dans la réalité il doit suivre la grammaire conçue par les syntacticiens. Une solution est d'ajouter une étape de post-traitement pour récupérer l'arbre original. Concernant les productions unitaires, on peut les laisser telles qu'elles sont et modifier l'algorithme **CKY** afin de les accepter. Bien sûr l'algorithme souffre toujours du problème d'ambigüité syntaxique. Exemple, "Je mange du riz avec une fourchette" et "Je mange du riz avec de la viande".

On va analyser la phrase "la petite forme une petite phrase" en utilisant **CKY** selon la grammaire suivante :

- S → NP VP | VP
- VP →V NP
- NP → **DET ADJ N** | DET N | PRON
- PRON → je | tu | il | elle
- V → forme | veut | mange
- DET → un | une | la | le
- ADJ → petite | grand | bleu
- N → petite | forme | phrase | chat | poisson

On transforme cette grammaire en FNC en remplaçant la règle en gras par :

- NP → DET AP
- AP → ADJ N

La règle unitaire $S\ VP$ sera remplacée par $S\ VNP$. Le tableau d'analyse est illustré dans la figure 5.3.

la	petite	forme	une	petite	phrase
(DET, 0, 0, 0)	**(NP, 1, 1, 1)**	-	-	-	**(S, 2, 1, 1)**
	(ADJ, 0, 0, 0); (N, 0, 0, 0)	(AP, 2, 1, 2)	-	-	-
		(V, 0, 0, 0); (N, 0, 0, 0)	-	(VP, 3, 1, 1)	**(VP, 3, 1, 1)**; (S, 3, 1, 1)
			(DET, 0, 0, 0)	(NP, 4, 1, 2)	**(NP, 4, 1, 1)**
				(ADJ, 0, 0, 0); (N, 0, 0, 0)	**(AP, 5, 1, 1)**
					(N, 0, 0, 0)

Figure 5.3 : *Exemple de l'analyse CKY de la phrase "la petite forme une petite phrase"*

2.2 Algorithme CKY probabiliste

Dans l'algorithme **CKY**, on peut tomber sur un cas où on a plus d'un arbre syntaxique. Pour guider le choix des règles, on ajoute la probabilité de chaque production ; utilise une PCFG. Lors de la transformation d'une grammaire probabiliste $G < \Sigma, N, P, S >$ en FNC, les nouvelles règles créées par transformation en FNC ont une probabilité égale à 1. Étant donné un arbre syntaxique T du mot w, sa probabilité est estimée selon l'équation 5.2.

$$P(T, w) = \prod_{(A_i \to \beta_i) \in T} P(A_i \to \beta_i) \tag{5.2}$$

L'arbre syntaxique le plus adéquat pour analyser le mot w est celui avec le maximum de probabilité (voir 5.3).

$$\hat{T}(w) = \arg\max_{T(w)} P(T, w) \tag{5.3}$$

Concernant l'algorithme **CKY**, on ajoute la probabilité d'occurrence d'une règle dans chaque cellule. Chaque variable A aura au maximum une chance pour produire deux variables. Dans ce cas, on va créer un tableau triangulaire de trois dimensions $T[n, n, |N|]$ où on stocke la probabilité de chaque variable dans cette cellule en plus de la position k utilisée pour chercher les fils et les deux variables des fils gauche et droit. On suppose que toutes les probabilités des cellules sont initialisées par 0. L'algorithme 5.3 représente la version probabiliste de l'algorithme **CKY**.

3 Analyse de dépendances

Les dépendances syntaxiques sont des relations binaires entre deux mots. L'analyse des dépendances sert à trouver ces relations et créer un arbre syntaxique. Dans cet arbre, tous les nœuds sont des mots et chaque arc est une relation. On va présenter deux approches pour analyser les dépendances : par transitions et par graphes.

3.1 Par transition

Un analyseur des dépendances par transition peut être implémenté en utilisant la méthode SHIFT-REDUCE. C'est une machine abstraite ayant la configuration $C = (\sigma, \beta, A)$ (voir la figure 5.4) où :

— σ est une pile

Algorithme 5.3 : *Reconnaissance d'une phrase en utilisant la méthode CKY probabiliste*

Données : une grammaire $G < \Sigma, N, P, S >$ en FNC ; une phrase $w = w_1 \dots w_n$
Résultat : $T[n, n, |N|]$
pour $i = 1 \dots n$ **faire**
  $T[i, i, A] \leftarrow \{(P(A \rightarrow w_i), 0, A, A)/(A \rightarrow w_i) \in P\}$;
fin
pour $i = (n-1) \dots 1$ **faire**
  pour $j = (i+1) \dots n$ **faire**
    pour $k = i \dots (j-1)$ **faire**
      pour tous les A *tel que* $(A \rightarrow BC) \in P$ *et* $T[i, k, B] > 0$ *et* $T[k+1, j, C] > 0$ **faire**
        $p \rightarrow P(A \rightarrow BC) * T[i, k, B][1] * T[k+1, j, C][1]$;
        si $p > T[i, j, A][1]$ **alors**
          $T[i, j, A] \leftarrow (p, k, B, C)$;
        fin
      fin
    fin
  fin
fin
si $T[1, n, S] = 0$ **alors**
  Erreur "La phrase n'a pas été reconnue" ;
fin

— β est le tampon (buffer) d'entrée. Il contient le mot en entrée avec une tête qui pointe sur le mot courant.
— A est la liste des arcs créés (dépendances)

L'analyse commence avec la configuration $C_{initiale} = ([ROOT], w, \emptyset)$ et termine avec la configuration $C_{finale} = ([ROOT], \varnothing, A)$. Si on arrive à la fin du mot avec une pile non vide (on considère le sommet $ROOT$ comme vide) sans actions de vidage, on peut conclure que ce mot n'appartient pas au langage. Une transition d'un état vers un autre peut être une action sur :
— σ : empiler ou dépiler un mot
— β : retirer un mot ou ajouter un au début
— A : ajouter une dépendance entre 2 mots

Oracle est un système qui décide la transition suivante.

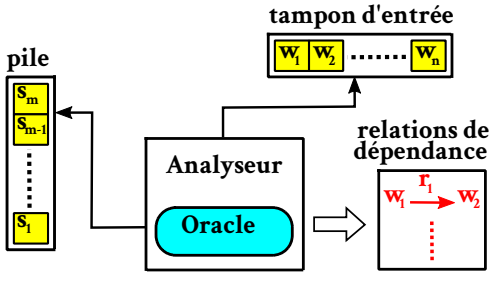


Figure 5.4 : *Architecture d'un analyseur des dépendances par transition*

Afin de décrire l'algorithme d'analyse, on va utiliser deux fonctions : "*Oracle*" qui choisit une transition

"*t*", et "*Appliquer*" qui exécute "*t*" sur la configuration. L'algorithme 5.4 décrit l'analyse des dépendances par transitions.

Algorithme 5.4 : *Analyse des dépendances par transitions*

Données : Le mot à analyser $w = w_1 w_2 \dots w_n$

Résultat : Liste des dépendances A

$C \leftarrow (\sigma = [ROOT], \beta = w, A = \emptyset)$;

tant que $\sigma \neq [ROOT]$ $OU \beta \neq \varnothing$ **faire**

  $t \leftarrow Oracle(C)$;

  $C \leftarrow Appliquer(C, t)$;

fin

Le système "Oracle" choisit la transition suivante $\hat{t}$ parmi l'ensemble des transitions possibles T. Lorsqu'on est dans la configuration actuelle $C = (\sigma, \beta, A)$, on utilise une fonction Ψ qui calcule un score en utilisant des caractéristiques basées sur cette configuration. La transition $\hat{t}$ choisie est celle qui maximise la fonction Ψ ayant des paramètres θ selon l'équation 5.4.

$$\hat{t} = \arg\max_{t \in T} \Psi(t, C, w; \theta) \tag{5.4}$$

Pour entraîner le système "Oracle", on doit annoter le texte en le transformant à une séquence de transitions. Le texte original est annoté avec les relations de dépendance. Pour transformer la sortie à une séquence de transitions, on utilise le système d'analyse en tel sorte à générer toutes les dépendances possibles. En utilisant des caractéristiques sur la configuration courante, on essaye d'estimer la transition suivante par la fonction Ψ. Cette fonction peut être MaxEnt (la plus utilisée), SVM ou des réseaux de neurones. Les caractéristiques utilisées par la fonction Ψ peuvent être des caractéristiques sur :

- la pile σ : le mot dans le sommet de la pile et sa catégorie grammaticale.
- le tampon d'entrée β : les trois premiers mots et leurs catégories grammaticales.
- la liste des dépendances A : les dépendances qui ont été estimées.
- la phrase analysée w : la distance entre le mot du sommet de la pile et le premier mot dans le tampon (nombre des mots entre eux dans la phrase w)

Il y a deux approches pour l'analyse de dépendances par transitions : Arc-standard et Arc-eager. Dans Arc-standard, les relations de dépendances sont détectées seulement entre les mots dans la pile. Lorsqu'un mot est considéré comme une tête d'une relation, il sera éliminé de la pile. La figure 5.5 représente un exemple d'analyse de la phrase "they like bagels with lox" en utilisant Arc-standard. Les transitions possibles dans cette approche sont :

- **SHIFT** : déplacer le premier élément dans le tampon vers la pile

$$(\sigma, w_i|\beta, A) \Rightarrow (\sigma|w_i, \beta, A)$$

- **ARC-LEFT** : établir un arc du premier élément dans le tampon vers le sommet de la pile

$$(\sigma|w_i, w_j|\beta, A) \Rightarrow (\sigma, w_j|\beta, A \cup \{w_j \rightarrow w_i\})$$

- **ARC-RIGHT** : établir un arc du sommet de la pile vers le premier élément dans le tampon

$$(\sigma|w_i, w_j|\beta, A) \Rightarrow (\sigma, w_i|\beta, A \cup \{w_i \rightarrow w_j\})$$

	σ	β	action	arc added to $\mathcal{A}$
1.	[ROOT]	*they like bagels with lox*	SHIFT	
2.	[ROOT, *they*]	*like bagels with lox*	ARC-LEFT	(*they* ← *like*)
3.	[ROOT]	*like bagels with lox*	SHIFT	
4.	[ROOT, *like*]	*bagels with lox*	SHIFT	
5.	[ROOT, *like*, *bagels*]	*with lox*	SHIFT	
6.	[ROOT, *like*, *bagels*, *with*]	*lox*	ARC-LEFT	(*with* ← *lox*)
7.	[ROOT, *like*, *bagels*]	*lox*	ARC-RIGHT	(*bagels* → *lox*)
8.	[ROOT, *like*]	*bagels*	ARC-RIGHT	(*like* → *bagels*)
9.	[ROOT]	*like*	ARC-RIGHT	(ROOT → *like*)
10.	[ROOT]	∅	DONE	

Figure 5.5 : *Exemple de dérivations non étiquetées de la phrase "they like bagels with lox" en utilisant Arc-standard (Eisenstein, 2018)*

Dans Arc-Eager, on essaye de détecter les relations de dépendance le plus tôt possible. Pour ce faire, les relations doivent être détectées entre le sommet de la pile et le mot courant. Dans ce cas, on n'avance pas la tête de lecture automatiquement lorsqu'on ajoute une nouvelle relation. Dans ce cas, la seule méthode pour le faire est d'empiler le mot dans la pile en utilisant la transition "SHIFT". Un exemple de l'analyse de la phrase "they like bagels with lox" en utilisant Arc-eager est illustré dans la figure 5.6. Les transitions possibles dans cette approches sont :

— **SHIFT** est le même que "Arc-standard"

— **ARC-LEFT** : établir un arc du premier élément dans le tampon vers le sommet de la pile

$$(\sigma|w_i, w_j|\beta, A) \xrightarrow{\forall w_k (w_k \rightarrow w_i) \notin A} (\sigma, w_j|\beta, A \cup \{w_j \rightarrow w_i\})$$

— **ARC-RIGHT** : établir un arc du sommet de la pile vers le premier élément dans le tampon

$$(\sigma|w_i, w_j|\beta, A) \Rightarrow (\sigma|w_i w_j, \beta, A \cup \{w_i \rightarrow w_j\})$$

— **REDUCE** : dépiler un mot s'il a déjà un parent

$$(\sigma|w_i, \beta, A) \xrightarrow{\exists w_k (w_k \rightarrow w_i) \in A} (\sigma, \beta, A)$$

	σ	β	action	arc added to $\mathcal{A}$
1.	[ROOT]	*they like bagels with lox*	SHIFT	
2.	[ROOT, *they*]	*like bagels with lox*	ARC-LEFT	(*they* ← *like*)
3.	[ROOT]	*like bagels with lox*	ARC-RIGHT	(ROOT → *like*)
4.	[ROOT, *like*]	*bagels with lox*	ARC-RIGHT	(*like* → *bagels*)
5.	[ROOT, *like*, *bagels*]	*with lox*	SHIFT	
6.	[ROOT, *like*, *bagels*, *with*]	*lox*	ARC-LEFT	(*with* ← *lox*)
7.	[ROOT, *like*, *bagels*]	*lox*	ARC-RIGHT	(*bagels* → *lox*)
8.	[ROOT, *like*, *bagels*, *lox*]	∅	REDUCE	
9.	[ROOT, *like*, *bagels*]	∅	REDUCE	
10.	[ROOT, *like*]	∅	REDUCE	
11.	[ROOT]	∅	DONE	

Figure 5.6 : *Exemple de dérivations non étiquetées de la phrase "they like bagels with lox" en utilisant Arc-eager (Eisenstein, 2018)*

Les relations présentées ici sont des relations non étiquetées ; juste la relation tête-dépendant sans type. Afin d'ajouter le type de la relation, on doit enrichir les transitions "ARC-LEFT" et "ARC-RIGHT" avec

les types possibles. Dans Arc-standard, à la place de 3 transitions, on aura $1 + 2R$ transitions où R est le nombre des types de dépendances.

3.2 Par graphe

Le filtrage des relations non probables est une autre approche pour trouver l'arbre de dépendance d'un mot donné w. Dans un premier temps, on considère toutes les dépendances possibles entre les mots. Ensuite, on commence à éliminer les relations non probables jusqu'à arriver au résultat final. D'une façon plus formelle, on commence l'analyse par un graphe complet $G = (V, E)$ où V est l'ensemble de nœuds (mots) et E l'ensemble d'arcs (relations de dépendance). Ensuite, on cherche l'arbre $T = (V, F)$ parmi les arbres possibles $\mathcal{T}(G)$ qui est un sous-graphe de G maximisant un certain score comme indiqué dans l'équation 5.5.

$$\hat{T} = \arg \max_{T \in \mathcal{T}(G)} \Psi(T, w; \theta) \tag{5.5}$$

Le score Ψ d'un arbre T est la somme des scores ψ de ces arcs (voir l'équation 5.6). θ est l'ensemble des paramètres utilisés dans la fonction du score.

$$\Psi(T, w; \theta) = \sum_{e \in F/T=(V,F)} \psi(e, w; \theta) \tag{5.6}$$

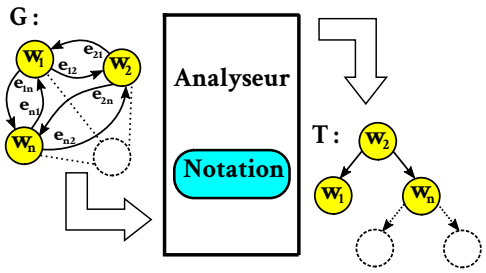


Figure 5.7 : *Architecture d'un analyseur des dépendances par graphe*

Afin d'estimer le score d'un arc e, on peut utiliser un certain nombre de caractéristiques f comme : le mot de l'entête, sa catégorie grammaticale, son lemme, ses préfixes et suffixes, la direction de l'arc, la distance entre l'entête et le dépendant, etc. Un algorithme d'apprentissage ayant les paramètres θ est utilisé pour apprendre à estimer le score d'un arc e selon ces caractéristiques. En utilisant des scores initiaux, on génère un arbre et on le compare avec l'arbre destinataire. Si les arbres ne sont pas identiques, on calcule l'erreur et on mis à jours les paramètres. Le score d'un arc e est représenté par l'équation 5.7.

$$\psi(e, w; \theta) = \sum_{k=1}^{K} \theta_k f_k(e, w) \tag{5.7}$$

Afin de créer un arbre à partir d'un graphe, on peut utiliser l'algorithme de Chu-Liu/Edmonds (voir l'algorithme 5.5). Pour analyser un mot $w = w_1 \dots w_n$, on commence par construire un graphe complet $G = (V, E)$ où chaque nœud v_i représente un mot w_i et chaque arc e représente une relation possible entre deux nœuds. Afin de représenter l'information du mot racine, on ajoute un nœud (ROOT) avec des arcs vers tous le reste des nœuds. Pour chaque arc e du graphe G, on affecte un poids $G.p(e)$ qui est estimé en utilisant l'apprentissage automatique. En prenant le nœud "ROOT" comme racine et le graphe G, on essaye de trouver un arbre $T = (V, F)$ couvrant de poids maximal. Un arbre couvrant est un sous-graphe acyclique maximal où tous les nœuds sont connectés et il n'y a plus qu'un arc entrant vers un nœud. Deux fonctions sont utilisées dans cette algorithme :

— **Contracter** : une fonction qui fusionne deux nœuds u et v composant un cycle C
  — $\forall e = (u', v) \in E : G.p(e) \leftarrow G.p(e) - G.p((u, v))$
  — $\forall e = (v', u) \in E : G.p(e) \leftarrow G.p(e) - G.p((v, u))$

— **Etendre** : une fonction qui désassemble les deux nœuds u et v d'un cycle C. L'arc qui enfreint la condition "*pas de deux arcs entrants*" est supprimé.

Algorithme 5.5 : *Analyse de Chu-Liu-Edmonds : Arbre couvrant de poids maximal*

Données : un graphe pondéré $G = (V, E), ROOT$
Résultat : un arbre couvrant $T = (V, F)$
Fonction `ArbreCouvrantMax`$(G, ROOT)$
  $F \leftarrow \emptyset$;
  pour tous les $v \in V$ **faire**
    $meilleurInArc \leftarrow \arg\max_{e=(u,v)\in E} G.p(e)$; $F \leftarrow F \cup meilleurInArc$;
    pour tous les $e = (u, v) \in E$ **faire**
      $G.p(e) \leftarrow G.p(e) - G.p(meilleurInArc)$;
    fin
    si $T = (V, F)$ *est un arbre couvrant* **alors**
      retourner T ;
    sinon
      $C \leftarrow$ un cycle de F ; $G' \leftarrow Contracter(G, C)$;
      $T' \leftarrow ArbreCouvrantMax(G', ROOT)$; $T \leftarrow Etendre(T', C)$;
      retourner T ;
    fin
  fin
Fin

Prenons l'exemple de la phrase "Book a flight". La figure 5.8 représente son analyse en utilisant l'algorithme de Chu-Liu-Edmonds.

Discussion

Qu'est ce qu'un langage sans syntaxe ? Essayer d'imaginer un peuple qui parle une langue (exemple, le français) sans utiliser des règles grammaticales et en utilisant seulement le vocabulaire. Aucune personne ne sera capable de comprendre l'autre. Certes, on peut avoir les éléments de la phrase ; mais sans structure, on ne peut pas savoir leurs rôles et leurs relations.

Il existe plusieurs théories de la structure syntaxique. Décrire toutes ces théories veut dire rédiger tout un livre juste pour la syntaxe. On s'intéresse à deux structures principales : constituante et fonctionnelle. La première commence à décomposer la phrase en syntagmes jusqu'à arriver à un seul mot (méthode utilisée pour enseigner la langue aux élèves du primaire). La deuxième essaye de trouver des relations syntaxiques binaires entre les mots.

L'approche la plus célèbre pour représenter les constituants est la grammaire à contexte libre. Afin d'analyser une telle structure, on peut utiliser l'algorithme CKY. Cet algorithme est amélioré en ajoutant des probabilités aux règles afin d'avoir une analyse déterministe. Il existe des méthodes qui utilisent des techniques plus récentes comme BERT (voir le chapitre suivant). Pour évaluer une analyse automatique, on

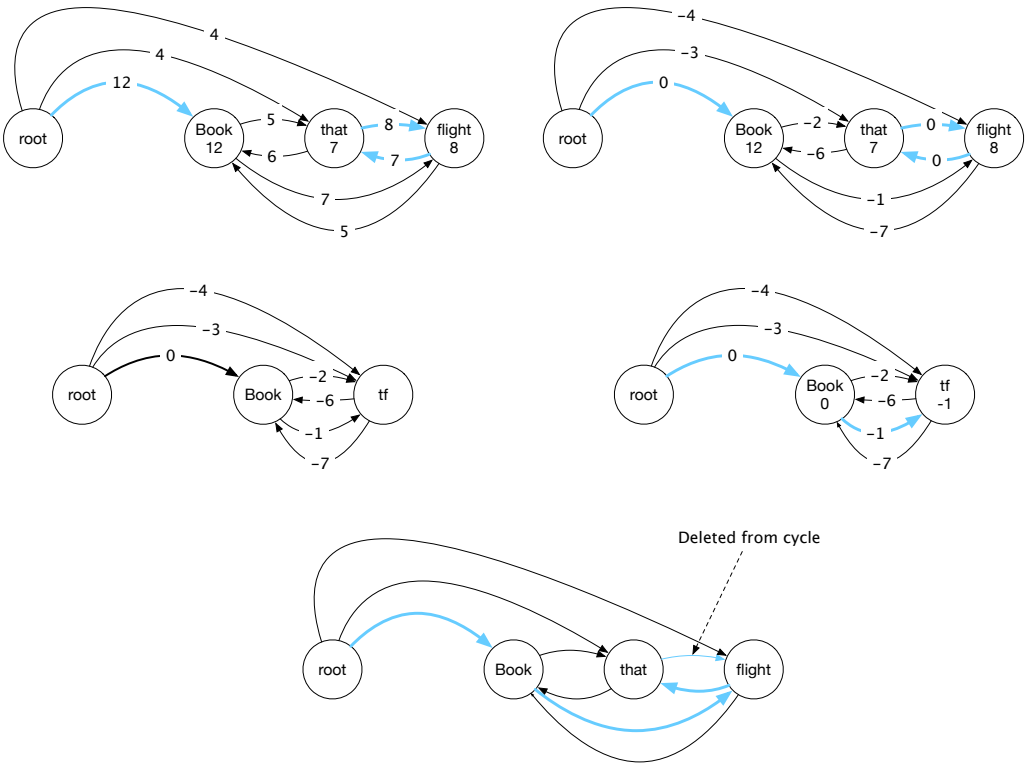


Figure 5.8 : *Exemple de l'analyse de la phrase "Book a flight" en utilisant l'algorithme de Chu-Liu-Edmonds (Jurafsky et Martin, 2019)*

peut utiliser le nombre des phrases ayants des arbres syntaxiques justes. Cette mesure ne peut pas différencier entre deux arbres syntaxiques erronées ; il se peut qu'une des deux est plus juste qu'une autre.

L'analyse de dépendances peut être accomplie en considérant les relations comme des transitions et donc utiliser un automate à pile pour les détecter. Une autre approche est de considérer toutes les relations possibles et d'utiliser les propriétés des graphes afin de trouver un arbre syntaxique. Cette dernière approche est plus adéquate pour les relations à long terme ; lorsque la phrase est trop longue. Dans le cas des dépendances, on peut évaluer un arbre en utilisant le nombre des relations correctes.

CHAPITRE 6

SÉMANTIQUE LEXICALE

LES LANGUES

Nous pouvons comprendre un texte en utilisant le sens de chaque mot qui lui compose. Un jour quelqu'un décida d'inventer les métaphores, ce qui a résulté au phénomène de polysémie ; un mot avec plusieurs sens. Depuis ce jour, les être humains ont devenu perdus ! Afin de traiter les mots automatiquement, on doit les encoder. Il y a deux représentations : en se basant sur les relations sémantiques avec les autres mots ou en utilisant une représentation vectorielle. Dans la première représentation, les mots polysémiques peuvent avoir des différents codes selon leurs sens. Dans la représentation vectorielle, on peut choisir de représenter un mot par un seul vecteur quelque soit son sens ou d'utiliser une représentation plus avancée qui se base sur le contexte (mots voisins). Dans ce chapitre, on va présenter les deux approches : bases de données lexicales et représentation vectorielle.

Un mot, comme unité de traitement du texte, doit être encodé afin de faciliter les tâches appliquées sur le texte. Une information est encodée dans le cerveau humain en utilisant un phénomène appelé : potentialisation à long terme. C'est le point de vue de l'approche physiologique. Quand à l'approche mentale, parmi les types d'encodage, on peut citer l'encodage visuel. Dans ce cas, la représentation mentale d'un mot n'a aucune relation avec la langue parlée par l'individu. Par exemple, les mots "شَجَرَة /shajarah/" en arabe, "tree" en anglais, "arbre" en français et "木 /ki/" en japonais font référence au même concept : un objet avec un tronc sur lequel s'insèrent des branches ramifiées portant le feuillage. En informatique, représenter les concepts en utilisant des images augmente la taille du stockage et le temps du traitement. Plutôt, on utilise un encodage moins couteau comme les relations avec les autres concepts ou encore un vecteur de nombres. Une représentation est meilleure lorsqu'elle peut faire face à l'ambigüité. Prenons trois phrases, en français, qui contiennent le mot "café" mais avec trois sens différents :

— Je veux boire du café.
— Je veux aller au café.
— Je veux récolter du café.

Le mot "café" dans ce cas n'aura pas la même représentation mentale. Dans la première phrase, ce mot représente un liquide ; sauf si on peut imaginer quelqu'un qui boit une matière solide (les grains du café). Dans la deuxième phrase, le mot "café" représente une place ; on ne peut pas aller vers un liquide sauf si on le considère comme une place. Lorsqu'on parle d'une place, il est commun de considérer le cafétéria et pas la place où se trouve un verre du café. La dernière phrase utilise le mot "café" comme un produit recueilli de la terre. Je pense la motivation de ce chapitre est claire : encodage des mots en se focalisant sur le niveau sémantique.

1 Bases de données lexicales

Une base lexicale est une de données contenant le vocabulaire d'une langue. En plus du vocabulaire, elle fourni des informations concernant chaque mot : catégorie grammaticale, lemme, fréquence, etc. Ces informations sont liées entre elles par des relations. La figure 6.1 représente une base lexicale. Un mot fait parti d'un lexème qui est représenté par un et un seul lemme. Un lexème peut représenter plusieurs sens ; qui sont référencés ici par **Synset** et définis par un glossaire.

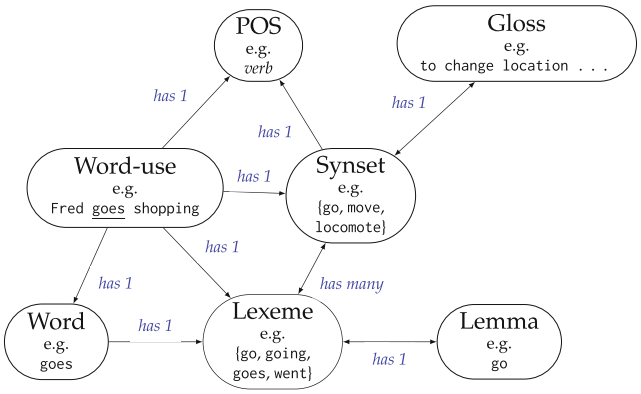


Figure 6.1 : *Exemple des informations et leurs relations dans une base lexicale (White et al., 2019)*

1.1 Relations sémantiques

Rappelons les relations sémantiques présentées dans le premier chapitre :

- **Synonymie** : avoir des sens similaires dans un contexte donné
- **Antonymie** : avoir des sens opposés dans un contexte donné
- Les relations taxonomiques (de classification)
  - **Hyponymie** : être plus spécifique qu'un autre sens. Il entraîne un relation **IS-A**. Ex. "voiture IS-A véhicule ".
  - **Hyperonymie** : être plus générique qu'un autre sens.
  - **Méronymie** : être une partie d'une chose. Ex. "roue est un méronyme de voiture ; voiture est le holonyme de roue".

1.2 Wordnet

WordNet (Miller, 1995) est une base de données lexicale pour l'anglais. La figure 6.2 représente un exemple des différents sens du mot "reason". La base de données contient trois parties : (1) noms (2) verbes (3) adjectifs et adverbes. Un sens regroupe plusieurs mots de la même partie et il est représénté par un identifiant appelé **Synset** (Synonyms set). Par exemple, 05659525 : reason#3, understanding#4, intellect#2. Un sens est défini par un glossaire (**Gloss**). Exemple, 05659525 : (the capacity for rational thought or inference or discrimination) "we are told that man is endowed with reason and capable of distinguishing good from evil".

Un sens est marqué par une catégorie lexicographique (**supersense**). Exemple. 05659525 : noun.cognition
Le tableau 6.1 représente les catégories lexicographiques utilisées dans Wordnet.

WordNet représente les relations sémantiques entre les sens. Le tableau 6.2 représente les relations sémantiques des noms et des verbes.

Noun

- S: (n) **reason**, ground (a rational motive for a belief or action) *"the reason that war was declared"; "the grounds for their declaration"*
- S: (n) **reason** (an explanation of the cause of some phenomenon) *"the reason a steady state was never reached was that the back pressure built up too slowly"*
- S: (n) **reason**, understanding, intellect (the capacity for rational thought or inference or discrimination) *"we are told that man is endowed with reason and capable of distinguishing good from evil"*
- S: (n) rationality, **reason**, reasonableness (the state of having good sense and sound judgment) *"his rationality may have been impaired"; "he had to rely less on reason than on rousing their emotions"*
- S: (n) cause, **reason**, grounds (a justification for something existing or happening) *"he had no cause to complain"; "they had good reason to rejoice"*
- S: (n) **reason** (a fact that logically justifies some premise or conclusion) *"there is reason to believe he is lying"*

Verb

- S: (v) **reason**, reason out, conclude (decide by reasoning; draw or come to a conclusion) *"We reasoned that it was cheaper to rent than to buy a house"*
- S: (v) argue, **reason** (present reasons and arguments)
- S: (v) **reason** (think logically) *"The children must learn to reason"*

Figure 6.2 : *Exemple de WordNet généré par* `http://wordnetweb.princeton.edu/perl/webwn`

Catégorie	Exemple	Catégorie	Exemple	Catégorie	Exemple
ACT	service	GROUP	place	PLANT	tree
ANIMAL	dog	LOCATION	area	POSSESSION	price
ARTIFACT	car	MOTIVE	reason	PROCESS	process
ATTRIBUTE	quality	NATURAL EVENT	experience	QUANTITY	amount
BODY	hair	NATURAL OBJECT	flower	RELATION	portion
COGNITION	way	OTHER	stuff	SHAPE	square
COMMUNICATION	review	PERSON	people	STATE	pain
FEELING	discomfort	PHENOMENON	result	SUBSTANCE	oil
FOOD	food			TIME	day

Tableau 6.1 : *Catégories lexicographiques des noms dans WordNet (Jurafsky et Martin, 2019)*

Il existe des projets pour porter Wordnet aux autres langues. Parmi ces projets, on peut citer : Global WordNet Association [1] et Open Multilingual Wordnet [2]. Afin de naviguer Wordnet, il existe plusieurs APIs. Voici une petite liste de ces APIs :

— NLTK (Python) : `https://www.nltk.org/howto/wordnet.html` [visité le 2021-09-25]
— JWI (Java) : `http://projects.csail.mit.edu/jwi` [visité le 2021-09-25]
— Wordnet (Ruby) : `https://github.com/wordnet/wordnet` [visité le 2021-09-25]
— OpenNlp (C#) : `https://github.com/AlexPoint/OpenNlp` [visité le 2021-09-25]

1.3 Autres ressources

WordNet n'est pas la seule base de donnée lexicale ; il existe d'autres. VerbNet est une autre base de don-

1. Global WordNet Association : `http://globalwordnet.org/resources/wordnets-in-the-world/` [visité le 2021-09-11]
2. Open Multilingual Wordnet : `http://compling.hss.ntu.edu.sg/omw/` [visité le 2021-09-11]

Noms		
Relation	**Définition**	**Exemple**
Hypernym	d'un concept spécifique vers un autre générique	breakfast[1] →meal[1]
Hyponym	d'un concept générique vers un autre spécifique	meal[1] →lunch[1]
Instance Hypernym	d'une instance vers son concept	Austen[1] →author[1]
Instance Hyponym	d'un concept vers son instance	composer[1] →Bach[1]
Part Meronym	d'un concept entier vers une partie	table[2] →leg[3]
Part Holonym	d'une partie vers un entier	course[7] →meal[1]
Antonym	d'un concept vers son opposition sémantique	leader[1] ↔ follower[1]
Derivation	d'un mot vers un autre ayant la même racine	destruction[1] ↔ destroy[1]
Verbes		
Relation	**Définition**	**Exemple**
Hypernym	d'un évènement spécifique vers un autre générique	fly[9] →travel[5]
Troponym	d'un évènement générique vers un autre spécifique	walk[1] →stroll[1]
Entails	d'un évènement vers un autre qui l'implique	snore[1] →sleep[1]
Antonym	d'un évènement vers son opposition sémantique	increase[1] ↔ decrease[1]

Tableau 6.2 : *Relations sémantiques de Wordnet (Jurafsky et Martin, 2019)*

née lexicale, mais seulement pour les verbes. Elle inclue 30 rôles thématiques principaux. Les verbes sont organisés en classes. **FrameNet** est une autre base de donnée lexicale basée sur la théorie du sens appelée "cadre sémantique" (frame semantic). Un cadre peut être un évènement, une relation ou un entité avec ces participants. Par exemple, le concept "Cuisinier" implique une personne qui cuisine, la nourriture, un récipient et une source de chaleur. Chaque cadre est activé par un ensemble des unités lexicales. Exemple, blanchir, bouillir, griller, dorer, mijoter, cuire. Ces deux projets vont être repris dans le chapitre suivant (sens des propositions).

Un autre projet similaire à Wordnet est BabelNet [3]. C'est un réseau sémantique multilingue qui utilise plusieurs sources comme Wordnet, Wikipédia, VerbNet, etc. Chaque concept a un synset comme dans **WordNet** et qui est partagé par plusieurs langues. La structure de BabelNet est illustrée dans la figure 6.3.

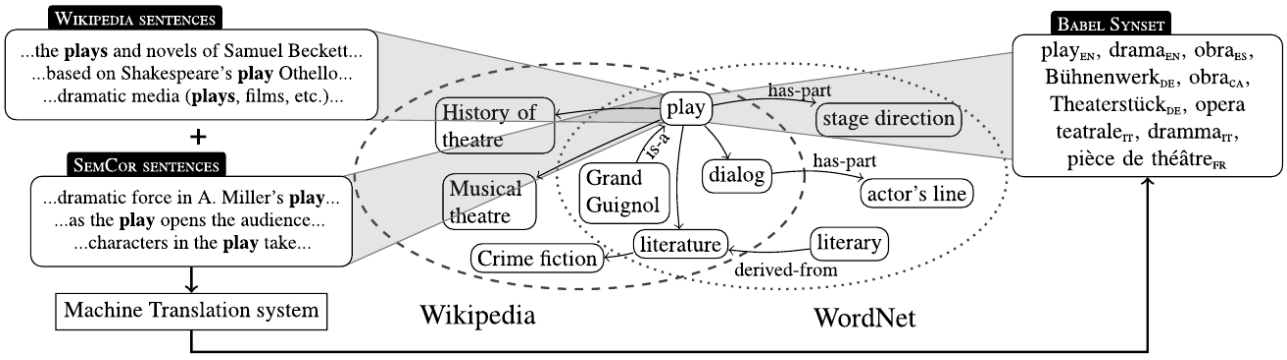


Figure 6.3 : *Structure de BabelNet (Navigli et Ponzetto, 2012)*

3. BabelNet : https://babelnet.org/ [visité le 2021-09-25]

2 Représentation vectorielle des mots

La représentation vectorielle la plus simple est l'utilisation de l'encodage **One-Hot**. Dans cette représentation, on prend un vecteur de taille égale à la taille du vocabulaire et on choisit une position pour représenter un mot donné. Donc, le mot sera représenté avec un vecteur contenant des zéros sauf la position réservée à lui où il va avoir un 1. Cette représentation est vraiment couteuse en terme de taille. En plus, on ne peut pas vraiment représenter un document ou une phrase en utilisant cette représentation. Cela est dû au fait qu'elle ne représente pas les relations sémantiques entre les mots : similarité (Ex. chat, chien) et proximité (Ex. café, tasse).

Cette représentation peut être utilisée avec les algorithmes d'apprentissage automatique comme les réseaux de neurones. Mais, il existe d'autres représentations vectorielles qui sont plus informatives. Dans ce cas, un terme peut être représenté par rapport à une autre référence comme :

- **Terme-document** : on représente un terme par les documents qui le contiennent (ou l'inverse). Ex. Term Frequency - Inverse Document Frequency (TF-IDF).
- **Terme-terme** : on représente un terme par d'autres termes.
- **Terme-concept-document** : on représente les termes et les documents par un vecteur de concepts. Ex. Analyse sémantique latente ; Latent Semantic Analysis (LSA).

2.1 TF-IDF

Le sens d'un document ou d'une phrase peut être représenté par les mots qui lui composent et vice-versa. Donc, un document peut être représenté par les fréquences d'occurrence des mots de vocabulaire. Du même, un mot peut être représenté par les fréquences de ces occurrences dans chaque document. La fréquence d'un mot dans un document/phrase s'appelle "Term Frequency (TF)". Elle peut être calculée en comptant le nombre d'apparition d'un terme t dans un document d comme indiqué par l'équation 6.1.

$$TF_d(t) = |\{t_i \in d / t_i = t\}| \tag{6.1}$$

En général, cette représentation est utilisée dans les tâches de recherche d'information. Des fois, on veut attribuer plus de poids pour les nouveaux mots. Cela est motivé par le fait que les mots qui se répètent trop dans un domaine précis n'ont pas un sens ajouté. Par exemple, le mot "ordinateur" ne sera pas vraiment important si le domaine traité est l'informatique. Ca sera plus bénéfique de considérer les mots plus fréquents dans le document, mais qui ne sont pas aussi fréquents dans des documents du même domaine. Afin de calculer la nouveauté d'un terme t, on utilise un ensemble de documents D du même domaine. L'équation 6.2 représente la méthode de calcul de Inverse Document Frequency (IDF).

$$IDF_D(t) = \log_{10}\left(\frac{|\{d \in D\}|}{|\{d \in D / t \in d\}|}\right) \tag{6.2}$$

En multipliant l'importance du mot t dans le document d (TF) par sa nouveauté par rapport un ensemble de documents D (IDF), on aura son encodage normalisé : TF-IDF (voir l'équation 6.3).

$$TF\text{-}IDF_{d,D}(t) = TF_d(t) * IDF_D(t) \tag{6.3}$$

Prenons les trois phrases suivantes :

- S1 : un ordinateur peut vous aider
- S2 : il peut vous aider et il veut vous aider

— S3 : il veut un ordinateur et un ordinateur pour vous

Chaque document peut être représenté en utilisant un vecteur des mots du vocabulaire comme indiqué dans le tableau 6.3.

	un	ordinateur	peut	vous	aider	il	et	veut	pour
S1	1	1	1	1	1	0	0	0	0
S2	0	0	1	2	2	2	1	1	0
S3	2	2	0	1	0	1	1	1	1

Tableau 6.3 : *Exemple des représentations TF de trois documents*

Si nous avons deux documents a et b représentés par deux vecteurs $\overrightarrow{a}$ et $\overrightarrow{b}$ respectivement, on peut calculer leur similarité. La similarité la plus utilisée en TALN est la similarité cosinus indiquée dans l'équation 6.4. Les deux documents sont considérés totalement identiques si la similarité cosinus égale à 1.

$$Cos(\theta) = \frac{\overrightarrow{a}\,\overrightarrow{b}}{\|\overrightarrow{a}\|\,\|\overrightarrow{b}\|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\,\sqrt{\sum_{i=1}^{n} b_i^2}} \tag{6.4}$$

Maintenant, on va calculer les similarités cosinus entre les trois phrases précédentes. $Cos(S1, S2) = \frac{5}{\sqrt{5}\sqrt{15}} = \frac{1}{\sqrt{3}} = 0.577$. $Cos(S1, S3) = \frac{5}{\sqrt{5}\sqrt{13}} = 0.620$. $Cos(S2, S3) = \frac{6}{\sqrt{15}\sqrt{13}} = 0.429$. Dans ce cas, la première phrase est plus similaire à la troisième. Cette fois-ci, on veut un exemple plus concret. On va considérer seulement les deux mots "ordinateur" et "vous" pour représenter les trois phrases graphiquement comme indiqué dans la figure 6.4. Il est clair que la phrase $S1$ est plus proche de la phrase $S2$. En calculant les trois cosinus, on aura $Cos(S3, S1) = \frac{3}{\sqrt{5}\sqrt{2}} = 0.948$, $Cos(S1, S2) = \frac{2}{\sqrt{2}\sqrt{4}} = 0.707$, $Cos(S3, S2) = \frac{2}{\sqrt{5}\sqrt{4}} = 0.447$.

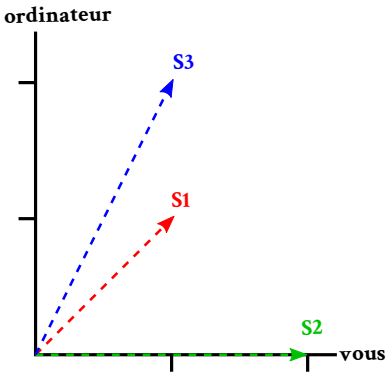


Figure 6.4 : *Représentation graphique des vecteurs TF de trois phrases en utilisant deux mots*

2.2 Mot-Mot

Un mot peut être représenté par rapport aux autres mots du vocabulaire en utilisant la co-occurrence. Afin de représenter les mots d'un vocabulaire V, on doit utiliser une matrice $|V| \times |V|$. Le vecteur des $|V|$ mots qui représentent un mot est appelé "contexte". La co-occurrence peut être calculée par rapport aux documents, aux phrases ou des fenêtres auteur du mot. Par rapport au document, on compte le nombre des documents où les deux mots ont apparus ensemble. Cela doit utiliser beaucoup de documents et donne

des vecteurs éparts (la plupart des valeurs sont des zéros). Une autre technique pour exprimer la co-occurrence est l'utilisation d'une fenêtre avec des mots avant et des mots après.

Prenons l'exemple des phrases précédente. On va les rappeler ici :

— S1 : un ordinateur peut vous aider
— S2 : il peut vous aider et il veut vous aider
— S3 : il veut un ordinateur et un ordinateur pour vous

Les parties soulignées sont un exemple d'une fenêtre 1-1 qui capture le contexte du mot "ordinateur". En utilisant la même fenêtre, le vocabulaire est encodé selon le tableau 6.4. Afin de calculer la similarité entre deux mots, on peut utiliser la similarité cosinus présentée précédemment. On peut remarquer que cet encodage soufre du problème des zéros ; il y a un gaspillage de l'espace. Un document peut être encoder comme le centre des vecteurs des mots qui lui composent.

	un	ordinateur	peut	vous	aider	il	et	veut	pour
un	0	3	0	0	0	0	1	1	0
ordinateur	3	0	1	0	0	0	1	0	1
peut	0	1	0	2	0	1	0	0	0
vous	0	0	2	0	3	0	0	1	1
aider	0	0	0	3	0	0	1	0	0
il	0	0	1	0	0	0	1	2	0
et	0	0	0	0	1	1	0	0	0
veut	1	0	0	1	0	2	0	0	0
pour	0	1	0	1	0	0	0	0	0

Tableau 6.4 : *Exemple d'encodage Mot-Mot*

2.3 Analyse sémantique latente (LSA)

Nous avons vu que les représentations terme-document et terme-terme sont souvent des vecteurs éparts ; contenant plusieurs zéros. En plus, leurs dimensions sont énormes surtout si notre langage est riche en mots. Revenons au premier chapitre, on a présenté une représentation appelée : analyse sémique. On choisit des propriétés pour représenter les mots en indiquant si la propriété existe ou non. Du même, on peut définir L concepts comme des propriétés abstraites (on ne sait pas c'est quoi ces propriétés).

Dans LSA, on veut représenter les N termes et les M documents en utilisant un vecteur de taille L comme deux matrices : $T[N, L]$ et $D[M, L]$ respectivement. Pour ce faire, on doit choisir le nombre des concepts (taille du vecteur) $L \leq \min(N, M)$. On commence par une matrice terme-document $X[N, M]$. Ensuite, on la décompose en utilisant la décomposition en valeurs singulières ; en anglais Singular Value Decomposition (SVD). Ceci revient à la représenter en utilisant les deux autres matrices : document-concept D et terme-concept T comme $X = T \times S \times D^\top$. L'équation 6.5 représente cette décomposition en plus de détail.

$$\overbrace{\begin{bmatrix} x_{11} & \dots & \dots & \dots & x_{1M} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ x_{N1} & \dots & \dots & \dots & x_{NM} \end{bmatrix}}^{X : \text{terme-document}} = \overbrace{\begin{bmatrix} \begin{bmatrix} t_{11} \\ \vdots \\ \vdots \\ t_{N1} \end{bmatrix} \dots \begin{bmatrix} t_{L1} \\ \vdots \\ \vdots \\ t_{L1} \end{bmatrix} \end{bmatrix}}^{T : \text{terme-concept}} \times \overbrace{\begin{bmatrix} s_{11} & \dots & 0 \\ 0 & \ddots & 0 \\ 0 & \dots & s_{LL} \end{bmatrix}}^{S : \text{concept-concept}} \times \overbrace{\begin{bmatrix} \begin{bmatrix} d_{11} & \dots & \dots & \dots & d_{1M} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} d_{11} & \dots & \dots & \dots & d_{1M} \end{bmatrix} \end{bmatrix}}^{D^\top : \text{concept-document}} \tag{6.5}$$

Dans cette composition, les deux vecteurs décomposés sont conditionnés comme dans l'équation 6.6 et l'équation 6.7.

$$T^\top T = \mathbb{I}_{N\times N} \tag{6.6}$$

$$D^\top D = \mathbb{I}_{M\times M} \tag{6.7}$$

Afin de résoudre SVD, on peut utiliser des méthodes de programmation dynamique comme l'algorithme de Lanczos et la décomposition QR.

La représentation d'un terme avec un vecteur des nombres réels est appelé "**embedding**". La plupart des **embeddings** connus actuellement utilisent les réseaux de neurones. Malgré c'est une représentation vectorielle, j'ai choisi de présenter les **embeddings** dans une section à part.

3 Word embedding

Les représentations document-mot et mot-mot basées sur la co-occurrence occupent un grand espace mémoire ; elles sont difficiles à gérer. Dans LSA, nous avons vu que la représentation d'un mot est compactée comme un petit vecteur de nombres réels. Ceci est appelé : "Word **embedding**" ou, en français, "Plongement lexical". Ici, je vais utiliser la première nomination puisqu'elle est plus cool et plus utilisée.

Dans cette section, on va présenter Word **embedding** basé sur les réseaux de neurones. On va présenter deux types des **embeddings** :

- traditionnel : on affecte à chaque mot une seule représentation. Mais, elle ne peut pas prendre la polysémie en considération. Les algorithmes qui seront présentés sont : Word2vec et **GloVe**.
- contextuel : on affecte à chaque mot plusieurs représentations ; chacune selon son contexte. Prenons les trois phrases : "Le meilleur préservatif contre les souris est un chat", "La souris pour ordinateur est un système de pointage" et "J'adore la souris, c'est mon morceau favori". Le mot "souris" veut dire respectivement : "animal", "dispositif" et "Partie du gigot de mouton". L'idée de cette représentation est d'avoir des différents codes selon le sens et pas seulement le mot. On va présenter les algorithmes suivants : **ELMo** et **BERT**.

3.1 Word2vec

Dans le modèle Mot-Mot, nous avons vu la notion du contexte (les mots qui entourent un autre). L'idée ici est d'utiliser un modèle de langage neuronal afin d'encoder un mot en utilisant son contexte. Préalablement, les mots sont encodés en utilisant **One-Hot**. Une fois le modèle entraîné, les mots auront des codes plus compacts. Word2vec est un outil fourni par Google implémentant deux méthodes de Word **embedding** (Mikolov et al., 2013a)

- **CBOW** : Continuous Bag-of-Words
- **Skip-gram** : Continuous Skip-gram

Les mots sont encodés sur un petit vecteur (50-1000) en utilisant un encodeur-décodeur. Dans la méthode CBOW, on essaye d'estimer le mot w_i en utilisant les mots avant et après. Ceci revient à maximiser sa probabilité comme indiqué dans l'équation 6.8.

$$\max \frac{-1}{V} \sum_{i=1}^{V} p(w_i | w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}) \tag{6.8}$$

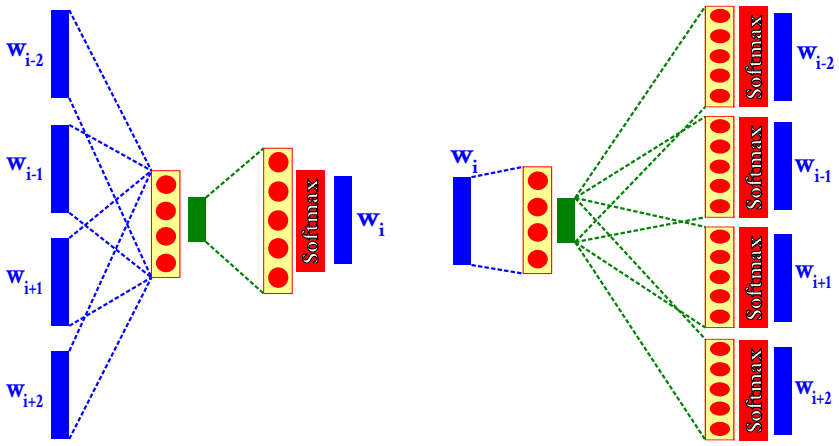
La figure 6.5(a) représente l'architecture CBOW avec un contexte de 2-2. Ici, la première couche contient L neurones avec $4 * L * |V|$ paramètres où L est la taille de l'embedding et V est l'ensemble du vocabulaire.

La couche de sortie contient $|V|$ neurones avec $L * |V|$ paramètres. En appliquant une fonction "Softmax" sur ce vecteur, on aura un vecteur de probabilités où on doit maximiser la probabilité du mot destinataire w_i (elle doit être 1) et minimiser les probabilités des autres mots du vocabulaire (elles doivent être 0).

Dans la méthode Skip-gram, on essaye d'estimer le contexte étant donné le mot w_i. Cela revient à maximiser les probabilités du contexte comme indiqué dans l'équation 6.9.

$$\max \frac{-1}{V} \sum_{i=1}^{V} \sum_{j=i-2; j\neq i}^{i+2} p(w_j|w_i) \tag{6.9}$$

La figure 6.5(b) représente l'architecture Skip-gram avec un contexte de 2-2. La première couche cachée contient L neurones et $L * |V|$ paramètres où L est la taille de l'embedding et V est l'ensemble du vocabulaire. En sortie, nous avons 4 couches en parallèle (on peut avoir plus selon la taille du contexte). Chacune contient $|V|$ neurones avec $L * |V|$ paramètres. En appliquant une fonction "Softmax" sur chaque vecteur, on aura un vecteur de probabilités où on doit maximiser la probabilité du mot destinataire (elle doit être 1) et minimiser les probabilités des autres mots du vocabulaire (elles doivent être 0).



(a) CBOW (b) Skip-Gram

Figure 6.5 : *Architecture Word2vec avec un contexte 2-2*

3.2 GloVe

GloVe (Global Vectors) est une méthode développée par Stanford (Pennington et al., 2014). Elle essaye d'exploiter les deux approches : matrice mot-mot (comme LSA) et apprentissage par contexte (comme CBOW). On prépare une matrice Mot-Mot $X[V, V]$ où X_{ij} est le nombre des occurrence du mot w_j dans le contexte de w_i, et X_i est le nombre d'occurrences de w_i dans le corpus. La probabilité d'occurrence de w_j dans le contexte de w_i est estimée comme $P_{ij} = \frac{X_{ij}}{X_i}$.

La figure 6.6 représente un exemple des probabilités conditionnelles. Si on veut trouver la relation entre deux mots (Ex. $w_i = ice$ et $w_j = steam$) par rapport à un mot w_k, on peut calculer le ratio entre leurs probabilités $R(w_i, w_j) = \frac{P_{ik}}{P_{jk}}$. Donc :

- Si $R(w_k, w_i) \wedge \neg R(w_k, w_j)$, le ratio sera grand. Ex. solid
- Si $\neg R(w_k, w_i) \wedge R(w_k, w_j)$, le ratio sera petit. Ex. gas
- Si $R(w_k, w_i) \wedge R(w_k, w_j)$, le ratio tend vers 1 . Ex. water
- Si $\neg R(w_k, w_i) \wedge \neg R(w_k, w_j)$, le ratio tend vers 1 . Ex. fashion

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 6.6 : *Exemple des probabilités conditionnelles et un ratio entre deux probabilités (Pennington et al., 2014)*

On veut entraîner une fonction F qui estime le ratio de w_i, w_j par rapport à un mot $\tilde{w}_k$ comme indiqué dans l'équation 6.10.

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \tag{6.10}$$

Il existe plusieurs fonctions F qui peuvent assurer l'équation précédente. Afin de restreindre cette fonction, on utilise la soustraction entre les deux mots w_i et w_j, comme indiqué dans l'équation 6.11.

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \tag{6.11}$$

Une autre restriction peut être appliquée en transformant les arguments de cette fonction à un scalaire. Ceci peut être fait en appliquant une multiplication matricielle, comme indiqué dans l'équation 6.12.

$$F((w_i - w_j)^\top \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \tag{6.12}$$

On doit pouvoir échanger $w \leftrightarrow \tilde{w}$ et aussi $X \leftrightarrow X^\top$. Pour garantir la symétrie, il faut tout d'abord considérer la fonction F comme étant un homomorphisme entre $(\mathbb{R}, +)$ et $(\mathbb{R}_{>0}, \times)$. Donc, elle peut être représentée par l'équation 6.13.

$$F((w_i - w_j)^\top \tilde{w}_k) = \frac{F(w_i^\top \tilde{w}_k)}{F(w_j^\top \tilde{w}_k)} \tag{6.13}$$

D'après l'équation 6.12 et l'équation 6.13, on peut déduire que $F(w_i^\top \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}$. Une solution est de considérer la fonction F comme une fonction exponentielle $F = exp$. Donc, la solution de l'égalité est présentée dans l'équation 6.14.

$$w_i^\top \tilde{w}_k = \log X_{ik} - \log X_i \tag{6.14}$$

Afin d'encoder $\tilde{w}_k$, on va entrainer un réseau de neurone afin d'estimer $\log X_{ik} - \log X_i$ comme indiqué dans la figure 6.7. Puisque la valeur $logX_i$ est indépendante de k, on peut entraîner un biais b_i sur w_i. Pour respecter la symétrie, il faut entraîner un biais $\tilde{b}_k$ sur $\tilde{w}_k$. Donc, la fonction d'estimation est représentée par l'équation 6.15.

$$w_i^\top \tilde{w}_j + b_i + \tilde{b}_j = \log X_{ij} \tag{6.15}$$

Afin d'entraîner le modèle, la fonction objective J utilisée est la méthode des moindres carrés. Le problème est que J ne doit pas pondérer les co-occurrences de la même façon : les co-occurrences rares doivent avoir moins d'impact sur J. Donc, on doit définir une fonction f qui normalise la valeur de son argument x par rapport à une valeur maximale x_{max} (voir l'équation 6.16).

$$f(x) = \begin{cases} \frac{x}{x_{max}} & \text{si } x < x_{max} \\ 1 & \text{sinon} \end{cases} \tag{6.16}$$

La fonction du coût sera calculée en utilisant l'équation 6.17.

$$J(\theta) = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(w_i^\top \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2 \tag{6.17}$$

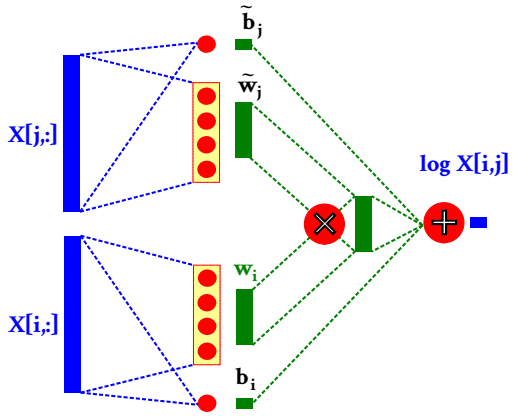


Figure 6.7 : *Architecture de la méthode GloVe*

3.3 ELMo

Comme il est déjà mentionné, les représentations Word2vec et **GloVe** ne prennent pas la polysémie en considération ; un mot est représenté par un seul vecteur quelque soit son sens. **ELMo** [4] (Embeddings from Language Models) est un modèle contextuel développé par AllenNLP (Peters et al., 2018) qui prend en considération la phrase comme contexte afin de générer la représentation d'un mot. C'est un modèle bi-directionnel ; qui prend en considération tous les mots avant et après. En plus, il est basé sur les caractères et pas les mots. Donc, il a la capacité de prendre en considération des caractéristiques morphologiques et des mots hors vocabulaire.

La figure 6.8 représente l'architecture du modèle ELMo. Étant donné un mot w_k, on calcule sa représentation x_k en se basant sur les caractères qui lui composent (Kim et al., 2015). On utilise L couches des cellules **Bi-LSTM** (LSTMs bidirectionnelles). Pour chaque phrase de N mot, on essaye de maximiser la somme des logarithmes des probabilités en avant et en arrière, comme indiqué dans l'équation 6.18. Les paramètres à entraîner sont : Θ_x (embedding des caractères), $\overrightarrow{\Theta}_{LSTM}$ (les LSTMs en avant), $\overleftarrow{\Theta}_{LSTM}$ (les LSTMs en arrière) et Θ_s (combinaison entre les deux directions : en avant et en arrière).

$$\sum_{k=1}^{N} \log P(w_k|w_1,\ldots,w_{k-1};\Theta_x,\overrightarrow{\Theta}_{LSTM},\Theta_s) + \log P(w_k|w_{k+1},\ldots,w_N;\Theta_x,\overleftarrow{\Theta}_{LSTM},\Theta_s) \tag{6.18}$$

A la fin d'entraînement, la représentation contextuelle d'un mot w_k sera la combinaison entre sa représentation en utilisant le modèle de langage basé caractères x_k^{LM} et les vecteurs des couches cachées des deux réseaux **LSTM** (en avant et en arrière) :

$$R_k = \{x_k^{LM}, \overrightarrow{h}_{LM}^{k,j}, \overleftarrow{h}_{LM}^{k,j} | j = 1\ldots L\} = \{h_{LM}^{k,j} | j = 0\ldots L\}$$

Afin d'intégrer **ELMo** avec une tâche *task*, on entraîne des paramètres Θ^{task} afin d'inférer une représentation unique liée à la tâche. Cette représentation est estimée selon l'équation 6.19.

$$ELMo_k^{task} = E(R_k;\Theta^{task}) = \gamma^{task}\sum_{j=0}^{L}\theta_j^{task}h_{LM}^{k,j} \tag{6.19}$$

4. ELMo : http://allennlp.org/elmo [visité le 2021-09-11]

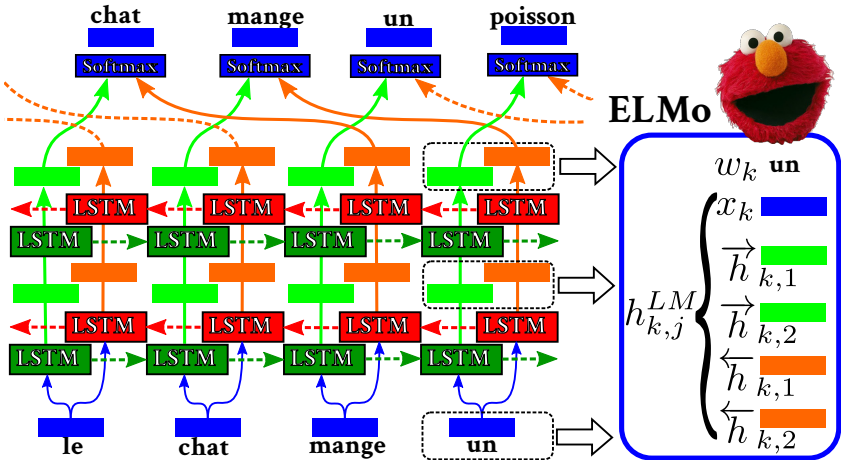


Figure 6.8 : *Architecture du modèle ELMo*

3.4 BERT

BERT [5] (Bidirectional Encoder Representations from Transformers) est un autre **embedding** contextuel développé par Google (Devlin et al., 2019). Comme **ELMo**, **BERT** prend aussi la totalité de la phrase comme contexte. Il est bidirectionnel ; il prend en considération les mots avant et après. Contrairement à **ELMo** qui se base sur les **LSTMs** bidirectionnels, **BERT** se base sur un modèle **transformer** (Vaswani et al., 2017). Ceci permet à un mot w_i d'avoir une vision globale sur les mots de la phrase et pas une vision temporelle (chaine de mots ordonnés). Cette représentation est basée sur les tokens ; les mots sont séparés en radical + affixes.

La figure 6.9 représente l'architecture du modèle **BERT**. Le texte est séparé en tokens en utilisant "Word-piece" (Wu et al., 2016). L'entrée a un maximum de $T = 512$ tokens. Le premier token est un marqueur spécial "[CLS]" utilisé pour la classification. En cas de deux phrases en entrée, on utilise un token "[SEP]" pour les séparer. Chaque token est représenté par 3 **embeddings** de taille N

- **Embedding de token** : transformation d'un vocabulaire de taille V vers un vecteur de taille N ;
- **Embedding de position** : transformation de la position du token dans la phrase sur une taille max T vers un vecteur de taille N ;
- **Embedding de segment** : transformation du segment du token (phrase1 ou phrase2) encodé avec une taille de 2 vers un vecteur de taille N.

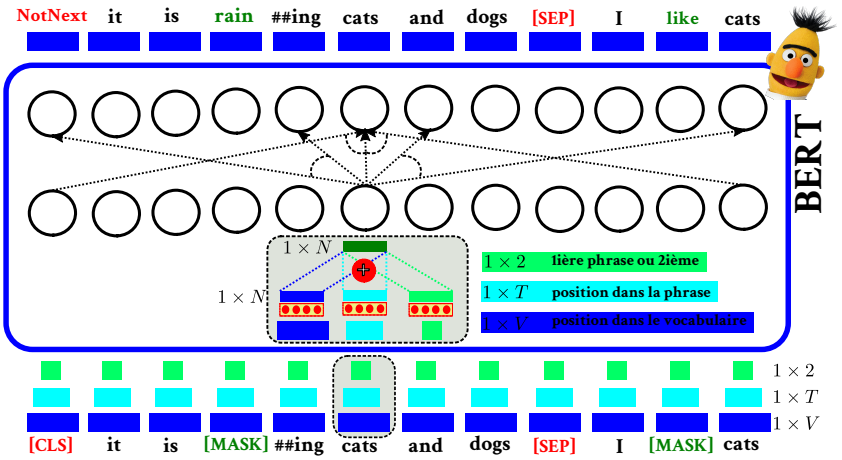


Figure 6.9 : *Architecture du modèle BERT*

5. BERT : https://github.com/google-research/bert [visité le 2021-09-11]

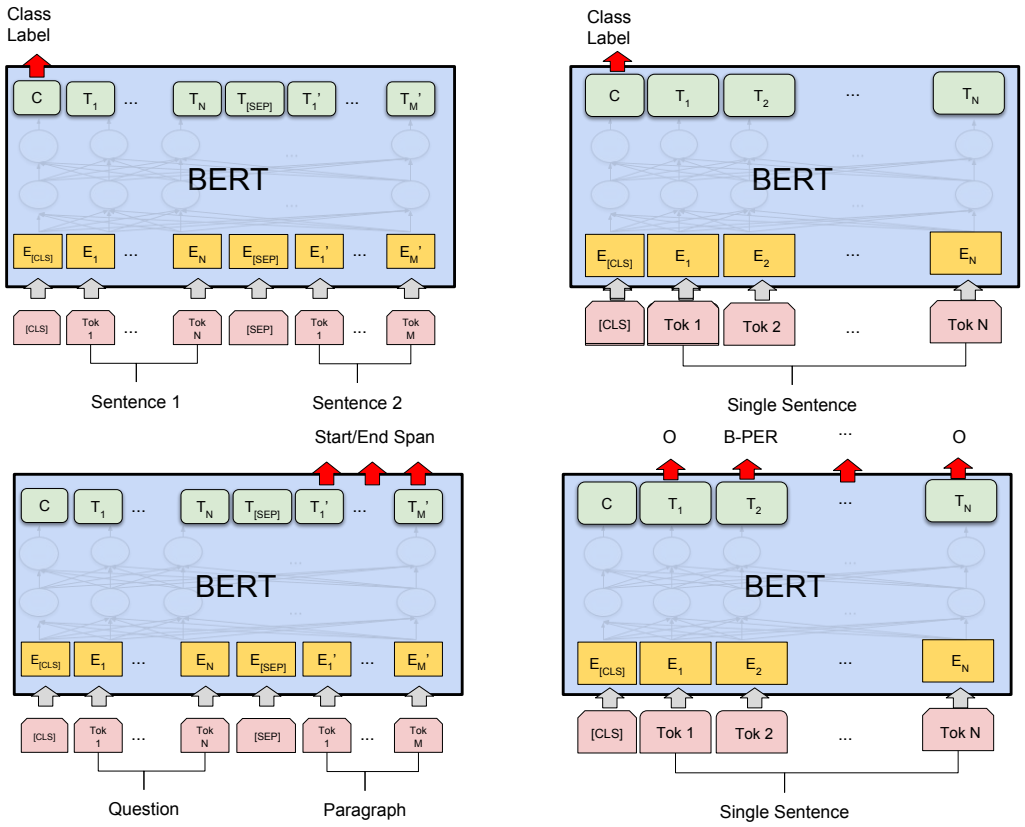


Figure 6.10 : *Réglage de BERT sur des différentes tâches (Devlin et al., 2019)*

Le modèle **BERT** utilise l'apprentissage par transfert ; on entraîne le modèle avec une tâche (pré-entraînement), ensuite on l'entraîne avec une tâche similaire (réglage). Dans le cas de **BERT**, on utilise deux tâches afin d'avoir un modèle pré-entraîné :

- **Modèle de langage masqué** : on masque aléatoirement 15% des tokens d'une phrase et on essaye de les inférer. Pour le faire, on utilise un token spécial : "[MASK]". Puisque ce token n'apparait pas dans l'étape de réglage, on l'utilise pour 80% des remplacements. Parmi ces masques, on utilise 10% avec un token quelconque et 10% sans changement.
- **Prédiction de la phrase suivante** : prédire si la deuxième phrase suit la première. Le résultat est dans la sortie du token "[CLS]" ($CLS \in \{IsNext, NotNext\}$) ; c'est un classement binaire.

Puisque **BERT** se base sur le concept de l'apprentissage par transfert, on peut régler le modèle sur d'autres tâches. Dans ce cas, la représentation vectorielle est le modèle neuronale lui-même. La figure 6.10 représente quelques tâches accomplies avec **BERT**. En haut à gauche, c'est une tâche qui classifie la relation entre deux phrases ; comme la similarité (similaire/non similaire). En haut à droit, c'est une tâche de classification d'une phrase ; comme l'analyse de sentiments. En bas à gauche, c'est la tâche de question/-réponse où l'entrée est une question et un paragraphe et la sortie est une partie du paragraphe contenant la réponse. En bas à droit, c'est une tâche d'annotation (tagging) ; comme la reconnaissance des entités nommées.

3.5 Évaluation des modèles

Il existe deux approches pour évaluer un modèle : intrinsèque et extrinsèque. Cette dernière a comme but de comparer deux modèles en terme d'une tâche donnée. Par exemple, **GloVe** surpasse LSA dans la tâche de reconnaissance des entités nommées selon Pennington et al. (2014). Concernant l'évaluation intrinsèque, plusieurs méthodes/corpus ont été proposés :

— WordSimilarity-353 Test Collection (Finkelstein et al., 2002) : dans cette collection, les similarités entre les mots ont été annotées manuellement (un nombre entre 0 et 10). Afin de tester un modèle, on calcule la corrélation de Spearman entre les similarités basées sur les représentations (similarité cosinus) et les similarités manuelles.
— SimLex-999 (Hill et al., 2015) : ici, on teste la similarité (coast, shore) et pas l'association (clothes, closet). La similarité entre deux mots est un nombre annoté manuellement.
— Analogies de mots (Mikolov et al., 2013b) : c'est un dataset de la forme (w_{i1} : w_{j1} :: w_{i2} : w_{j2}). Il a comme objectif de tester la capacité des embeddings à représenter des relations d'analogie $w_{j2} = w_{i1} - w_{i2} + w_{j1}$. Par exemple, (King :Queen :: Man :Woman) →King - Man + Woman = Queen.

En plus, il faut évaluer le biais dans le modèle. En se basant sur le corpus d'entraînement, un modèle peut apprendre des analogies biaisées. Par exemple, il peut apprendre des stéréotypes comme "she" avec "homemaker", "nurse", "receptionist" et "he" avec "maestro", "skipper", "protege" (Caliskan et al., 2017). Ceci peut affecter la performance de certaines tâches. Par exemple, la résolution des anaphores peut échouer à lier le pronoms "she" avec "doctor".

4 Désambigüisation lexicale

La désambigüisation lexicale, en anglais Word Sense Disambiguation (WSD), est la tâche concernée par la recherche du sens correct d'un mot dans une phrase. Elle est utile pour plusieurs tâches :
— Analyse syntaxique : "I fish in the river" (Verbe ou nom ?). "The fish was too big" (Verbe ou nom ?).
— Traduction automatique : "I withdrawed money from the bank" ("banque" ou "rive" ?). "I fish on the bank" ("banque" ou "rive" ?).

4.1 Basée sur des bases de connaissance

La méthode la plus basique pour appliquer WSD est l'algorithme de Lesk fourni par **WordNet** (voir l'algorithme 6.1). On commence par la récupération de tous les lexèmes du mot w appartenant à la phrase s. Au début, on considère que le sens du mot w est celui le plus fréquent (la fréquence est calculée à partir d'un corpus). Pour chaque sens du mot, on calcule le nombre des mots en commun entre son glossaire/exemples et les mots de la phrase s. Le sens avec un nombre maximum est le sens voulu.

Les bases de données lexicales peuvent être représentées comme des graphes. Une des méthodes qui utilisent la structure de graphe pour la tâche WSD est Babelfy [6] (Moro et al., 2014). Cette méthode est exécutée sur trois étapes :

1. **Construction des signatures sémantiques** : on commence par construire un graphe en utilisant tous les concepts à partir d'un réseau sémantique. Pour chaque arc reliant deux concepts, on attribut un poids en se basant sur le nombre des triangles qui les relient. Ensuite, on calcule la probabilité d'un concept sachant un autre en se basant sur ces poids. Enfin, on minimise le graphe en utilisant la méthode "Random walk with restart".
2. **Identification des candidats** : on applique l'étiquetage morphosyntaxique sur le texte d'entrée. Ensuite, on extrait tous les sens possibles des mots ou des expressions de la phrase d'entrée.
3. **Désambigüisation des candidats** : on construit un graphe en utilisant la signature sémantique et les candidats. On cherche un sous-graphe en éliminant les liens faibles.

6. Babelfy : http://babelfy.org/ [visité le 2021-09-11]

Algorithme 6.1 : *Algorithme de Lesk*

Données : un mot w ; une phrase s contenant w
Résultat : Le sens de w
meilleur_sens ←plus fréquent parmi les sens de w ;
superposition_max ←0 ;
contexte ←l'ensemble des mots de s ;
pour tous les *sens* w_i *de* w **faire**
    signature ←l'ensemble des mots dans le **gloss** et exemples du sens w_i ;
    superposition ←nombre des mots en commun entre **contexte** et **signature** ;
    si *superposition* > *superposition_max* **alors**
        superposition_max ←superposition ;
        meilleur_sens ←w_i ;
    fin
fin
retourner meilleur_sens ;

4.2 Basée sur l'apprentissage automatique

La désambigüisation des mots peut être vue comme une tâche d'étiquetage des séquences. Donc, on peut appliquer les **HMM** ou les réseaux de neurones récurrents afin de la résoudre. Pour ce faire, on utilise un corpus annoté (Ex. SemCor) où chaque mot est suivi par le numéro de son sens dans une base lexicale (**WordNet**). Par exemple, "You will find9 that avocado1 is1 unlike1 other1 fruit1 you have ever1 tasted2". En entrée, on peut utiliser les mêmes caractéristiques utilisées dans l'étiquetage des séquences comme les mots précédents, leurs classes, le mot courant, etc. En sortie, on aura un vecteur **One-Hot** représentant la classe (le sens est un nombre).

Une autre méthode pour la WSD est d'utiliser les **embeddings** contextuels. La figure 6.11 représente la désambigüisation des mots d'une phrase en utilisant un modèle d'un **embedding** contextuel comme **ELMo** ou **BERT**. Étant donné un modèle pré-entrainé, on le règle sur un corpus annoté afin de capturer les **embeddings** de chaque sens. Pour avoir le **embedding** d'un sens v_s, on calcule la moyenne des **embeddings** c_i qui l'appartiennent (voir l'équation 6.20).

$$v_s = \frac{1}{n}\sum_{i=1}^{n} c_i \tag{6.20}$$

Lors du test, on passe la phrase par le modèle. Pour chaque mot w, on cherche les **embeddings** de tous les sens et on prend le plus proche en utilisant la similarité cosinus.

Discussion

Imaginer le mot "café". Quelle est l'image que vous avez visualisée ? Peut-être, vous avez imaginé un vers plein du café chaud. Maintenant, imaginer le mot café en le liant avec la phrase "je vais au café". Ici, l'image va changer à un immeuble avec des tables et des chaises où on boit du café. Dans les deux cas, votre cerveau a affecté une image au mot. On peut constater immédiatement que l'encodage des concepts est une étape primordiale dans le traitement des idées. Dans chaque langue, les idées sont représentées par des phrases et les concepts qui les composent sont représentés par des mots.

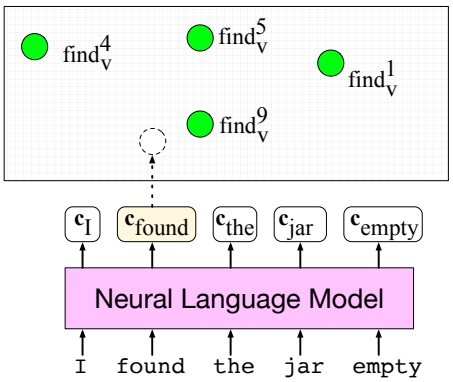


Figure 6.11 : *Exemple de désambigüisation avec le plus proche voisin (Jurafsky et Martin, 2019)*

Un mot peut être représenté par un identifiant dans une base de donnée. On peut représenter les relations sémantiques du mot avec les autres sous forme d'un graphe. Une autre représentation est sous forme d'un vecteur où la plus basique est l'encodage One-Hot. Il existe plusieurs méthodes de représentation vectorielle des plus simples comme TF-IDF jusqu'aux plus avancées comme les embeddings contextuelles. L'utilisation d'un modèle contextuelle ne veut pas dire que la tâche va s'améliorer. Des fois, on peut trouver des tâches qui sont plus performantes avec des modèles plus simples. D'où la nécessité d'appliquer une évaluation extrinsèque afin de choisir le modèle le plus adéquat.

CHAPITRE 7

SÉMANTIQUE DE LA PHRASE

LES LANGUES

Généralement, une phrase se compose d'une action et des composants auteur d'elle qui jouent des rôles thématiques. Dans l'analyse syntaxique, nous avons vu que la phrase peut être décomposée en : sujet, verbe et objet. En français, la plupart du temps, le sujet est celui qui a fait l'action; mais pas toujours. Afin de représenter une phrase sémantiquement, on doit trouver les rôles de chaque composant. La représentation peut être logique, graphique, vectorielle, etc. On peut avoir la représentation vectorielle d'une phrase en calculant le centre des embeddings des mots qui lui composent. Les deux autres approches vont être présentées en détail dans ce chapitre.

Trouver les rôles sémantiques des composants d'une phrase est le premier pas vers sa représentation sémantique. Une bonne représentation est celle qui n'a aucune relation avec une langue précise. Une représentation sémantique d'une phrase a plusieurs applications :

- Compréhension du langage naturel
- Questions-Réponses
- Recherche d'information
- Traduction automatique
- Résumé automatique

1 Rôles sémantiques

Chaque groupe nominal joue un rôle sémantique dans un évènement de la phrase. Afin de comprendre une phrase ou représenter son sens, on doit détecter ces rôles. Prenons les phrases suivantes :

1. Mon chat a attrapé une souris avec ses griffes
2. Mon chat a attrapé une souris avec sa queue
3. Mon chat a attrapé une souris avec un autre chat

Dans les trois phrases, "Mon chat" est celui qui a fait l'évènement (agent) et "une souris" c'est celui qui l'a subi (thème). Malgré que les compléments d'objet indirects des trois phrases se commencent par la préposition "avec", les trois syntagmes nominaux qui la suivent ont des rôles sémantiques différents. Le syntagme "ses griffes" représente l'instrument, "sa queue" représente la moyenne et "un autre chat" représente un autre agent. Ici, on va présenter quelques rôles sémantiques et deux ressources pour les représenter.

1.1 Rôles thématiques

Le rôle thématique (sémantique) décrit le sens d'un groupe nominal par rapport un évènement exprimé par un verbe de la phrase. Le tableau 7.1 représente quelques rôles thématiques avec leurs descriptions et

exemples.

Rôle	Description	Exemple
AGENT	Le causeur volontaire d'un évènement	John a cassé la fenêtre avec une pierre.
EXPERIENCER	L'expérimentateur d'un évènement	John a mal à la tête.
FORCE	Le causeur non volontaire d'un évènement	Le vent souffle les débris.
THEME	Le participent affecté directement par l'évènement	John a cassé la fenêtre avec une pierre.
RESULT	Le produit final d'un évènement	La ville a construit un terrain de baseball.
CONTENT	Une proposition ou e contenu d'un évènement propositionnel	Mona a demandé "Vous avez rencontré Mary Ann dans un supermarché?"
INSTRUMENT	Un instrument utilisé dans l'évènement	une pierre a cassé la fenêtre.
BENEFICIARY	Le bénéficiaire d'un évènement	Ann fait des réservations d'hôtel pour son patron.
SOURCE	L'origine de l'objet d'un évènement de transfert	Je suis arrivé de Boston.
GOAL	La destination de l'objet d'un évènement de transfert	Je suis allé à Portland.

Tableau 7.1 : *Quelques rôles thématiques (Jurafsky et Martin, 2019)*

1.2 FrameNet

FrameNet [1] est un projet qui vise à annoter les rôles sémantiques en se basant sur la théorie "**Frame semantics**" (Sémantique des cadres) de **Fillmore**. **NLTK** [2] fournit un API pour utiliser **FrameNet**. Un cadre est une représentation schématique d'une situation avec des participants ayants des rôles sémantiques. Il doit pouvoir détecter la reformulation d'une phrase avec le même sens. Par exemple, les phrases suivantes ont le même cadre :

- The price of petrol increased.
- The price of petrol rose.
- There has been a rise in the price of petrol.

FrameNet se compose d'un ensemble des cadres sémantiques préparés manuellement. Un cadre (Frame) est une représentation schématique d'une situation. Chaque cadre se compose d'un nom, une définition, des éléments de cadre, des relations avec d'autres cadres et des unités lexicales. Un élément d'un cadre (Frame Element : FE) est un rôle sémantique spécifique au cadre qui décrit un participant ou une situation dans le cadre. Il est composé d'un rôle sémantique, un type sémantique, une définition et un exemple. Il existe deux types de rôles : de base (Core) qui sont essentiels, et d'autres secondaires (Non-Core). Les relations avec les autres cadres sont représentées comme un tuple (type-relation, cadre). Parmi ces relations, on peut mentionner : l'héritage, l'utilisation, la causalité, etc. Les unités lexicales (Lexical Units) sont représentées par un ensemble des lemmes avec leurs catégories grammaticales. Une unité lexicale déclenche le cadre lorsqu'elle est rencontrée.

1. FrameNet : https://framenet.icsi.berkeley.edu/fndrupal/ [visité le 2021-09-11]
2. NLTK Framework : https://www.nltk.org/howto/framenet.html [visité 2021-09-11]

Le tableau 7.2 représente le cadre sémantique appelé “Cause_to_fragment” (faire fragmenter). Les éléments essentiels d’un cadre sont : l’agent qui est un être sensible, une cause, un patient complet et des pièces. La définition décrit comment ces éléments interagissent. Il existe des éléments secondaires comme l’instrument utilisé dans la fragmentation. Concernant les relations avec les autres cadres, on peut citer “Is Causative of : Breaking_apart”. Donc, le fait de fragmenter une chose est la cause de se briser. Ce cadre peut être activé par plusieurs mots comme les verbes : break apart, dissect, smash, etc.

Les unités lexicales sont utilisées pour déclencher des cadres. Une unité lexicale est un tuple (lemme, catégorie lexicale) qui représente un sens d’un mot donné. Le sens est lié à un cadre sémantique. Le tableau 7.3 représente quelques unités lexicales du mot “break”. Parmi les cadres activés par ce mot, on trouve le cadre “Cause_to_fragment” présenté dans le tableau 7.2. En général, les verbes sont les déclencheurs les plus utilisés dans **FrameNet**.

FrameNet fournit un ensemble des entrées lexicales (Lexical Entry). Une entrée lexicale représente la structure syntaxique d’un cadre par rapport une unité lexicale. Elle contient un tableau qui lie chaque élément de cadre avec l’ensemble de ces réalisations syntaxiques. Par exemple, l’élément “Whole_patient” est lié avec “NP.Obj” (un syntagme nominal qui est un objet) dans 29 exemples. **FrameNet** fournit un autre tableau qui représente la liste de patrons de valence. Chaque ligne représente l’ordre des éléments de cadre par rapport à la structure syntaxique. Un exemple de la liste de patrons de valence du verbe “fracture” du cadre “Cause_to_fragment” est donné dans le tableau 7.4. Chaque patron est accompagné par le nombre des exemples annotés.

FrameNet fournit un ensemble de phrases annotées pour chaque entrée lexicale. La figure 7.1 représente un extrait des annotations lexicographiques du verbe “fracture” et le cadre “Cause_to_fragment”. Ici, les éléments de cadres absents dans l’annotation sont marqués par “[INI]” (indefinite null instantiation). Le corpus annoté est généralement utilisé pour entraîner ou tester un système d’étiquetage des rôles sémantiques.

- 429-s20-rcoll-skull
  1. [Agent Former England Under-21 player Keith Benton] FRACTURED[Target] [Whole_patient his son Seb ’s skull] [Time when he hit the ball into the crowd during a match in Buckingham]. [Pieces INI]
  2. [Agent He] hit a lamp-post and FRACTURED[Target] [Whole_patient Mike ’s skull]. [Pieces INI]
  3. When he found the man [Agent he] threw the acid into his face and beat him with the hammer , FRACTURING[Target] [Whole_patient his skull] and his thumb. [Pieces INI]
  4. [Agent A nanny] has been jailed after FRACTURING[Target] [Whole_patient the skulls of two new born babies in her care]. [Pieces INI]

- 520-s20-np-vping
- 620-s20-np-ppother
- 660-s20-trans-simple
  1. [Agent Then 17-year-old Lee Diaz, of North End Gardens, Bishop Auckland], attacked a second party-goer, Carl Gent, punching him in the face and FRACTURING[Target] [Whole_patient his jaw]. [Pieces INI]

- 680-s20-pass
  1. [Whole_patient It] was FRACTURED[Target] [Instrument with a solvent-cleaned chisel], and the outer orange layer discarded. [Agent CNI][Pieces INI]

Figure 7.1 : *Extrait des annotations lexicographiques du déclencheur “fracture” (verbe) du cadre “Cause_to_fragment”*

Cause_to_fragment	
Définition	An Agent suddenly and often violently separates the Whole_patient into two or more smaller Pieces, resulting in the Whole_patient no longer existing as such. Several lexical items are marked with the semantic type Negative, which indicates that the fragmentation is necessarily judged as injurious to the original Whole_patient. Compare this frame with Damaging, Render_non-functional, and Removing.
FEs (Core)	
Agent [Agt] Semantic Type : Sentient	The conscious entity, generally a person, that performs the intentional action that results in the Whole_patient being broken into Pieces. I and I alone can SHATTER the gem and break the curse.
Cause [cau]	An event which leads to the fragmentation of the Whole_patient.
Pieces [Pieces]	The fragments of the Whole_patient that result from the Agent's action. I SMASHED the toy boat to flinders.
Whole_patient [Pat]	The entity which is destroyed by the Agent and that ends up broken into Pieces. Shattering someone's confidence is a little different than SHATTERING a dish.
FEs (None-Core)	
Degree [Degr] Semantic Type : Degree	The degree to which the fracturing is completed. I SHATTERED the vase completely.
Explanation [Exp] Semantic Type : State_of_affairs	A state of affairs that the Agent is responding to in performing the action. He TORE the treaty UP out of frustration.
Explanation [Exp] Semantic Type : State_of_affairs	A state of affairs that the Agent is responding to in performing the action. He TORE the treaty UP out of frustration.
Instrument [Ins] Semantic Type : Physical_entity	An entity directed by the Agent that interacts with a Whole_patient to accomplish its fracture.
...	
Frame-frame Relations	
Inherits from	Transitive_action
Uses	Destroying
Is Causative of	Breaking_apart
Lexical Units	
	break apart.v, break down.v, break up.v, break.v, chip.v, cleave.v, dissect.v, dissolve.v, fracture.v, fragment.v, rend.v, rip up.v, rip.v, rive.v, shatter.v, shiver.v, shred.v, sliver.v, smash.v, snap.v, splinter.v, split.v, take apart.v, tear up.v, tear.v

Tableau 7.2 : *Exemple d'une partie du cadre sémantique "Cause_to_fragment", https://framenet2.icsi.berkeley.edu/fnReports/data/frameIndex.xml?frame=Cause_to_fragment [visité le 2021-09-11]*

Lexical Unit	Frame	Exemple
break.n	Opportunity	
break.v	Cause_harm	Jolosa broke a rival player's jaw.
break.v	Compliance	He broke his promess.
break.v	Experience_bodily_harm	I broke my arm in the accident.
break.v	Cause_to_fragment	Michael broke the bottle against his head
break.v	Render_nonfunctional	I guess I broke the doorknob by twisting it too hard.
break.v	Breaking_off	The handle broke off of the pot.
break.v	Breaking_apart	The handle broke off of the pot.

Tableau 7.3 : *Quelques unités lexicales du mot "break"*

Number Annotated	Patterns				
1 TOTAL	Agent	Instrument	Pieces	Whole_patient	
(1)	CNI - -	PP[with] Dep	INI - -	NP Ext	
1 TOTAL	Agent	Means	Pieces	Time	Whole_patient
(1)	NP Ext	2nd - -	INI - -	Sinterrog Dep	NP Obj
4 TOTAL	Agent	Pieces	Whole_patient		
(4)	NP Ext	INI - -	NP Obj		

Tableau 7.4 : *Entrée lexicale du déclencheur "fracture" (verbe) du cadre "Cause_to_fragment" : extrait de liste de patrons de valence*

1.3 PropBank

PropBank [3] (Propositional Bank) est un corpus des phrases annotées en se basant sur la structure Prédicat-Arguments. **NLTK** [4] fournit un API pour accéder à **PropBank**. L'annotation se base sur moins de rôles sémantiques : agent et patient. L'agent participe volontairement dans un évènement ou un état. Il peut aussi causer un évènement ou un changement d'état d'un autre participant. Le patient est le participant qui éprouve un changement d'état. Il peut aussi être affectée par un autre participant.

PropBank est structuré comme un ensemble des fichiers de prédicats (verbes) ; en général sous formes de fichiers XML. Chaque prédicat définit plusieurs ensembles de rôles (rolesets) qui représentent les différents sens. Chaque roleset est structuré comme suit :

— Un identifiant numérique et un ensemble des verbes ayant le même sens. Par exemple, know.01 : be cognizant of, realize ; know.02 : be familiar with, have experienced.

— Des rôles : un verbe dans un roleset a plusieurs arguments annotés par le mot clé **Arg** suivi par un numéro entre 0 et 5. Arg0 et Arg1 sont toujours réservés pour le PROTO-AGENT et le PROTO-

3. PropBank : https://propbank.github.io/ [visité le 2021-09-11]
4. NLTK PropBank : https://www.nltk.org/howto/propbank.html [visité le 2021-09-11]

PATIENT respectivement. Le reste des arguments ne sont pas consistants dans le corpus. En général, Arg2 veut dire : benefactive, instrument, attribute, ou end state ; Arg3 veut dire : start point, benefactive, instrument, ou attribute ; et Arg4 veut dire : end point. Des modificateurs (Modifiers) peuvent être fournis par un roleset ; ils sont marqués par le mot clé **ArgM**. Par exemple, ArgM-TMP : Quand ? ArgM-LOC : Où ? ArgM-MNR : Comment ?

— Des exemples annotés

La figure 7.2 représente un extrait du premier sens du prédicat “know” dans **PropBank**.

— **Roleset id**
  — **know.01** : be cognizant of, realize

— **Roles**
  — **Arg0** : knower
  — **Arg1** : fact that is known
  — **Arg2** : entity that arg1 is known ABOUT

— **Example : know-v : sentential thing known**
  — [$_{Arg0}$ The other side] knows [$_{Arg1}$ that Giuliani has always been prochoice].

— **Example : know-v : attributive**
  — [$_{Arg0}$ He] did[$_{ArgM-NEG}$ n’t] know [$_{Arg1}$ (anything)] [$_{Arg2}$ about most of the cases] [$_{ArgM-TMP}$ until Wednesday].

Figure 7.2 : *Extrait des annotations ProBank du prédicat “know”, http://verbs.colorado.edu/propbank/framesets-english-aliases/know.html [visité le 2021-09-11]*

2 Étiquetage de rôles sémantiques

L’étiquetage des rôles sémantiques, en anglais Semantic Role Labeling (SRL), est la tâche d’attribution des rôles sémantiques aux mots ou aux expressions dans une phrase. La figure 7.3 représente une phrase avec l’étiquetage des rôles sémantiques basé sur **PropBank**. C’est une tâche d’étiquetage des séquences. Elle peut être implémentée soit en utilisant des caractéristiques ou en utilisant les réseaux de neurones.

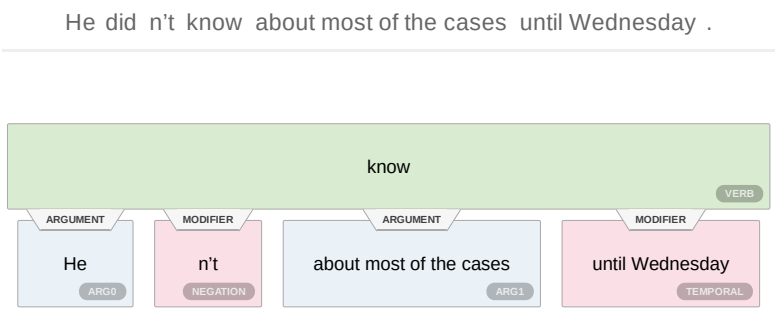


Figure 7.3 : *Exemple d’étiquetage de rôles sémantiques en se basant sur PropBank, https://demo.allennlp.org/semantic-role-labeling/ [visité le 2021-09-11]*

2.1 En utilisant des caractéristiques

On commence par analyser la phrase syntaxiquement pour avoir son arbre syntaxique. Ensuite, on parcourt chaque nœud de ce dernier pour décider la classe en se basant sur certaines caractéristiques. Les classes peuvent être celles de **FrameNet** ou **PropBank** en plus de “None” pour marquer un nœud sans rôle. Un classificateur est entraîné afin de classer un nœud en utilisant des caractéristiques comme :

— Le prédicat principal de la phrase. En général, c'est le verbe du **syntagme** verbal (VP : verbal phrase) descendant directe de la racine.
— Le type du **syntagme** (Ex. NP, S, PP).
— Le mot d'entête (principal) du **syntagme**. Le mot principal d'un **syntagme** nominal est un nom, celui d'un **syntagme** prépositionnel est une préposition, etc. Celui-ci peut être détecté en se basant sur la grammaire du langage.
— La catégorie grammaticale du mot principal.
— Le chemin du nœud concerné vers le prédicat. Exemple, NP↑S↓VP ↓VPD. La figure 7.4 représente un arbre syntaxique avec le chemin d'un syntagme vers le prédicat principal.
— La voie : active ou passive.
— La position par rapport au prédicat : avant ou après.
— ...

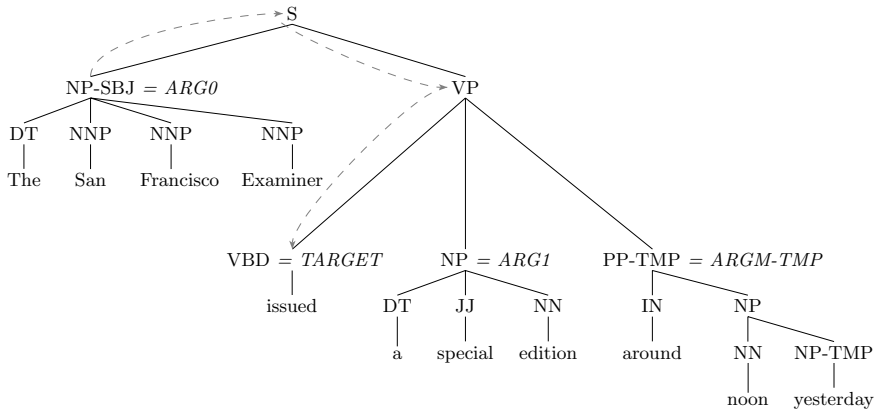


Figure 7.4 : *Exemple d'un arbre syntaxique avec le chemin d'un syntagme vers le prédicat principal (Jurafsky et Martin, 2019)*

Quelques optimisations peuvent être appliquées afin d'améliorer la tâche d'annotation. Les feuilles de l'arbre représentent les catégories grammaticales. Donc, on ne les classe pas puisque seuls les syntagmes peuvent être des arguments. On peut, aussi, entraîner un classificateur pour classer le nœud comme "Argument" ou "None". Ensuite, entraîner un autre classificateur seulement avec les nœuds avec "Argument".

2.2 En utilisant les réseaux de neurones

Nous avons vu que l'étiquetage des séquences peut être implémenté en utilisant la méthode **IOB** (inside, outside, begins). La figure 7.5 représente un système d'étiquetage de rôles sémantiques en utilisant les **LSTMs** proposé par He et al. (2017). Dans un moment donné, l'entrée est le **embedding** du mot courant fusionné avec un indicateur de prédicat (prédicat ou non). La sortie est un vecteur des probabilités des classes **PropBank**.

3 Représentation sémantique des phrases

Une représentation sémantique d'une phrase ne doit pas être dépendante à la structure syntaxique. Elle doit pouvoir exprimer les phrases ayant le même sens de la même manière. Par exemple, les deux phrases suivantes doivent avoir la même représentation :

— L'étudiant a préparé un rapport.
— Un rapport a été préparé par l'étudiant.

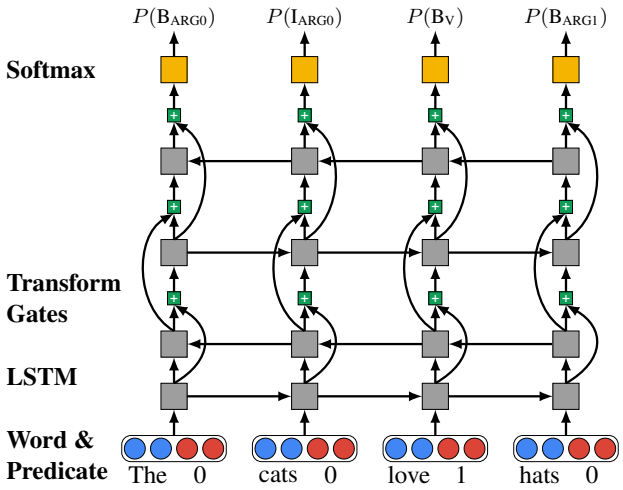


Figure 7.5 : *Système d'étiquetage de rôles sémantiques avec LSTM (He et al., 2017)*

Nous avons vu qu'une phrase peut être représentée par un ensemble de cadres en utilisant **FrameNet**. En plus de cette représentation, on peut représenter le sens en utilisant la logique du premier ordre ou en utilisant les graphes. Parmi les difficultés rencontrées dans la tâche d'analyse sémantique, on peut citer l'ambigüité. Par exemple, la phrase "Elle a emporté les clefs de la maison au garage." veut dire que la maison est une source de l'action "emporter" ou un dépendant des clefs.

3.1 Logique du premier ordre

La logique du premier ordre, First Order Logic (FOL), peut être utilisée pour représenter le sens des phrases. C'est une représentation indépendante de la structure syntaxique du langage. Une expression en FOL peut être vérifiée et inférée facilement. Elle se compose des termes, des prédicats, des connecteurs et des qualificateurs.

Un terme représente un objet qui peut être : une constante, une fonction ou une variable. Une **constante** représente un objet spécifique dans le modèle. Par exemple, les mots Karim, ESI, Algérie qui sont des entités nommées : nom propre, organisation et pays respectivement. Bien sûr, une constante peut référer un nom abstrait comme par exemple "Informatique". Des fois, on réfère un objet, pas par son nom mais, par sa relation avec une constante. Par exemple, l'expression "emplacement de l'ESI" réfère à une place. Mais en utilisant seulement les constantes, on ne peut pas la décrire en utilisant FOL. Une **fonction** peut être utilisée afin de retourner un objet en fonction d'un autre. Dans ce cas, l'expression précédente peut être représentée comme "EmplacementDe(ESI)". Prenons maintenant l'expression "un étudiant" qui réfère un objet non spécifique ; on ne sait pas qui est cet étudiant. Les deux types de termes (constante et fonction) ne peuvent pas représenter cette information. Une **variable** peut être utilisée comme référence vers un objet inconnu (anonyme). Chaque variable est représentée comme une lettre en minuscule (Ex. x, y, z).

Un prédicat représente une relation logique entre plusieurs termes qui retourne soit vrai ou faux. Prenons l'exemple "ESI enseigne l'informatique". Cette phrase peut être représentée comme :

$$enseigner(ESI, INFORMATIQUE)$$

Dans cet exemple, le prédicat est un verbe transitif. Maintenant, prenons l'exemple "ESI est une école". Cette phrase peut être représentée en utilisant un prédicat unitaire :

$$ecole(ESI)$$

Ici, le prédicat n'est pas utilisé pour décrire une relation entre plusieurs termes. Il est utilisé pour représenter une propriété de la constante "ESI". On doit être capable de différencier entre un prédicat unitaire et une fonction. Un prédicat retourne une valeur logique, or une fonction retourne un nouveau objet. Prenons un exemple avec les deux : "Karim connait l'adresse de l'ESI". Cette phrase peut être représentée comme :

$$connaitre(KARIM, AdresseDe(ESI))$$

Les prédicats, étant des relations logiques, doivent être liés pour avoir une expression plus complexe. Par exemple, la phrase "Karim est un enseignant à l'ESI qui est une école." peut être représentée comme :

$$enseignant(KARIM) \wedge location(ESI) \wedge ecole(ESI)$$

Ici, on a utilisé un prédicat *location* pour représenter l'emplacement. Aussi, on a employé le connecteur $\wedge$ afin de lier les prédicats. La liste des connecteurs possibles est la suivante : ET ($\wedge$) OU ($\vee$), NON ($\neg$), IMPLIQUE ($\rightarrow$) et EQUIVALENT ($\Leftrightarrow$).

Les variables sont utilisées afin de référer des objets anonymes. Mais, il n'ont pas la capacité à décrire qu'il s'agit d'un objet ou tous les objets d'une collection. Cela est possible en utilisant les quantificateurs : IL-EXISTE ($\exists$) et POUR-TOUS ($\forall$). Prenons l'exemple "Je mange à un restaurant près de l'ESI.". Ceci peut être représenté comme suit :

$$\exists x\, Restaurant(x) \wedge PresDe(EmplacementDe(x), EmplacementDe(ESI)) \wedge MangerA(Interlocuteur, x)$$

Dans ce dernier exemple, nous avons utilisé le prédicat "*MangerA*" pour indiquer la location où on a mangé. Si on veut décrire la location, le temps, etc. pour chaque évènement, on doit enrichir notre domaine avec des prédicats verbe-location, verbe-temps, etc. Cela va causer le domaine à exploser; un nombre énorme des prédicats. Aussi, on ne peut pas utiliser un nombre variable des arguments pour chaque prédicat puisque dans FOL chaque prédicat a un nombre précis des arguments. Une solution est d'introduire une variable d'évènement et on utilise des prédicats pour indiquer la location de l'évènement (*Location*), temps de l'évènement (*Temps*), etc. Le prédicat de l'évènement sera utilisé avec un prédicat pour l'agent et un autre pour le patient. L'exemple précédent sera représenté comme suit :

$$\exists x\, \exists e\, Restaurant(x) \wedge PresDe(EmplacementDe(x), EmplacementDe(ESI))$$

$$\wedge Manger(e) \wedge Mangeur(e, Interlocuteur) \wedge Location(e, x)$$

Ici, l'évènement "e" a été représenté comme une variable. Afin de définir le type de l'évènement, on a utilisé un prédicat unitaire "$Manger(e)$". Afin de représenter les arguments de cet évènement, on a utilisé des prédicats binaires qui sont des rôles. Le premier argument est la variable d'évènement et le deuxième est le participant ayant ce rôle. Cette représentation est appelée une représentation des évènements néo-Davidsonienne d'après le philosophe Donald Davidson (Davidson, 1967).

3.2 Graphes (AMR)

Abstract Meaning Representation (AMR) est un langage de représentation sémantique (Banarescu et al., 2013) qui peut être représenté sous forme d'un graphe. Le graphe doit être raciné, étiqueté, dirigé et acyclique. Il sert à représenter une phrase indépendamment de la syntaxe. Toutefois, il reste dépendant de l'anglais et ne peut pas être considéré comme un langage multilingue. Un exemple de la représentation de la phrase "The boy wants to go" est présenté dans la figure 7.6. La représentation AMR est illustrée sous deux formats : textuelle et graphique, en plus de la représentation logique.

Format logique

∃ w, b, g :
instance(w, want-01)
∧ instance(g, go-01)
∧ instance(b, boy)
∧ arg0(w, b)
∧ arg1(w, g)
∧ arg0(g, b)

Format AMR

```
(w / want-01
:arg0 (b / boy)
:arg1 (g / go-01
:arg0 b))
```

Format Graphe

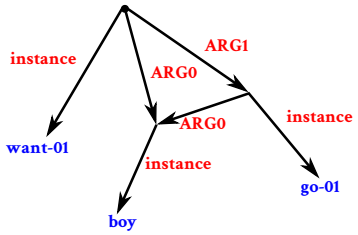


Figure 7.6 : *Exemple de la représentation AMR de la phrase : "The boy wants to go"*

Dans AMR, les concepts d'une phrase sont représentés sous forme des mots de l'anglais (Ex. boy), des concepts de **PropBank** (Ex. want-01) ou des mots clés spéciaux. Le langage suit le modèle néo-Davidsonien ; chaque entité, évènement, propriété et état sont représenté comme une variable. Par exemple, la représentation "(b / boy) veut dire "b" est une instance du concept "boy". Les entités sont reliées par des relations. Par exemple, la représentation "(d / die-01 :location (p / park))" veut dire qu'il y avait un mort "d" dans le parc "p". AMR utilise les arguments des cadres **PropBank** en plus d'autres relations présentées dans le tableau 7.5.

Arguments du cadre	:arg0, :arg1, :arg2, :arg3, :arg4, :arg5
Relations sémantiques générales	:accompanier, :age, :beneficiary, :cause, :compared-to, :concession, :condition, :consist-of, :degree, :destination, :direction, :domain, :duration, :employed-by, :example, :extent, :frequency, :instrument, :li, :location, :manner, :medium, :mod, :mode, :name, :part, :path, :polarity, :poss, :purpose, :source, :subevent, :subset, :time, :topic, :value
Relations de quantité	:quant, :unit, :scale
Relations de date	:day, :month, :year, :weekday, :time, :timezone, :quarter, :dayperiod, :season, :year2, :decade, :century, :calendar, :era
Relations de liste	:op1, :op2, :op3, :op4, :op5, :op6, :op7, :op8, :op9, :op10

Tableau 7.5 : *Relations AMR*

4 Analyse sémantique

Dans l'analyse sémantique, on va présenter comment passer d'une phrase en langage naturel vers une représentation de la logique du premier ordre. Cette analyse est appliquée au fur et à mesure avec l'analyse syntaxique en utilisant des règles sémantiques. Ces dernières peuvent être des λ-expressions qui décrivent des fonctions anonymes sur des variables. Une λ-expression est écrite sous forme "$\lambda x.P(x)$" ; elle représente une fonction. Ces fonctions peuvent être appliquées par une opération appelée λ-Reduction qui sert à substituer une variable par une expression. Il y a deux annotations $\lambda x.P(x)(A)$ ou $\lambda x.P(x)@A$ pour dire : substituer la première variable par "A" (dans ce qui suit, on va utiliser la deuxième). Prenons une fonction avec deux variables qui représente le fait que la première variable aime la deuxième :

$$\lambda y.\lambda x.LIKES(x,y)$$

Le remplacement commence toujours par le premier λ :

$$\lambda y.\lambda x.LIKES(x,y)@BRIT = \lambda x.LIKES(x,BRIT)$$

Après la réduction, le résultat reste toujours une λ-expression. Donc, on peut appliquer une deuxième λ-Reduction :

$$\lambda x.LIKES(x, BRIT)@ALEX = LIKES(ALEX, BRIT)$$

Maintenant, on revient à l'analyse sémantique. Étant donné la grammaire $G < \Sigma, N, P, S >$, on affecte pour chaque variable de N une réalisation sémantique (Ex. NP.sem). Pour chaque production de P, on affecte une opération sémantique : une expression en FOL, une λ-expression ou une λ-Reduction. La réalisation sémantique de la variable à gauche de la production sera l'exécution de l'opération sémantique. Prenons l'exemple des règles syntaxiques annotées sémantiquement présentées dans le tableau 7.6.

S	→ NP VP	VP.sem@NP.sem
VP	→ V_t NP	V_t.sem@NP.sem
VP	→ V_i	V_i.sem
V_t	→ likes	λy.λx.LIKES(x, y)
V_i	→ sleeps	λx.SLEEPS(x)
NP	→ Alex	ALEX
NP	→ Brit	BRIT

Tableau 7.6 : *Grammaire à contexte libre minimale avec les annotations sémantiques*

Lors de l'analyse syntaxique, on calcule la valeur sémantique de chaque variable visitée dans l'arbre syntaxique. Dans notre cas, on va utiliser une analyse ascendante ; parcours postfixe de l'arbre syntaxique. La figure 7.7 est un exemple d'une arbre syntaxique/sémantique de la phrase "A dog likes Alex". On commence par appliquer la règle "NP → Alex" pour avoir la réalisation sémantique :

$$NP.sem = ALEX$$

Ensuite, on applique la règle "V_t → likes" pour avoir :

$$V_t.sem = \lambda y.\lambda x.LIKES(x, y)$$

Après, on applique la règle "NP → Brit" pour avoir la réalisation sémantique :

$$NP.sem = BRIT$$

Ces deux dernière variables nous permet d'appliquer la règle "VP → V_t NP" pour avoir la réalisation sémantique :

$$\begin{aligned} VP.sem &= V_t.sem@NP.sem \\ &= \lambda y.\lambda x.LIKES(x, y)@BRIT \\ &= \lambda x.LIKES(x, BRIT) \end{aligned}$$

Finalement, on peut appliquer la règle "S → NP VP" pour avoir :

$$\begin{aligned} S.sem &= VP.sem@NP.sem \\ &= \lambda x.LIKES(x, BRIT)@ALEX \\ &= LIKES(ALEX, BRIT) \end{aligned}$$

Maintenant, essayons d'analyser la phrase "A dog likes Alex" (certain chien aime Alex). On peut la représenter en FOL comme "$\exists x\ DOG(x) \wedge LIKES(x, ALEX)$". Les règles sémantiques pour prendre les

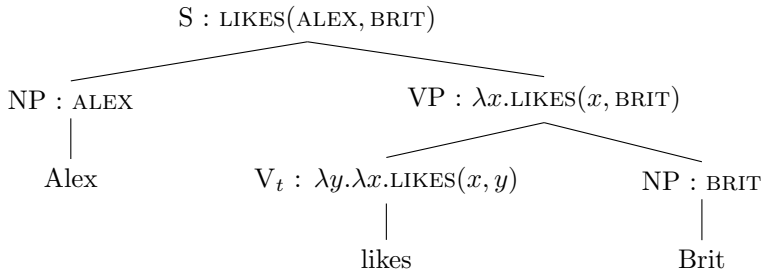


Figure 7.7 : *Exemple d'une grammaire syntaxique-sémantique, ainsi que l'arbre de dérivation de la phrase "Alex likes Brit" (Eisenstein, 2018)*

quantificateurs en considération sont indiquées dans le tableau 7.7. Le quantificateur doit être ajouté lorsqu'on rencontre un déterminant (a, an) et donc on aura $\exists x$. Aussi, on sait qu'avec un quantificateur, on doit avoir un prédicat qui décrit le type de x, donc $\exists x P(x)$. Ce prédicat est le nom qui suit le quantificateur ; donc, on peut définir une λ-expression qui remplace P par la réalisation sémantique du nom suivant, d'où $\lambda P.\exists x P(x)$. Il faut aussi ajouter un ou plusieurs prédicats qui définissent la relation de x avec les autres termes. Vu qu'on peut générer plusieurs prédicats avec une seule λ-expression, on peut définir une seule $\lambda P.P(x)$. λP doit être réduite avant λQ (le nom est plus proche au déterminant), donc $\lambda P.\lambda Q.\exists x\ P(x) \wedge Q(x)$. Dans ce cas :

$$\lambda P.\lambda Q.\exists x\ P(x) \wedge Q(x) @ DOG = \lambda Q.\exists x\ DOG(x) \wedge Q(x)$$

S	→ NP VP	NP.sem@VP.sem	DET	→ every	$\lambda P.\lambda Q.\forall x(P(x) \Rightarrow Q(x))$
VP	→ V_t NP	V_t.sem@NP.sem	V_t	→ likes	$\lambda P.\lambda x.P(\lambda y.LIKES(x,y))$
VP	→ V_i	V_i.sem	V_i	→ sleeps	λx.SLEEPS(x)
NP	→ DET NN	DET.sem@NN.sem	NN	→ Dog	DOG
NP	→ NNP	$\lambda P.P(NNP.sem)$	NNP	→ Alex	ALEX
DET	→ a	$\lambda P.\lambda Q.\exists x\ P(x) \wedge Q(x)$	NNP	→ Brit	BRIT

Tableau 7.7 : *Grammaire à contexte libre minimale avec les annotations sémantiques*

Supposons que le NP précédent est celui généré par la racine "S →NP VP". Si on garde la grammaire précédente (tableau 7.6), le contenu du syntagme verbale sera $\lambda x.LIKES(x, BRIT)$ et la sémantique de la phrase sera :

$$\begin{aligned} S.sem &= VP.sem@NP.sem \\ &= \lambda x.LIKES(x, ALEX)@(\lambda Q.\exists x\ DOG(x) \wedge Q(x)) \\ &= LIKES(\lambda Q.\exists x\ DOG(x) \wedge Q(x), BRIT) \end{aligned}$$

Cela n'est pas juste ; on peut voir que c'est toujours une λ-expression. Ce qu'on veut faire est de réduire λQ avec le contenu sémantique du syntagme verbale, et donc :

$$\begin{aligned} S.sem &= NP.sem@VP.sem \\ &= (\lambda Q.\exists x\ DOG(x) \wedge Q(x))@(\lambda x.LIKES(x, ALEX)) \\ &= \exists x\ DOG(x) \wedge \lambda x.LIKES(x, ALEX))@x \\ &= \exists x\ DOG(x) \wedge LIKES(x, ALEX) \end{aligned}$$

Lorsqu'on veut générer l'exemple du début (Alex loves Brits), on doit appliquer "NP.sem@VP.sem". Mais, on sait clairement que $NP.sem = ALEX$ n'est pas une λ-expression ; on ne peut pas la réduire. La solution

est d'appliquer "VP.sem@NP.sem" sans changer la règle sémantique "NP.sem@VP.sem". On doit, donc, ajouter une expression $\lambda P.P(NNP.sem)$ dans le syntagme nominal qui génère un nom propre. Dans notre cas, $NNP.sem = ALEX$ et donc :

$$\begin{aligned} S.sem &= NP.sem@VP.sem \\ &= (\lambda P.P(ALEX))@(\lambda x.LIKES(x, BRITS)) \\ &= \lambda x.LIKES(x, BRITS))@ALEX \\ &= LIKES(ALEX, BRITS) \end{aligned}$$

On a supposé que la réalisation sémantique du **syntagme** verbal est $\lambda x.LIKES(x, BRITS)$. On va essayer de calculer la réalisation avec la règle sémantique de $VP \rightarrow V_t NP$ avant sa modification (tableau 7.6).

$$\begin{aligned} VP.sem &= V_t.sem@NP.sem \\ &= \lambda y.\lambda x.LIKES(x, y)@(\lambda P.P(BRIT)) \\ &= \lambda x.LIKES(x, \lambda P.P(BRIT)) \end{aligned}$$

Ce n'est pas le résultat voulu. Pour un verbe transitif, on veut résoudre le deuxième argument y en premier. Donc, on doit lui passer "BRIT" et pour le faire on doit nous débarrasser de λP. Dans ce cas, on laisse λx à coté et on essaye d'appliquer λP sur le reste. La règle sémantique sera $\lambda P.\lambda x.P(\lambda y.LIKES(x, y))$, et donc :

$$\begin{aligned} VP.sem &= V_t.sem@NP.sem \\ &= \lambda P.\lambda x.P(\lambda y.LIKES(x, y))@(\lambda P.P(BRIT)) \\ &= \lambda x.\lambda P.P(BRIT)@(\lambda y.LIKES(x, y)) \\ &= \lambda x.\lambda y.LIKES(x, y)@(BRIT) \\ &= \lambda x.LIKES(x, BRIT) \end{aligned}$$

L'analyse sémantique de la phrase "A dog likes Alex" est illustrée sous forme d'un arbre dans la figure 7.8. Cette analyse se fait en parallèle avec l'analyse syntaxique comme l'analyse **CKY**. Lorsqu'une règle syntaxique est appliquée, on applique sa règle sémantique équivalente. Nous avons vu que les grammaires des langages naturels sont plus complexes et donc non déterministes. **CKY** probabiliste peut être utilisée pour résoudre le problème d'ambigüité. Sinon, on peut utiliser le sens généré afin de guider l'analyse. Donc, étant donné une phrase w et une fonction de score Φ qui possède des paramètres θ, la forme sémantique finale $\hat{z}$ est celle qui maximise cette fonction de score comme indiqué dans l'équation 7.1.

$$\hat{z} = \arg\max_{z} \Phi(z|w, \theta) \tag{7.1}$$

La phrase w, étant une séquence, Φ peut être représentée comme une fonction séquentielle qui génère la représentation suivante sachant des caractéristiques comme le mot courant, les mots passés, leurs représentations, etc. Donc, afin de maximiser Φ, on doit tester plusieurs chemins d'analyse. Une des méthodes pour optimiser ce processus est d'utiliser **Beam Search**. L'annotation manuelle des représentations sémantiques est vraiment une tâche couteuse. Une autre méthode pour entraîner le modèle est en utilisant les dénotations. Par exemple, si la tâche de l'analyse sémantique a comme but d'avoir la représentation sémantique des questions, on peut utiliser SQL comme représentation. La dénotation veut dire attribuer à chaque question une liste des résultats possibles à partir d'une base de donnée précise. Dans ce cas, lors de l'entraînement, on ne va pas tester si la requête SQL générée est correcte ; mais, on va tester le résultat de son exécution.

Parmi les APIs qui nous permettent d'analyser des phrases sémantiquement, on peut citer **NLTK** [5]. Dans

5. NLTK semantic parsing : https://www.nltk.org/book/ch10.html [visité le 2021-09-11]

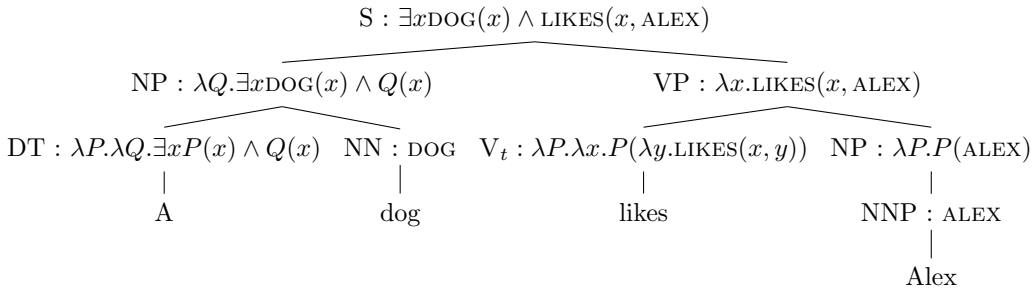


Figure 7.8 : *Exemple d'une grammaire syntaxique-sémantique avec quantificateurs, ainsi que l'arbre de dérivation de la phrase "A dog likes Alex" (Eisenstein, 2018)*

le code suivant, le résultat est :

all z2.(dog(z2) -> exists z1.(bone(z1) & give(angus,z1,z2)))

```
from nltk import load_parser
parser = load_parser('grammars/book_grammars/simple-sem.fcfg', trace=0)
sentence = 'Angus gives a bone to every dog'
tokens = sentence.split()
for tree in parser.parse(tokens):
    print(tree.label()['SEM'])
```

Discussion

La forme syntaxique ne porte pas le sens de la phrase ; elle sert à vérifier qu'une phrase est bien écrite selon la langue en question. Les rôles sémantiques des différents éléments d'une phrase peuvent être déduits à partir des fonctions syntaxiques. Par exemple, on peut appliquer une analyse syntaxique et faire une correspondance avec les rôles sémantiques. Un sujet du verbe peut être considéré comme un agent. Mais, cela n'est pas toujours le cas ; dans la forme passive, le sujet est un patient. Donc, la transition du niveau syntaxique vers le niveau sémantique n'est pas aussi simple que ça. Il existe plusieurs projets qui visent à faciliter cette tâche comme FrameNet et PropBank. Ils représentent les phrases sous forme des cadres où l'évènement est l'élément central.

Représenter une phrase comme cadres est utile dans des tâches de compréhension du langage. Mais, des fois, cette représentation n'est pas adéquate pour certaines tâches. Dans le raisonnement par machine, on veut avoir une machine qui peut appliquer des déductions. Par exemple, on peut trouver des contradictions entre les phrases d'un article de presse. Une des représentations qui peut nous permet ça est la logique propositionnelle et plus précisément la logique du premier ordre. Afin de générer une telle représentation, on peut utiliser une grammaire constituante et affecter à chaque règle syntaxique une règle sémantique. La forme sémantique n'est pas toujours une expression de la logique du premier ordre. Elle peut être une requête SQL, si on veut implémenter un système de Question/réponse. Aussi, elle peut être une commande, si on veut implémenter un assistant personnel intelligent comme Alexa, Cortana ou Siri.

CHAPITRE 8

DÉTECTION DE LA CORÉFÉRENCE

LES LANGUES

Une coréférence est une relation entre deux termes : un substitut et son antécédent. Les prénoms personnels sont un exemple des références. Dans la plupart des fois, les phrases ne sont totalement claires que lorsqu'on connaisse le sens de leurs références. La tâche qui nous permet ça est la résolution de la coréférence. Parmi les tâches similaires à celle-ci, on peut citer : l'annotation sémantique et la reconnaissance des entités nommées. Dans ce chapitre, on va discuter les références dans le contexte linguistique ainsi que la résolution de la coréférence.

Prenons l'exemple "La fille a cueilli une fleur. Elle l'a sentie. Elle a une très bonne odeur." représentant trois phrases consécutives. Afin de comprendre le sens de la deuxième phrase, on doit savoir que "elle" veut dire "la fille" et "l" veut dire "une fleur". Le pronom "elle" de la troisième phrase est plus difficile à résoudre ; il référence la fille ou la fleur ? En introduisant le sens de la deuxième phrase "la fille a sentie une fleur", on peut déduire que "elle" est une référence à "une fleur". Plusieurs tâches peuvent motiver la résolution des coréférences :

- Résumé automatique : dans le cas d'un résumé automatique extractif, on essaye d'extraire les phrases les plus pertinentes. Supposons que le résumé du texte précédent soit la troisième phrase. Comme étape de post-traitement, on doit remplacer les références absentes afin de donner plus de sens au résumé.
- Questions/Réponses : dans l'exemple précédent, la réponse de la question "Qui a une bonne odeur ?" n'est pas "elle" mais plutôt "la fleur".
- Système de dialogue : le système doit pouvoir lier les références utilisées par l'utilisateur afin de mener une conversation.
- Traduction automatique : il existe des langues qui n'utilisent pas des références dans leurs phrases ; on doit trouver la référence en utilisant le contexte. Dans ce cas, on peut utiliser cette tâche afin de trouver l'entité référencée.

1 Références

Une référence est définie linguistiquement dans Larousse comme : "*Propriété des signes linguistiques leur permettant de renvoyer à des entités extralinguistiques (objets ou individus appartenant au monde réel ou à un monde imaginaire)*". Elle est connue aussi comme "référent". Dans cette section, on va discuter les formes des références, les manières de référencement et leurs propriétés.

1.1 Formes des références

Les références peuvent être classifiées selon les catégories syntaxiques, en plus d'autres critères (Schmolz, 2015). La forme la plus connue des références est l'utilisation des pronoms et surtout les pronoms personnels. Un **pronom** est un mot-outil utilisé afin de se substituer à un nom ou un syntagme nominal. La catégorie la plus connue est le pronom personnel qui serve à représenter les trois types des personnes grammaticales : l'énonciateur, le destinataire et la personne absente. Un exemple de ce genre de références est : "Karim est entré. Il a commencé le cours.". Un pronom possessif est un pronom qui renvoie à un objet possesseur (et des fois à un objet possédé). Exemple, "Karim a commencé son cours.". Résoudre la référence "son" revient à répondre à la question "le cours de qui ?". Un pronom démonstratif le nom de ce qu'on veut montrer. Exemple, "Ces livres sont intéressants. Je vous conseille celui-ci.". Ces trois sont les catégories les plus utilisées des pronoms.

Une autre forme de référence peut être les **syntagmes nominaux**. Ils peuvent être utilisés pour référencer des entités. Par exemple, "J'ai un petit chat. Cet animal est très méchant.". Afin de comprendre la deuxième phrase, on doit savoir de quel animal on est en train de parler. On peut trouver des **syntagme**s nominaux avec un article défini qui sont en réalité des références. Par exemple, "J'ai un petit chat. Le chat est très méchant.". Contrairement aux pronoms qui sont toujours des références, les **syntagmes** nominaux peuvent ne pas l'être. Donc, en plus de chercher l'antécédent, on doit aussi décider si c'est une référence ou non.

Les **noms propres** ne sont pas considérés comme références, mais ils peuvent être liés avec d'autres noms propres. En général, le nom propre référence un autre nom propre complet (avec plus d'information). On peut citer les abréviations qui peuvent être des références vers leurs versions complètes. Un exemple, "L'école nationale supérieure d'informatique se situe à Alger. Comme toutes les universités algériennes, il faut avoir le BAC pour étudier à l'ESI.". Aussi, on peut trouver ça dans les noms propres qui sont longs ; où la répétition prend plus d'espace. Dans la plupart des langues, on a tendance à utiliser des formes plus contractes des noms propres. Exemple, "La république algérienne démocratique et populaire est un pays d'Afrique du nord. L'Algérie a une superficie de plus de 2 millions km^2".

Il existe des langues où on omit complètement les références ; ces dernières sont connues par le nom **Anaphore zéro**. Lorsqu'on ne veut pas répéter le même **syntagme** nominal, on peut l'omettre. Par exemple, "Karim a présenté et ϕ a expliqué le cours.". Pour trouver le sujet du verbe "expliquer", on doit résoudre une référence qui n'existe pas ; marquée ici par "ϕ". Dans des langues, comme le japonais, il est plus naturel d'omettre les pronoms personnels dans les phrases. Par exemple, "カリムさん は ESI に生きます。ϕ あそこに教えます。". Ceci peut être traduit, mot par mot, comme : "M. Karim à l'ESI va. ϕ là-bas enseigne.". Le pronom "il" a été omis et peut être détecté en utilisant le contexte.

Il existe des formes plus complexes qui ne référencent pas un objet mais plutôt une action. Une de ces formes est le **syntagme** verbal (Ex. "Si Mouloud achète un nouveau vélo, je le ferai aussi."). Dans ce cas, un **syntagme** verbal ne référence qu'un autre **syntagme** verbal. On peut utiliser des adverbes comme références (Ex. "Moammed was too busy, and so was I.").

1.2 Manière de référencement

Le référencement peut être classifié selon la position de la référence par rapport le référent en anaphore et cataphore. En terme de nombre des antécédents, on peut avoir un antécédent unique ou des antécédents partagés. Selon la direction de référencement, on peut avoir une référence à direction unique ou deux entités en coréférence.

- **Anaphore** : une référence vers un mot ou un **syntagme** précédent appelé "antécédent". En général, ce terme est utilisé même pour les références des éléments qui viennent après. Mais ici, ce concept a un terme qui lui définit (point suivant). Une anaphore peut être un pronom (Ex. "Le cours est très long. Il prendra plus de temps."), un syntagme nominal (Ex. "J'ai rencontré Fatima. Une personne qui adore assister les autres."), un zéro anaphore (Ex. "Le chat a attrapé la souris et ϕ l'a mangé."), etc.
- **Cataphore** : une référence vers un mot ou un **syntagme** suivant appelé "postcédent". Elle est comme l'anaphore, mais le référent vient après la référence. Exemple d'un pronom (Ex. "Il est très long, ce cours !"), syntagme nominal (Ex. "If you want a book, take the one on the shelf."), etc. Ici, la référence de type zéro anaphore ne peut pas se manifester comme une cataphore. Puisque dans les langues qui utilisent ce phénomène, on peut comprendre la référence à partir du contexte (ce qu'on a déjà dit).
- **Antécédents partagés** : une référence vers plusieurs mots et/ou **syntagmes**. Que ce soit une anaphore ou une cataphore, une référence peut avoir plusieurs référents dans une seule relation. Exemple, "Le cours sera suivi par un exercice. Ils sont importants pour la compréhension.". Ici, la relation n'est pas binaire ; le référent est une combinaison de plusieurs éléments.
- **Syntagmes nominaux en coréférence** : deux **syntagmes** nominaux où chacun est une référence vers l'autre. Dans ce cas, le premier est un référent antécédent du deuxième et au même temps le deuxième est un référent postcédent du premier. Exemple, "Certains de nos collègues nous ont vraiment soutenu. Ce genre de personnes gagne notre gratitude.".

Dans ce qu'on a présenté, le référent existe dans le texte ; il est textuel. On appelle ça "**Endophore**", qui comporte les anaphores et les cataphores. Lorsque la référence est contextuelle (elle n'existe pas dans le texte), on l'appelle "**Exophore**". L'exophore est un élément qui ne référence aucun objet ou situation ; il veut dire que la référence ne puisse être connue qu'à base du contexte (Halliday et Hasan, 2014). Dans l'exemple "Je vais là-bas", le mot "là-bas" ne peut être compris si on est avec l'interlocuteur. Cette référence peut être un endophore si la destination a été mentionnée dans le texte. Mais lorsqu'on parle avec quelqu'un, on n'utilise pas que la langue parlée ; on utilise aussi les signaux.

1.3 Propriétés des relations de coréférence

Afin de détecter le référent, on peut utiliser son agrément avec la référence basé sur certaines propriétés. Quelques traits grammaticaux doivent être identiques pour les deux : référence et référent. Ici on va discuter les trois traits qui définissent les pronoms : le nombre, la personne et le genre. Bien sûr, il existe des langues qui ne supportent pas les trois traits. Selon le **nombre**, une entité peut être en : singulier, duel ou pluriel (il existe d'autres formes). Dans l'exemple "Les oiseaux sont sur une arbre. Ils chantent", le pronom "ils" référence "les oiseux" puisque les deux sont en pluriel. Dans le cas des antécédents partagés, cette propriété va être plus difficile à gérer. Par exemple, "Le cours sera suivi par un exercice. Ils sont importants pour la compréhension.". Il y a aussi le problème des entités singuliers qui sont sémantiquement en pluriel. Dans leur cas, on peut utiliser le singulier ou le pluriel comme référence. Par exemple, l'entreprise "IBM" peut être référencée par "elle" ou "ils". Selon la **personne**, une entité peut être considérée comme : première, deuxième ou troisième personne. En général, la référence et le référent doivent être en agrément en terme de personne. Exemple, "Mon frère$_1$ et moi$_2$ avons réparé son vélo$_1$ après le mien$_2$.". Mais, on peut trouver des textes où l'interlocuteur utilise la troisième personne à la place de la deuxième pour avoir un effet comique ou pour attirer l'attention de quelqu'un. Selon le **genre**, une entité peut être : masculin, féminin ou neutre. Exemple, "Lorsque la fille a rencontré son père, il a été content.".

En plus des traits grammaticaux, il existe certaines propriétés qui aident la détection du référent (antécédent). Les **contraintes de théorie contraignante** sont des contraintes syntaxiques sur la relation

mention-antécédent. Un pronom réfléchi se réfère à l'agent de l'action qui est en général un sujet. En français, il peut être conjoint (me, te, se, nous, vous) ou disjoint (moi-même, soi, nous-même, etc.). Un exemple en anglais, "Janet bought herself a bottle of fish sauce.". Ici, on sait que "herself" est une référence vers "Janet" contrairement au pronom "her" dans l'exemple "Janet bought her a bottle of fish sauce.". La **récence** veut dire que les entités mentionnées plus récemment sont plus probables d'être un antécédent. Exemple, "Le médecin a trouvé une vieille carte. Jim a trouvé une carte encore plus ancienne. Elle décrivait une île.". En se basant sur le **rôle grammatical**, les sujets sont plus probables d'être des référents que les objets. Exemple, "Karim est allé au restaurant avec son ami. Il a demandé un plat de couscous.". La **sémantique du verbe** joue un rôle dans la détection du référent. La mention suit l'emphase du verbe ; celui qui a causé l'évènement. Dans l'exemple "John telephoned Bill. He lost the laptop.", l'évènement de téléphone a été causé par le sujet ; donc le pronom "he" revient au sujet. Contrairement à l'exemple "John criticized Bill. He lost the laptop." où la critique a été causée par l'objet. Certes, l'agent ici c'est le sujet ; mais c'est l'objet qui a déclenché l'évènement de critique.

2 Résolution des coréférences

La résolution des coréférences passe par deux étapes : détection de la mention et liaison des coréférences. Tout d'abord, il faut détecter les différentes références dans le texte. Ensuite, il faut lier ces références avec leurs référents. Il existe plusieurs modèles de liaison : Mention-Pair, Mention-Rank et Entity-based. La résolution des coréférences peut être accomplie en suivant une des deux approches : par règles ou par apprentissage automatique. Dans cette dernière, on peut se baser sur des caractéristiques définies manuellement ou par les **embeddings**.

2.1 Détection de mention

Les mentions sont des entités qui peuvent être des **syntagmes** nominaux ou des pronoms. L'identification des mentions en coréférence est une étape importante dans des tâches de TALN comme la résolution des coréférences et la reconnaissance des entités nommées. Cette étape peut être appliquée en utilisant un ensemble des règles ; comme celles définies par Lee et al. (2013). On commence par marquer les syntagmes nominaux, les pronoms et les entités nommées qui n'ont pas été marqués dans le texte comme des mentions candidates. Ensuite, on sélectionne les mentions par élimination comme suit :

- On supprime une mention s'il existe une autre mention plus grande ayant la même tête syntaxique. Par exemple, on doit supprimer "La boite de développement" si une autre mention plus grande existe "La boite de développement située à Alger".
- Il faut écarter les entités numériques telles que les pourcentages, l'argent, les cardinaux et les quantités. Par exemple, "9%", "100$", "cent neuf".
- On élimine les mentions qui sont une partie d'une autre (quantificateur + "of"). Exemple, "a total of 177 projects", "none of them", "millions of people".
- On supprime les pronoms qui ne sont pas utilisés comme référence ("it" pléonastique). Dans l'exemple "It is thought that ...", le pronom "it" n'est pas une référence. On peut détecter un tel pronom en utilisant une liste des verbes cognitifs (Ex. "believe, think, etc.").
- On élimine les formes adjectivales des nations ou les acronymes nationales (Ex., "American", "U.S.", "U.K.").

Une autre méthode pour détecter si une mention est une coréférence ou non est en utilisant l'apprentissage automatique. Dans (Uryupina et Moschitti, 2013), on commence par la génération de l'arbre syntaxique d'une phrase. On parcourt les nœuds et on classe les syntagmes nominaux comme coréférence ou non en utilisant un SVM. Les caractéristiques utilisées sont les propriétés présentées précédemment

comme le nombre, le genre, etc. Autres systèmes ont été développés, mais ils sont liés à une tâche spécifique et ne peuvent pas être utilisés séparément. La figure 8.1 représente trois architectures de détection des mentions proposées par Yu et al. (2020). Étant donné un document D, on commence par considérer toutes les mentions possibles x_i^* marquées par le mot du début s_i et le mot de fin e_i. Dans le système (c), on utilise un modèle **BERT** pré-entraîné pour générer les représentations des mots x_t^*. Pour chaque mention possible de cette phrase, on prend les représentations du début x_{si}^* et de fin x_{ei}^* et les concaténer pour avoir la représentation du mention. Cette dernière est utilisée comme entrée à un réseau de neurones à propagation avant (FFNN) afin de classer le mention comme coréférence ou non. Dans le système (b), on utilise un modèle **ELMo** pré-entraîné afin de générer les représentations des mots comme dans le système (c). Le code de chaque mot est une concaténation des couches cachées du modèle. On utilise deux réseaux de neurones à propagation avant (FFNN) afin d'encoder le début et la fin sous forme de deux vecteurs. Ces derniers sont utilisés comme entrée d'un troisième FFNN afin de classer la mention comme coréférence ou non. Le système (a) est un peu plus complexe : pour chaque mot, il utilise une représentation en utilisant **GloVe**, une autre en utilisant **ELMo** et une troisième basée sur les caractères en utilisant les CNNs. On applique une attention sur $\{x_{si}^* \dots x_{ei}^*\}$ pour avoir une représentation de la mention h_i^*. Comme les deux autres systèmes, on utilise un FFNN pour classer la mention comme coréférence ou non. Comme entrée, on utilise la représentation du mot du début x_{si}^*, du mot de fin x_{ei}^*, de la mention h_i^* et le **embedding** de la taille de la mention $\phi(i)$.

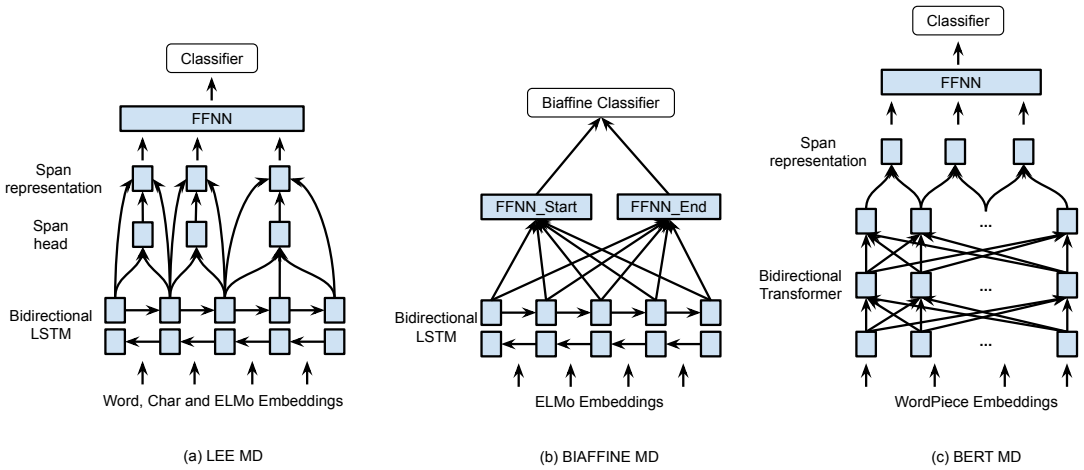


Figure 8.1 : *Trois architectures pour la détection des mentions (Yu et al., 2020)*

2.2 Liaison des coréférences

La liaison des coréférences consiste à lier une référence à son référent. En général, cette opération est appelée regroupement des mentions puisqu'on regroupe toutes les mentions qui référencent la même entité ensemble. Pour ce faire, il y a deux classes des modèles : modèles basés sur les mentions et modèles basés sur les entités. Dans ce qui suit, on va considérer les coréférences comme des anaphores : mention et antécédent.

Modèles Mention-Pair

On entraîne un modèle de classement binaire qui prend deux mentions m_i et m_j et rendre une probabilité que m_i est un antécédent de m_j : $P(Coref|m_i, m_j)$. Les caractéristiques utilisées peuvent être des propriétés des deux mentions comme la personne, le genre, le nombre, etc. Afin de détecter les coréférences, le modèle est appliqué deux à deux. Dans la figure 8.2, le modèle doit estimer une grande probabilité P(coref| "Victoria Chen", "she") et une petite probabilité P(coref| "Megabucks Banking", "she").

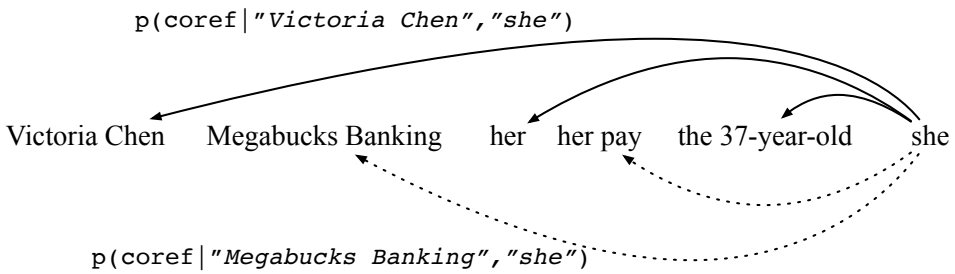


Figure 8.2 : *Exemple d'un modèle Mention-Pair (Jurafsky et Martin, 2019)*

Afin d'entraîner le modèle, on utilise toutes les combinaisons binaires possibles. Le problème est qu'on va avoir plusieurs exemples négatifs ; un dataset déséquilibré. Une solution est de ne prendre en considération que les mentions négatives qui se trouvent entre deux mentions positives.

Modèles Mention-Rank

Ce modèle est similaire au modèle Mention-Pair dans le fait qu'on compare les mentions deux à deux. Mais, la différence est que celui-ci apprenne à classer les antécédents ; il apprend une sorte d'ordre. Dans ce cas, on utilise des différentes caractéristiques sur la référence m_i et l'antécédent m_j afin d'estimer une probabilité conditionnelle $P(m_j|m_i)$. Donc, la tâche consiste à maximiser la probabilité qu'une mention est un antécédent $\hat{a}$ comme indiqué dans l'équation 8.1

$$\hat{a} = \arg \max_{j \in \{\epsilon, 1, \ldots, (i-1)\}} P(w_j|w_i) \tag{8.1}$$

ϵ veut dire la mention n'a pas d'antécédent. La figure 8.3 représente un exemple de l'annotation Mention-Rank.

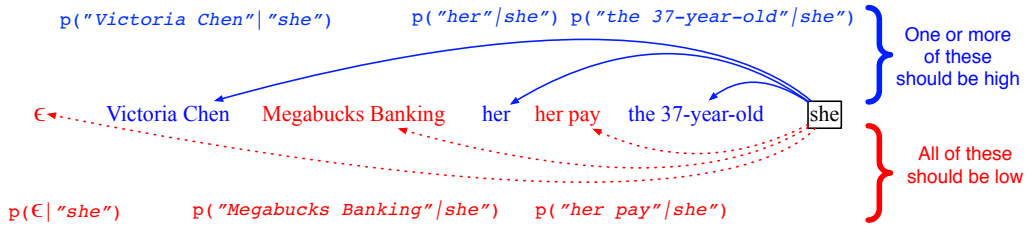


Figure 8.3 : *Exemple d'un modèle Mention-Rank (Jurafsky et Martin, 2019)*

Dans l'étape d'entraînement, on doit choisir un seul exemple positif parmi ceux possibles. Une approche consiste à prendre la mention la plus proche. Une fois un antécédent est détecté, le reste des antécédents sont détectés en utilisant la transitivité.

Modèles Entity-based

Le problème des modèles basés sur la mention est qu'ils comparent les mentions deux à deux ; ils ne prennent pas la relation avec des autres coréférences en considération. Par exemple, Ms Kennedy ← Kennedy, Kennedy ← He. Dans ces modèles, la coréférence est considérée comme un problème de classement ; or elle doit être considérée comme un problème de regroupement (clustering). Une des méthodes est d'appliquer Mention-Rank sur des clusters et pas sur des entités. On commence par la création des clusters d'une seule mention. Étant donné deux clusters des mentions, on vérifie si les deux sont compatibles en utilisant un algorithme d'apprentissage. Si les clusters sont compatibles, on les fusionne et on

applique ça jusqu'à ce que les clusters restants ne sont plus compatibles. Une autre méthode est de préparer tous les clusters possibles et extraire des caractéristiques du cluster. Ensuite, entrainer un modèle (en général, réseau de neurones) afin de décider si c'est un cluster valable ou non.

3 Évaluation

Il existe plusieurs métriques pour évaluer la résolution des coréférences (Moosavi et Strube, 2016). Dans **MUC** (Message Understanding Conference), l'évaluation est basée sur le nombre des liens binaires communs entre la référence et le système. Le corpus de test contient un texte et un ensemble de tous les liens binaires possibles. Cette métrique favorise les systèmes avec des chaînes larges. Aussi, elle ne prend pas en considération les singletons (les mentions sans antécédent).

B^3 est une métrique basée sur les mentions (mention-based metric). Le rappel et la précision globaux sont calculés en terme des rappels et précisions locaux par rapport à une chaîne de mentions. Supposant que nous avons un nombre des clusters de mentions destinataires (hypothèse), on annote H_i le cluster contenant la mention m_i. Donc, le cluster contenant la mention m_i selon le système est annoté S_i. Si nous avons N mentions, le rappel peut être calculé en utilisant le nombre des mentions en commun comme indiqué dans l'équation 8.2 où w_i est un poids attribué à la mention w_i.

$$R = \sum_{i=1}^{N} w_i \frac{|H_i \bigcap S_i|}{|H_i|} \tag{8.2}$$

CEAF (Constrained entity-alignment F-Measure) utilise chaque mention une seule fois lors du calcul. Pour ce faire, elle aligne une entité (cluster de mentions) S_i du système avec une autre entité H_j d'hypothèse (annotation manuelle) en utilisant une mesure de similarité. Deux mesures de similarité ont été utilisées dans les équations 8.3.

$$\phi_1(H_i, S_j) = |H_i \cap S_i| \quad \phi_2(H_i, S_j) = \frac{2|H_i \cap S_i|}{|H_i| + |S_j|} \tag{8.3}$$

Pour calculer le rappel, on cherche les clusters qui sont les plus similaires selon une fonction de similarité ϕ (voir l'équation 8.4).

$$R = \frac{\max_{i,j} \phi(H_i, S_j)}{\sum_i \phi(H_i, H_i)} \tag{8.4}$$

BLANC (BiLateral assessment of noun-phrase coreference) est une métrique basée sur les liens (link-based metric). Le rappel (précision) globale est la moyenne des rappels (précisions) des liens de coréférences et ceux des non coréférences. Les liens de coréférence sont représentées comme un tuple (référence, référent). Donc, étant donné un nombre N de mentions, on aura $N(N+1)/2$ liens de coréférences possibles. Le tableau 8.1 représente la matrice de confusion de BLANC où "w" veut dire "wrong", "r" veut dire "right", "c" veut dire "coreference" et "n" veut dire "Non-coreference". Dans ce cas, le rappel est calculé selon l'équation 8.5.

$$R_c = \frac{rc}{rc + wn}, \quad R_n = \frac{rn}{rn + wc}, \quad R = \frac{R_c + R_n}{2} \tag{8.5}$$

LEA (Link based entity aware) vise à représenter le rappel et la précision en terme de l'importance d'une entité et comment elle a été résolue. La taille d'une entité (nombre des mentions dans le cluster) est considéré comme mesure de son importance. La proportion du nombre des liens des mentions partagées entre l'hypothèse et le système par rapport le nombre des liens des mentions de l'hypothèse est considérée

		Système	
		Coréf	Non-Coréf
Hypothèse	Coréf	rc	wn
	Non-Coréf	wc	rn

Tableau 8.1 : *Matrice de confusion de BLANC (Recasens et Hovy, 2011)*

comme mesure de qualité de résolution. Donc, le rappel sera calculé comme indiqué dans l'équation 8.6 où *link* compte le nombre des liens dans un cluster.

$$R = \frac{\sum_{H_i}(|H_i| \times \sum_{S_j} \frac{link(H_i \bigcup S_j)}{H_i})}{\sum_{H_k} link(H_k)} \tag{8.6}$$

4 Tâches connexes

Afin d'accomplir la tâche de détection de la coréférence, il existe des tâches de prétraitement vues dans le chapitre 2, comme la tâche de segmentation du texte. Une autre tâche utile pour détecter les coréférences est l'étiquetage morpho-syntaxique vue dans le chapitre 4. L'analyse syntaxique (chapitre 5) peut être utilisée pour détecter les syntagmes nominaux. Les entités nommées sont un critère important dans la résolution des coréférences. La tâche de reconnaissance des entités nommées sera discutée brièvement dans cette section.

Il existe des tâches similaires à celle de la résolution des coréférences qui visent à chercher un référent d'une référence dans un texte donné. Il existe des références qui ont besoin de plus de contexte afin de comprendre le texte (niveau pragmatique). Pour ce faire, il existe une tâche appelée annotation sémantique (Entity linking) qui vise à lier des mentions avec des entités d'une base de connaissance. Une autre tâche moins similaire est l'attribution de la citation qui vise à trouver qui a dit/écrit un discours. C'est une sorte de classification où les classes représentent une liste des auteurs/personnes.

4.1 Annotation sémantique (Entity linking)

L'objectif de l'annotation sémantique est d'associer une mention dans un texte à une représentation d'une entité dans une base de connaissances structurée. Cette tâche est motivée par le fait d'avoir une connaissance approfondie sur les entités du texte dans le monde réel (contexte, niveau pragmatique). Cette tâche est liée à la base de connaissance utilisée ; lorsqu'on utilise Wikipédia comme base de connaissance, on appelle cette tâche "Wikification".

Avant d'appliquer cette étape, on doit détecter les mentions dans le texte (comme dans la résolution des coréférences). Afin d'annoter une mention du texte, on suit deux étapes :

- **Détection de la mention** : détecter l'ensemble des entités d'un base de connaissance liées à une mention en utilisant des requêtes. Vu que la base est structurée, la recherche sera facile ; dans Wikipedia, on utilise les titres.
- **Désambigüisation de la mention** : trouver l'entité la plus probable en utilisant l'apprentissage automatique. Comme caractéristiques, on prend celles de la mention et celles de l'entité.

4.2 Reconnaissance des entités nommées

Une entité nommée peut être une personne, une place, une organisation, une quantité, etc. La tâche de reconnaissance des entités nommées consiste à localiser et classer les entités nommées dans un texte. En anglais, elle est appelée “Named-entity recognition” (NER). La localisation se fait en utilisant la détection de mention vue précédemment dans ce chapitre. Afin de classer les mentions, on peut utiliser plusieurs approches :

- **Règles** : utiliser des règles et des listes de noms pour chercher les entités et détecter leurs types.
- **Apprentissage avec caractéristiques** : utiliser les **embeddings** du mot et ses voisins, les préfixes et les suffixes, l’appartenance à une liste, etc.
- **Étiquetage de séquences** : classer les mots en entités en les traitant comme une séquence en utilisant **IOB** vue dans le chapitre 4.

$$\underbrace{Google}_{B\text{–}ORG}\ \underbrace{LLC}_{I\text{–}ORG}\ \underbrace{est}_{O}\ \underbrace{fondée}_{O}\ \underbrace{dans}_{O}\ \underbrace{la}_{O}\ \underbrace{Silicon}_{B\text{–}LOC}\ \underbrace{Valley}_{I\text{–}LOC}\ \underbrace{par}_{O}\ \underbrace{Larry}_{B\text{–}PER}\ \underbrace{Page}_{I\text{–}PER}\ \underbrace{et}_{O}\ \underbrace{Sergey}_{B\text{–}PER}\ \underbrace{Brin}_{I\text{–}PER}$$

Discussion

Beaucoup de langues considèrent la répétition comme une forme non naturelle. Personnellement, je ne sait pas s’il existe des langues où on doit répéter les entités dans chaque phrase. L’être humain a la capacité de détecter le référent (celui qui a été référencé) facilement. Bien sûr, il existe des cas où les références sont tellement ambigües un être humain ne peut pas les résoudre. Il sera vraiment bénéfique si une machine peut exécuter cette tâche. Puisque les pronoms ne sont pas les seules références, cette tâche doit passer par une étape de détection des mentions.

Deux approches sont utilisées pour la liaison des coréférences : les modèles basés sur les mentions et les modèles basés sur les entités. En se basant sur les mentions, on essaye de trouver les relations deux à deux. Cela peut causer un problème d’incompatibilité entre les mentions d’une chaîne. Les modèles basés sur les entités cherchent à regrouper les mentions compatibles ensembles ; c’est une tâche de regroupement (clustering). Malgré que l’utilisation des entités peut éviter le problème d’incompatibilité, l’approche basée sur les mentions est plus utilisée vu la facilité de son implémentation.

Cette tâche est une sorte de classification, donc on peut utiliser le rappel, la précision et le F-score pour l’évaluer. Mais, comme le problème peut être vu comme des liens ou des entités, ces mesures peuvent être formulées de plusieurs façons. Plusieurs mesures ont été présentées dans ce chapitre, ainsi que des tâches similaires à la résolution des coréférences.