UNIVERSITE DE FRANCHE-COMTE

ECOLE DOCTORALE
« LANGAGES, ESPACES, TEMPS, SOCIETES »

Thèse en vue de l'obtention du titre de docteur en

LINGUISTIQUE ET TRAITEMENT AUTOMATIQUE DES LANGUES

ANALYSE LINGUISTIQUE ET FORMALISATION POUR LE TRAITEMENT AUTOMATIQUE DE LA PARAPHRASE

Présentée et soutenue publiquement par

Wannachai KAMPEERA

Le 29 avril 2013

Sous la direction de Madame le professeur Sylviane CARDEY-GREENFIELD

Membres du jury

Krzysztof BOGACKI, Professeur à l'université de Varsovie, Pologne, Rapporteur

Farouk BOUHADIBA, Université d'Oran, Algérie

Sylviane CARDEY-GREENFIELD, Professeur à l'université de Franche-Comté

Christian FLUHR, Professeur à l'Institut Supérieur Arabe de Traduction (ISAT), Alger, Rapporteur

Peter GREENFIELD, HDR à l'université de Franche-Comté

Laurent SPAGGIARI, Docteur en linguistique, Airbus-France

Résumé

L'analyse linguistique faite sur notre corpus du domaine agroalimentaire montre que les relations paraphrastiques entre plusieurs ensembles de paraphrases peuvent se décrire en termes de suites de transformations textuelles. Pour paraphraser, une substitution lexicale initiale se met en route et déclenche ainsi d'autres modifications syntaxiques, lexicales et morphologiques. Substituer un mot par un autre revient à dire qu'il y a remplacement d'un sens par un autre, *à priori* similaire. Dans cette perspective, pour un couple de paraphrases, il est possible d'identifier au moins deux unités lexicales qui, d'après nous, entretiennent la relation « sens source – sens dérivé ». Les autres sens majeurs dans la phrase sont alors considérés régis par ces deux sens.

Après avoir décrit les mécanismes de paraphrasage récurrents dans notre corpus, nous avons proposé une formalisation qui explique les différentes relations paraphrastiques entretenues par les paraphrases entre-elles. Cette formalisation a un but théorique et pédagogique et vient compléter la deuxième formalisation qui forme le noyau de notre travail ; celle des structures paraphrastiques sous-forme de prédicat-argument. On peut ainsi dire que la première formalisation sert à décrire l'action, à savoir les mécanismes de paraphrasage (et les différentes relations paraphrastiques) alors que la deuxième formalisation capte le résultat. Les prédicats correspondent en effet aux « sens sources – sens dérivés », les arguments, eux, à « d'autres sens majeurs régis par les prédicats », comme décrits plus haut. La formalisation des structures paraphrastiques prédicat-argument proposée est celle que nous considérons adaptée au traitement automatique de la paraphrase.

Nous avons ensuite implémenté un système d'extraction des structures paraphrastiques. Il s'agit d'un système compact opérationnel sur un volume de données relevant de notre domaine, et dont le propos est de donner un exemple concret d'un usage possible de notre formalisation.

Mots-clés :

paraphrase, structures paraphrastiques, traitement automatique de la paraphrase, extraction des structures paraphrastiques

Abstract

Linguistic analysis conducted on our corpus from the ago-food sector shows that the relations between sets of paraphrases can be described as series of textual transformations. To rephrase, an initial lexical substitution starts, then triggers other syntactic, lexical and morphological changes. Replacing one word by one another implies that there is a substitution of a sense by another sense, *a priori*, a similar one. In this perspective, for a pair of paraphrases, it is possible to identify at least two lexical units which, according to us, maintain the "source sense – target sense" relationship. The other main senses in the sentence are considered as being governed by these two senses.

After having described the frequent paraphrasing mechanisms in our corpus, we propose a formalisation which explains the different paraphrasing relationships maintained by the paraphrases between each other. This formalisation has a theoretic and pedagogic purpose which completes the second formalisation which latter forms the core of our work : paraphrase structures formalised as predicate-argument ones. We can say that the first formalisation describes the action, i.e. the paraphrasing mechanisms (and the different paraphrasing relationships), whilst the second formalisation captures the result. In fact, the predicates correspond to "source senses – target senses" and the arguments to "the other senses governed by the predicates" as described above. The proposed paraphrase structures formalisation as predicate-arguments is that which we consider suitable for paraphrase processing.

Finally we have implemented a paraphrase structures extraction system. This

is a compact operational system for the volume of data within our domain, the aim of which is to provide a concrete example of a possible use of our formalisation.

Keywords :

paraphrase, paraphrase structures, paraphrase processing, paraphrase structures extraction

Remerciements

Je tiens à exprimer tout d'abord mes remerciements à Madame le professeur Sylviane CARDEY-GREENFIELD, qui m'a offert l'opportunité de réaliser cette thèse et m'a accordé sa confiance tout au long du chemin et sans qui ce travail n'aurait pas vu le jour. Je lui suis reconnaissant de son encadrement de qualité, son attention et ses précieux conseils.

J'adresse également ma profonde gratitude à Monsieur Henri MADEC et Monsieur Peter GREENFIELD pour leur disponibilité et leur amitié.

Je remercie ma mère, N'Jeff et Miss P. pour leurs encouragements et leur soutien inconditionnel tout au long de mon parcours. Grâce à eux, ce projet a pu être mené à bien.

Enfin, je dois un grand merci à tous mes amis pour avoir partagé des moments précieux tout au long de ces dix années en France. « Vous êtes trop nombreux pour être cités sur une page ».

Table des matières

Introduction **1**

I Théories Linguistiques et Paraphrase **3**

Introduction **4**

1 La théorie sens-texte **5**

1.1 Vue d'ensemble 5

1.2 Les fonctions lexicales et le TAL 10

1.3 Conclusion 15

2 La grammaire guidée par prédicat **17**

2.1 Aperçu global 17

2.2 La PDG et la paraphrase 20

2.3 La PDG : avantages et inconvénients 24

2.4 Conclusion 28

3 Le lexique-grammaire **30**

3.1 Une brève présentation 30

3.2 Les transformations et la paraphrase 32

3.3 Le lexique-grammaire et le TAL 34

3.4 Conclusion 38

Conclusion **40**

II Analyse Linguistique et Formalisation 41

4 La portée de notre travail **42**

4.1 Notre méthodologie 42

4.2 Corpus 44

4.3 Typologie de la paraphrase 48

4.3.1 Exactitude du lien paraphrastique 50

4.3.2 Moyens paraphrastiques impliqués 54

4.3.3 Profondeur du lien paraphrastique 57

5 Analyse linguistique **64**

5.1 Introduction 64

5.2 Sème AIMER 65

5.2.1 X aimer Y 68

5.2.2 X être amateur de Y 70

5.2.3 Y plaire à X ; X être satisfait de Y 73

5.3 Sème INQUIÉTER 78

5.3.1 Y s'inquiéter de X 79

5.3.2 Y avoir une inquiétude sur X 81

5.3.3 Y être inquiet de X 83

5.4 Sème DÉCEVOIR 85

5.4.1 X décevoir Y ; X Y décevoir 87

5.4.2 X être décevant ; Y trouver X décevant 88

5.4.3 Déception de Y vis-à-vis de X ; déception de Y ; Y déception 91

5.4.4 Y être déçu par X 93
5.4.5 Y être déçu ; Y être déçu X (X = Proposition) 95
5.5 Récapitulatif : les idées extraites de l'analyse 96
5.5.1 Structure source - structures dérivées 96
5.5.2 Noyau sémantique de la phrase 97
5.5.3 Un sème = un ensemble de structures paraphrastiques . . . 98
5.5.4 Un modèle de la paraphrase formalisé 99
5.5.5 Extraction des structures paraphrastiques 99
5.6 Conclusion 100

6 Formalisation **102**
6.1 Structures paraphrastiques comme structure source-structure dérivée 102
6.2 Formalisation en vue d'extraction automatique 110
6.2.1 Postulas 110
6.2.2 Formalisation : ensemble de structures paraphrastiques . . . 111

III Implémentation **117**
Introduction 118

7 Extraction des structures paraphrastiques **120**
7.1 Processus d'extraction 120
7.2 Composantes du système 123
7.2.1 Labelgram 123
7.2.2 Extraction des phrases candidates 125
7.2.3 Chunking 127
7.2.4 Nettoyage des chunks 129
7.2.5 Transformation des chunks en structures prédicat-argument 130

8 Traitement point par point **132**

8.1 Module 2 : Extraction des phrases candidates 134

8.2 Module 3 : Chunking 135

8.3 Module 4 : Nettoyage des chunks 136

8.4 Module 5 : Transformation des chunks en structures prédicat-argument 137

8.5 Résultats et discussions 138

Conclusion **141**

Références Bibliographiques **145**

Références 145

Annexes **151**

A Données linguistiques **151**

A.1 Dictionnaire électronique Dicouèbe 151

A.2 Exemples de règles de PDG 163

B Codes du programme en Prolog **166**

B.1 92 phrases tagguées sentences.pl 166

B.2 Liste des dérivés derives.txt 172

B.3 Codes du programme noyau module2.pl 174

B.4 Données de sortie du module 2 candidates.pl 179

B.5 Données de sortie du module 2 candidates.log 182

B.6 Données de sortie du module 2 rejects.pl 187

B.7 Données de sortie du module 2 rejects.log 192

B.8 Codes du programme noyau module3.pl 202
B.9 Règles de chunking (grammaires de groupes de mots) 206
B.10 Données de sortie du module 3 chunked_ sentences.pl 208
B.11 Données de sortie du module 3 chunked_ sentences.log 212
B.12 Codes du programme noyau module4.pl 223
B.13 Dictionnaire forme fléchie - forme canonique flechie_canonique.pl . 228
B.14 Règles de réduction/décomposition 231
B.15 Données de sortie du module 4 chunked_ sentences_cleaned.pl . . 233
B.16 Données de sortie du module 4 chunked_ sentences_cleaned.log . . 238
B.17 Codes du programme noyau module5.pl 252
B.18 Ontologies pour libeller les arguments ontologies.pl 258
B.19 Liste des dérivés pour libeller le prédicat derives.pl 260
B.20 Sortie du module 5 pred_arg_structures.pl 262
B.21 Sortie du module 5 pred_arg_structures.log 271
B.22 Sortie du module 5 final_structures.txt 285

Table des figures

1.1 L'architecture de la théorie sens-texte 6

1.2 Quelques FL appliquées au thaï 9

1.3 Fonction lexicale Labor12 appliquée au thaï 11

2.1 Une des règles de PDG pour l'entrée lexicale *doubt* 19

2.2 Définition de la paraphrase par Schuster (2009, p. 46) 21

3.1 Tables du lexique-grammaire 31

4.1 Exemples de verbatims 45

4.2 Classification des sens 46

5.1 Définition d'*aimer* par Le Petit Robert (2004) 65

5.2 Exemples de phrases incluses dans le sème *aimer* 66

5.3 Regroupement des phrases du sème *aimer* 67

5.4 Equivalence structurelle entre SOV et SVO 68

5.5 Définition de la dérivation sémantique (Polguère et Mel'čuk, 2006) . 70

5.6 Relation Synonymique entre *aimer* et « être amateur de » 71

5.7 Transformation *aimer* $\Longrightarrow$ « être amateur de » 73

5.8 Transformation « X aimer Y » $\Longrightarrow$ « Y plaire à X » 75

5.9 Transformation « Y plaire à X » $\Longrightarrow$ « X être satisfait de Y » . . . 76

5.10 Définition d'*inquiéter* par Le Petit Robert (2004) 78

5.11 Regroupement des phrases du sème *inquiéter* 79

5.12 Transformation « X inquiéter Y » $\Longrightarrow$ « Y s'inquiéter de X » . . . 80

5.13 Transformation « X inquiéter Y » $\Longrightarrow$ « Y avoir une inquiétude sur X » 82

5.14 Transformation « X inquiéter Y » $\Longrightarrow$ « Y être inquiet de X » . . . 84

5.15 Définition de *décevoir* par Le Petit Robert (2004) 85

5.16 Regroupement des phrases du sème *décevoir* 86

5.17 Transformation « X décevoir Y » $\Longrightarrow$ « X être décevant (pour Y) ». 89

5.18 Transformation « X décevoir Y » $\Longrightarrow$ « Y trouver X décevant ». . . 90

5.19 Transformation « X décevoir Y » $\Longrightarrow$ « déception de Y vis-à-vis de X ». 92

5.20 Transformation « X décevoir Y » $\Longrightarrow$ « Y être déçu par X ». 94

6.1 Relations paraphrastiques $\mathbf{R} = \{Identité, R_D, RI_1, RI_2\}$ 104

6.2 Relations paraphrastiques $\mathbf{R} = \{Identité, R_D, RI_1, RI_2\}$ avec exemples 105

6.3 RI_2 : Relation Paraphrastique Indirecte entre les structures paraphrastiques dérivées de la structure source. 108

6.4 Les unités de sens les plus significatifs dans des structures paraphrastiques 111

6.5 Perception de l'ensemble de groupes de sens à un niveau plus abstrait, représentant un ensemble de paraphrases 112

6.6 Groupes de sens comme groupes de prédicat et ses arguments . . . 113

6.7 Phrases de surface possibles des groupes de sens dans la figure 6.6 . 114

6.8 Phrases de surface étiquetées des fonctions prédicat-argument . . . 115

6.9 Formalisation : ensemble de structures paraphrastiques sous forme de prédicat-argument 116

7.1 Processus d'extraction de structures paraphrastiques et ses composantes 122

7.2 Exemples de phrases d'entrée initiales au système d'extraction . . . 124

7.3 Liste de dérivés pour le sème *décevoir* 127

7.4 Exemples de règles de chunking 128

7.5 Règles de transformation des groupes nominaux 129

7.6 Deux grandes catégories d'entités 130

7.7 Exemple d'une structure de sortie finale 131

Liste des tableaux

4.1 Typologie de la paraphrase de J. Milićvić 50
4.2 Typologie de la paraphrase 2 51
4.3 Typologie de la paraphrase 3 53
4.4 Typologie de la paraphrase 4 57
4.5 Typologie de la paraphrase 5 63

Introduction

La paraphrase est un phénomène inhérent à la linguistique, à tel point qu'elle se trouve au cœur de plusieurs théories linguistiques. Certains considèrent que l'objet que les linguistes doivent tâcher de décrire serait les relations paraphrastiques. L'importance de l'étude de la paraphrase devient encore plus probante de nos jours avec le développement rapide et de grande envergure du domaine du traitement automatique des langues (TAL).

Alors que l'étiquetage du corpus et l'analyseur syntaxique préoccupent un grand nombre de linguistes depuis un demi-siècle et que les avancées y sont satisfaisantes, l'ère du TAL passe progressivement de simples traitements du texte à la compréhension automatique. C'est là que la paraphrase a son plus grand intérêt. En effet, le traitement de la paraphrase est primordial dans un bon nombre d'applications comme l'extraction d'informations, les systèmes de question-réponse, la traduction automatique, le résumé automatique de texte, la génération automatique de messages, la compréhension, la reconnaissance automatiques du sens, et l'extraction de connaissances, entre autres. Ces systèmes doivent « comprendre» le langage et savoir si un ensemble de phrases ont le même sens, autrement dit si elles sont paraphrases les unes des autres. Par exemple, il est impensable de concevoir un système de résumé automatique sans qu'il soit capable de détecter les paraphrases. De même, un système de question-réponse ne peut pas s'attendre à ce que les mil-

liers d'utilisateurs posent exactement la même question. Quant à l'extraction des connaissances, un tel système doit prendre en compte les paraphrases linguistiques aussi bien que celles qui sont non-linguistiques (e.g. paraphrases pragmatiques). Dans ce contexte, on essaie par exemple de représenter des connaissances sous un codage unique à partir d'expressions différentes contenues dans le texte (Fluhr & the contributors SAIMSI, 2013). Ces systèmes ont tous besoin d'un module de traitement de la paraphrase.

Ainsi se lèvent plusieurs questions à savoir : quels sont les outils linguistiques disponibles en traitement automatique de la paraphrase ? Certaines théories linguistiques proposent bel et bien des formalisations des règles paraphrastiques mais sont-elles susceptibles d'être adaptées au TAL sur une grande échelle ? Quelles seraient la méthodologie et la formalisation adéquates en vue de la construction des ressources linguistiques pour le traitement automatique de la paraphrase ? Nous allons essayer de répondre à ces questions à travers les trois parties de cette thèse.

Dans un premier temps, nous faisons un état des lieux des théories linguistiques qui mettent en avant la paraphrase et qui proposent des formalisations destinées à son traitement automatique. Nous présentons ces théories une par une et en dégageons les avantages et les inconvénients vis-à-vis du traitement automatique.

La deuxième partie constitue le noyau de cette thèse. Nous y exposons nos travaux d'analyse des données linguistiques tirées d'un corpus du domaine agroalimentaire. En tenant compte des résultats de ces analyses linguistiques, nos réflexions ainsi que les faits décrits dans la partie précédente, nous proposons une nouvelle formalisation des structures paraphrastiques.

En guise d'exemple de la mise en pratique des formalisations proposées, la troisième partie de la thèse est consacrée à l'implémentation d'un système d'extraction des structures paraphrastiques avec une méthodologie innovante.

Première partie

Théories Linguistiques et Paraphrase

Introduction

Dans ce chapitre, nous présentons trois approches différentes de la formalisation des règles paraphrastiques : les fonctions lexicales dans la théorie sens-texte d'Igor Mel'čuk, les règles paraphrastiques dans la Grammaire guidée par prédicat de Jörg Schuster et les transformations (de Zellig A. Harris) telles qu'elles ont été appliquées au français dans le lexique-grammaire de Maurice Gross. Il s'agit des règles formelles censées être susceptibles d'informatisation dans le contexte du traitement automatique des langues.

Nous considérons ces trois méthodes de formalisation en nous situant à la fois dans la linguistique théorique et le traitement automatique des langues. Nous montrons ainsi les avantages et inconvénients majeurs de ces règles paraphrastiques formelles à l'égard du traitement automatique.

Il est important de souligner que nous ne faisons pas une synthèse complète de toutes les approches existantes de la description de la paraphrase telles que les calculs logiques sur le sens (Martin, 1976), les transformations syntaxiques (Z. Harris, 1972 ; Smaby, 1971), l'approche énonciative d'A. Culioli (1976), et de C. Fuchs (1984). En effet, seules les règles paraphrastiques formelles faisant l'objet d'implémentation dans des applications de TAL actuelles nous intéressent. D'autre part, une synthèse complète des approches de la paraphrase ainsi que son histoire ont été faites par C. Fuchs (1980).

Chapitre 1

La théorie sens-texte

1.1 Vue d'ensemble

La théorie sens-texte (TST) d'Igor Mel'čuk (Žolkovskij & Mel'čuk, 1965) est un courant linguistique qui a un impact important sur la linguistique dans le 21ème siècle. Dans la TST, décrire une langue est équivalent à décrire la correspondance entre un ensemble de sens et un ensemble de textes, d'où l'existence d'un modèle de la paraphrase. Suivant cette perspective, on s'intéresse à la question de savoir comment un sens peut être exprimé dans une langue donnée. La figure 1.1 ci-dessous montre l'architecture du modèle sens-texte avec ses 7 couches (composantes) linguistiques.

La paraphrase se situe dès lors au centre de la théorie sens-texte. Nous considérons que la TST propose un des outils linguistiques formalisés le plus complet pour le paraphrasage : les fonctions lexicales (FL). En effet, malgré l'existence d'autres approches de la paraphrase (Culioli, 1990 ; Fuchs, 1980, 1982, 1984 ; Z. Harris, 1972 ; Martin, 1976), les règles de paraphrasage formulées en termes de fonctions lexicales font l'objet d'implémentations dans des systèmes de traitement automa-

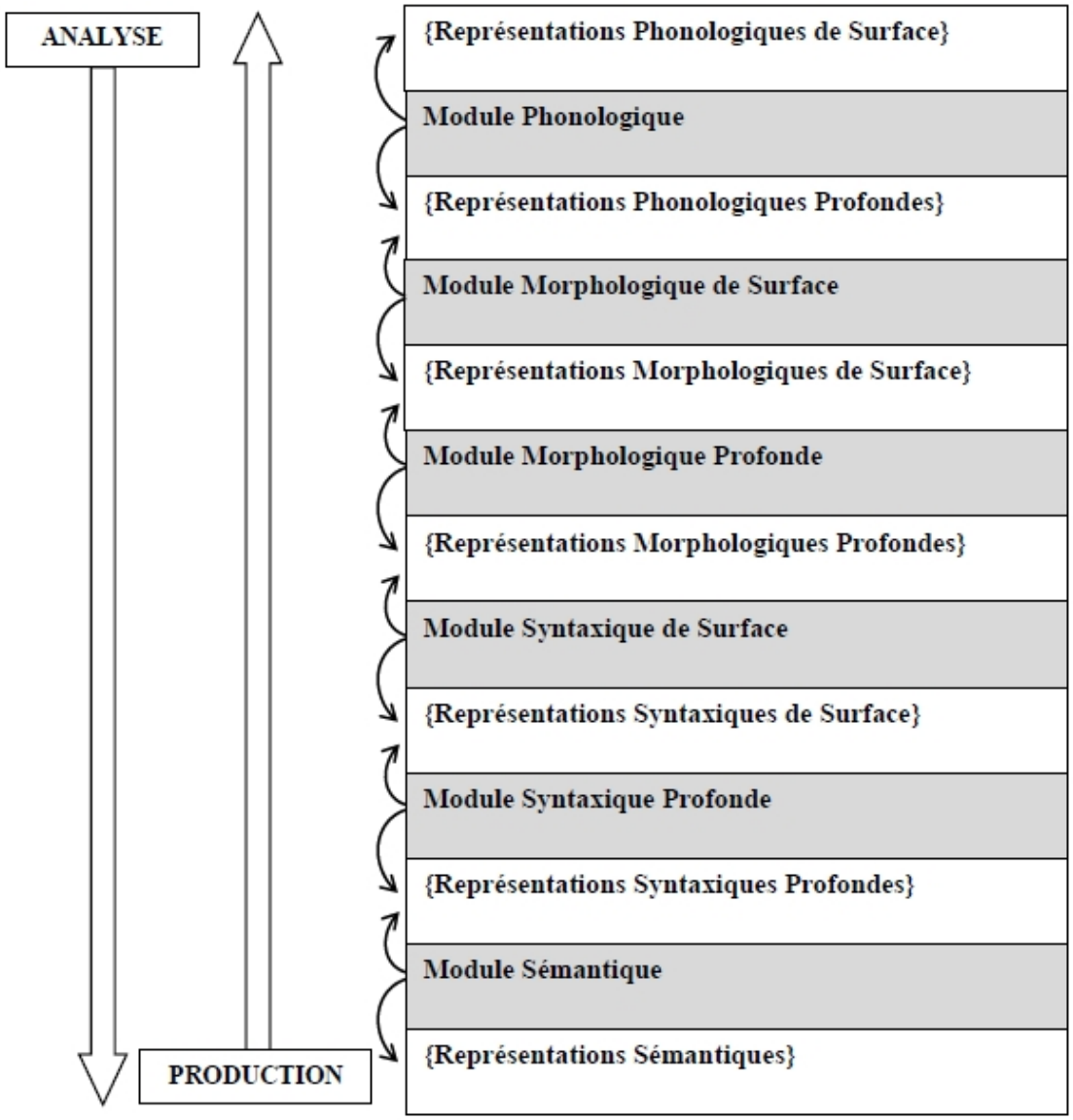


FIGURE 1.1 – L'architecture de la théorie sens-texte

tique des langues, plus précisément dans de nombreux systèmes de traduction automatique (Apresjan et al., 1992 ; Nasr, 1996) et de génération automatique de textes (CoGenTex, 1992 ; Iordanskaja, 1992).

Les fonctions lexicales sont un appareillage formel qui sert à modéliser des relations lexicales telles que les collocations et la dérivation sémantique lexicale.

Prenons un exemple simple de la fonction lexicale Magn.

Étant donné :

- la fonction lexicale Magn, ayant comme sens *intense/beaucoup* et
- les mots suivants *succès, dormir, aimer* ; l'application de la fonction lexicale Magn à ces trois mots donnent :
  - Magn(succès) = grand, franc, vif
  - Magn(dormir) = profondément, comme une souche, à poings fermés
  - Magn(aimer) = à la folie, beaucoup

Les mots *succès, dormir, aimer* sont appelés mot-clé ou argument de la fonction lexicale Magn. Les mots à droite du symbole « = » sont les valeurs ou résultats de l'application de la fonction lexicale sur le mot-clé. Considérons encore quelques exemples plus complexes tirés de (Milićević, 2007, p. 46) (nous modifions la partie valeur de la fonction lexicale pour faciliter la compréhension du lecteur) :

- Oper1(analyse) = effectuer, faire ARTICLE analyse

Oper1(analyse) signifie que le verbe support prend *analyse* comme son complément d'objet. Le numéro 1 dans Oper1 indique que le verbe support prend comme sujet le premier actant syntaxique profond de *analyse*. Pour être simple, un actant d'une lexie [1] correspond à une entité exigée par la lexie. L'actant rend complète la construction d'une phrase sémantiquement pleine. Par exemple, la lexie *manger* demande deux actants : « quelqu'un/quelque chose » comme sujet et « quelque chose » comme complément.

- Func0(analyse) = est en cours, se déroule

Func0 correspond au verbe support qui prend *analyse* comme sujet et 0 dans Func0 marque le complément d'objet direct de *analyse* (dans ce cas, il n'y a pas de complément d'objet direct).

1. Mot simple ou composé, sémantiquement plein

– Labor12(analyse) = soumettre NOM à ARTICLE analyse

Labor_{ij} est verbe support qui prend *analyse* comme son complément d'objet indirect, i dans Labor_{ij} indique son sujet et j, son complément d'objet direct.

Puisque les fonctions lexicales ont largement été étudiées et décrites d'une façon plus complète, nous n'entrerons pas dans le détail car nous ne pourrons mieux faire. Les lecteurs intéressés par les descriptions théoriques détaillées sur les fonctions lexicales peuvent consulter (Milićević, 2007) qui est la mise à jour des règles de paraphrasage lexico-syntaxiques de (Žolkovskij & Mel'čuk, 1965).

Du point de vue linguistique, les fonctions lexicales sont puissantes. En effet, elles permettent avant tout de générer des paraphrases beaucoup plus sophistiquées que de simples transformations syntaxiques. De plus, les FL seraient quasi-universelles, c'est-à-dire qu'il serait possible d'écrire les FL pour la plupart des langues existantes. Il est sans doute impossible de confirmer l'universalité des FL, néanmoins elles ont déjà été appliquées à plusieurs langues, espagnol, anglais, russe, français. Étant locuteur natif du thaï, une langue très éloignée de celles citées plus haut, nous pouvons sans difficultés donner quelques exemples de FL appliquées au thaï (Figure 1.2).

Passons à la question où les FL se situent par rapport au traitement automatique de la paraphrase. C'est dans cette perspective que nous discutons leurs avantages et leurs inconvénients.

thaï	Magn(ความสำเร็จ)	= อันยิ่งใหญ่
français	Magn(succès)	= grand, franc, vif
traduction littérale	Magn(succès)	= grand, énorme, gigantesque
thaï	Magn(หลับ)	= สนิท, เป็นตาย
français	Magn(dormir)	= profondément, comme une souche, à poings fermés
traduction littérale	Magn(dormir)	= nettement, fermement, comme un mort
thaï	Oper1(การวิเคราะห์)	= ทำการวิเคราะห์
français	Oper1(analyse)	= effectuer, faire ARTICLE analyse
traduction littérale	Oper1(analyse)	= faire analyse

FIGURE 1.2 – Quelques FL appliquées au thaï

1.2 Les fonctions lexicales et le TAL

Nous avons brièvement présenté la théorie sens-texte et ses outils pour le paraphrasage, les fonctions lexicales. Nous proposons dans cette section des remarques sur les FL du point du vue du traitement automatique, plus précisément leurs avantages et inconvénients.

Les avantages des fonctions lexicales :

- paraphrases sophistiquées, variées et stylisées
- quasi-universelles

Les inconvénients des fonctions lexicales :

- uniquement pour le paraphrasage (génération de paraphrases)
- difficultés dans le développement des ressources de FL

Développons point par point en commençant par les points positifs. D'abord, les FL permettent d'obtenir des paraphrases sophistiquées plus complexes que celles issues de simples transformations syntaxiques active-passive, réduction/effacement des conjonctions de coordination entre autres. Considérons quelques couples de paraphrases exprimés en termes de FL (Mel'čuk et al., 1992, p. 42).

1. (a) Le général [=I] Wanner a donné [Oper1(L)] cet ordre [L=II]
   Le général Wanner a donné cet ordre.

   (b) Cet ordre [L=I] vient [Func1(L)] du général [=II] Wanner
   Cet ordre vient du général Wanner.

2. (a) Elle [=I] effectue [Oper1(L)] une analyse [L=II] standard des échantillons [II=(analyse)]
   Elle effectue une analyse standard des échantillons.

(b) Elle [=I] soumet [Labor12] les échantillons [=II] à une analyse [L=III] standard

Elle soumet les échantillons à une analyse standard.

De nombreux exemples de FL ont été illustrés et décrits dans (Milićević, 2007) auquel nous nous référons souvent car il s'agit de l'ouvrage le plus récent qui traite exclusivement des systèmes de paraphrasage dans le cadre de la théorie sens-texte. Les fonctions lexicales représentent de ce fait un cadre théorique de paraphrasage très intéressant pour des systèmes de TAL telles que la génération automatique de texte et la traduction automatique qui visent des phrases de qualité en production.

Le second avantage des FL est qu'elles sont quasi-universelles, donc constituent un modèle théorique puissant de paraphrasage. Les FL peuvent par exemple être très utiles dans des systèmes de traduction automatique multilingue. Ainsi, il est par exemple possible de définir une fonction lexicale avec plusieurs valeurs et chaque valeur a un correspondant dans une langue cible. En cas de lacune lexicale dans la langue cible, on recourra à une fonction lexicale équivalente à la valeur manquante. Ainsi, *valoriser* n'a pas de correspondance verbale en thaï, mais le synonyme de *valoriser*, « mettre en valeur » (résultat de la fonction lexicale Labor12(valeur)) a bien son équivalent en thaï dont la traduction littérale est « accorder une valeur à » (Figure 1.3).

thaï	Labor12(คุณค่า) = ให้คุณค่าแก่
français	Labor12(*valeur*) = mettre en valeur
traduction littérale	Labor12(*valeur*) = accorder une valeur à

FIGURE 1.3 – Fonction lexicale Labor12 appliquée au thaï

Passons aux désavantages majeurs des FL. Premièrement, les FL peuvent convenablement s'utiliser en génération automatique de paraphrases mais elles pourraient difficilement s'appliquer à la reconnaissance de paraphrases. Dans le cas

de la génération, le système prend comme donnée d'entrée une phrase et produit un ensemble de paraphrases dont le cardinal n'est pas défini. Il est alors facile d'appliquer des règles de paraphrasage comme suite.

Étant donné la phrase d'entrée « j'aime vos produits » :

1. (a) Syn(aimer) = adorer *# synonyme*
   (b) Conv21(aimer) = plaire *# par exemple, vendre - acheter*
2. (a) S1(adorer) = adorateur ; « être adorateur de » si *adorer* est verbe principal *# correspond au nom ayant comme sens celui qui adore*

Processus de génération simplifié :

1. localiser le verbe de la phrase d'entrée : j'**aime** vos produits
2. appliquer toutes les règles paraphrastiques disponibles :
   – règle 1(a) : j' Syn(aime) vos produits $\longrightarrow$ j'adore vos produits
   – règle 1(b) : j' Conv21(aime) vos produits $\longrightarrow$ vos produits me plaisent
3. localiser le verbe de la phrase de sortie : j'**adore** vos produits
4. appliquer toutes les règles paraphrastiques disponibles :
   – règle 1(b) : j' S1(adore) vos produits $\longrightarrow$ je suis adorateur de vos produits

Plaçons-nous en mode reconnaissance avec les mêmes règles paraphrastiques. Étant donné un couple de phrases :

phrase1 vos produits me plaisent
phrase2 je suis adorateur de vos produits

Il y au moins deux façons de procéder plus directs, en se basant sur les règles :

solution1

soit essayer d'appliquer les règles paraphrastiques à **phrase1** et voir si l'ensemble des paraphrases produites contient **phrase2** ou vice-versa. En cas

d'échec, on essaie d'appliquer les règles paraphrastiques aux phrases de sorties et ainsi de suite ...

solution2

soit essayer d'appliquer les règles paraphrastiques à **phrase1** et **phrase2** et voir si l'intersection entre l'ensemble des paraphrases de **phrase1** et l'ensemble de paraphrases de **phrase2** n'est pas vide (d'où paraphrase).

Pour prouver la relation paraphrastique entre **phrase1** et **phrase2**, la règle suivante doit être présente : Conv21(plaire) = aimer. Ainsi, suivant la **solution1** ci-dessus, on prend **phrase1** comme phrase d'entrée et y applique la règle Conv21(plaire), d'où :

– vos produits me Conv21(plaisent) $\longrightarrow$ j'aime vos produit

On applique fonction Syn(aimer) à cette dernière phrase, d'où :

– j' Syn(aime) vos produits $\longrightarrow$ j'adore vos produits.

On applique S1(adorer) à cette dernière phrase, d'où :

– j' S1(adore) vos produits $\longrightarrow$ je suis adorateur de vos produits.

On remarque que le processus est complexe, coûteux car l'algorithme a trop de chemins à prendre. En effet, nous n'avons montré ci-dessus qu'un parcours possible alors qu'une règle comporte normalement plusieurs règles paraphrastiques (FL). De plus, l'application récursive des règles paraphrastiques à des phrases de sortie est problématique : le système ne sait pas où s'arrêter et à force de continuer, il finit probablement par appliquer une multitude de règles, et cela sans aucune garantie de pouvoir prouver l'équivalence sémantique avec les deux phrases de départ. En outre, il est difficile de déterminer par quel mot le système commence à appliquer des règles. Ainsi, la phrase « je suis adorateur de vos produits » présentent au

moins deux mots candidats les plus probables : *suis* et *adorateur*, sachant que le mauvais choix entraîne une perte de temps considérable. La mise en garde de (J. Milićević, 2007, p. 7) ci-dessous confirmerait notre thèse.

> Dans ce livre, il sera question de la relation de paraphrase, ainsi que de l'opération de production [= synthèse] de paraphrases. La reconnaissance [= l'analyse] de paraphrases sera laissée hors de notre propos.

Le deuxième désavantage des FL concerne les difficultés dans le développement des ressources de FL nécessaires à un éventuel traitement automatique à une plus grande échelle. Un système de TAL qui implémente les FL nécessite un dictionnaire comprenant les entrées lexicales avec leurs FL possibles ainsi que la valeur de ces dernières. Le Dictionnaire Explicatif et Combinatoire (Mel'čuk, 1999) est un dictionnaire de référence, réalisé dans le cadre de la théorie sens-texte. La base de données électronique du DEC, DiCo en cours de développement à l'Observatoire de linguistique Sens-Texte (OLST) contient 539 entrées lexicales polysémiques (ou vocables selon les auteurs) (source : http ://olst.ling.umontreal.ca/dicouebe/index.php, le 07 février 2012 à 16h01). Le DiCouèbe (cf. annexe A.1) est la version en ligne du DiCo, consultable à l'adresse http ://olst.ling.umontreal.ca/dicouebe/main.php. On remarque que le nombre de vocables total est assez limité malgré une vingtaine d'années de travaux. Cela est dû probablement au nombre élevé de FL pour chaque entrée lexicale.

Par ailleurs, recenser, formaliser et déterminer les FL pour un vocable n'est pas une tâche simple. Mihajlovska (2011) a relevé les difficultés que pose l'annotation des fonctions lexicales dans un corpus espagnol. De même, Milićević (2007, p. 166) donne des remarques sur les améliorations nécessaires sur les travaux lexicographiques :

> [...] Or, pour que les définitions (des entrées lexicales dans le DEC) puissent s'utiliser de façon efficace dans le paraphrasage, elles doivent être bien structurées [...] Bien que cette démarche (la démarche actuelle pour définir les entrées lexicales) soit claire en principe, nombre de questions de détail demeure.

Certes les fonctions lexicales constituent un outil de paraphrasage très puissant, cependant leurs ressources doit être très volumineuses pour pouvoir s'utiliser dans le traitement automatique de paraphrases du domaine général.

1.3 Conclusion

Nous avons rapidement présenté la théorie sens-texte dont la paraphrase représente un des éléments porteurs. Ensuite, nous avons abordé les fonctions lexicales, outils de paraphrasage, l'une des découvertes les plus importantes dans la TST, et que nous avons considérée comme un des meilleurs moyens linguistiques de paraphrasage existant. Nous avons donné quelques exemples de FL et avons, par la suite, discuté des avantages et inconvénients du modèle du point de vue du traitement automatique de la paraphrase. Les FL représentent ainsi une possibilité de générer des paraphrases sophistiquées dans le sens où elles vont au-delà de simples transformations syntaxiques. De plus, le concept de FL est quasi-universel car elles ne dépendent pas d'une langue spécifique.

Cependant, les FL présentent deux inconvénients majeurs en traitement automatique de la paraphrase. D'abord, ces outils de paraphrasage fonctionnent bien dans le sens de la production (génération de paraphrases) mais ne conviennent pas à l'analyse (reconnaissance de paraphrases). Le deuxième porte sur des difficultés dans le développement des ressources de FL : cette tâche exige un temps de travail

long en raison d'une part du nombre élevé de FL et de l'autre de leur annotation ainsi que de la définition des entrées lexicales qui n'est pas définitive.

Chapitre 2

La grammaire guidée par prédicat

2.1 Aperçu global

La grammaire guidée par prédicat (Predicate Driven Grammar abrégé en PDG) de Schuster (2009) est, comme son nom l'indique, une grammaire formelle qui s'inspire de la théorie des prédicats linguistiques. Il s'agit ici du prédicat dans le sens de « mot sémantiquement plein » plutôt que de prédicat dans le sens d'une opposition sujet-prédicat comme dans la logique d'Aristote. Le prédicat ne se restreint pas au verbe, d'autres parties du discours comme l'adjectif, l'adverbe, le nom, voire certaines conjonctions et prépositions, peuvent avoir une valeur de prédicat.

L'auteur propose non seulement une grammaire formelle destinée au traitement automatique, il expose à travers son livre des réflexions intéressantes concernant cette grammaire. On peut aussi dire que la théorie linguistique et la grammaire formelle PDG se répondent. La démarche ressemble à celle que l'on retrouve chez Chomsky, pourtant les réflexions sur la linguistique chez Schuster s'avèrent plus poussées.

Les travaux de Schuster ont pour origine en large partie la théorie sens-texte qui

modélise la langue en termes de relations entre le sens (représentation mentale) et le texte (produit langagier : texte, parole). La tâche de la linguistique dans cette perspective consiste alors à décrire la relation entre le sens et le texte. Ainsi, Schuster déclare :

> I would say that the facts which linguistics has to find must be such that they contribute to a description of a text-meaning relation and that a linguistic theory can be said to be correct and complete, if it returns for each input text t the set of meanings which t can have and for each meaning m the set of texts which express m. Equivalently, a linguistic theory can be said to be correct and complete, if it returns for each input text t the set of its paraphrases.
>
> **Traduction :** Je dirais que les faits que la linguistique tâche de trouver doivent être tels qu'ils contribuent à la description de la relation sens-texte et qu'une théorie linguistique peut être dite correcte et complète, si elle retourne, pour chaque texte d'entrée « t » l'ensemble des sens que « t » peut avoir et pour chaque sens « s » l'ensemble des textes qui expriment « s ». Parallèlement, une théorie linguistique peut être dite correcte et complète, si elle retourne pour chaque texte « t » l'ensemble de ses paraphrases.

Certes, les postulats de la théorie de Schuster tournent autour de la théorie sens-texte, il n'en va pas de même pour sa grammaire formelle PDG qui présente un mélange de concepts venant de plusieurs écoles. En effet, on retrouve dans les descriptions de la PDG : la grammaire hors contexte et la notion de syntagme, qui sont probablement venues des théories de N. Chomsky (1957, 1965) ; les fonctions lexicales de la théorie sens-texte ; les transformations de Z. S. Harris (1970)

pour définir la relation paraphrastique ; le lexique-grammaire chez Gross (1975). Considérons la règle de PDG suivante :

arity	2
syntax	s $\Longrightarrow$ vp–have -n-nf s
meaning	doubt[s=oper](he,love(she,he))
text	He has doubts she loves him
owners	doubt

FIGURE 2.1 – Une des règles de PDG pour l'entrée lexicale *doubt*

Toutes les règles de PDG codées sous forme de cinq sous-règles sont composées de :

1. arity (arité ou le nombre d'arguments du prédicat), ici 2 arguments
2. syntax (syntaxe indiquant la structure de la phrase), la phrase est composée d' :
   - un syntagme verbal vp–have
   - un prédicat au pluriel -n-nf (-n indique la place du prédicat *doubt*, -nf pluriel)
   - une phrase s
3. meaning (sens sous forme d'une extension de la logique des prédicats)
4. text (text correspond à la phrase canonique de cette règle)
5. owners (propriétaires ou prédicats).

Faisons une décomposition des concepts présents dans la règle de grammaire suivante. La grammaire hors contexte et la notion de syntagme se trouvent dans la partie syntax qui présente la règle de réécriture « *s* $\Longrightarrow$ *vp–have -n-nf s* ». Dans la partie meaning, *doubt[s=oper]* correspond à la fonction lexicale Oper de la théorie sens-texte dont l'application à *doubt* donne « *have doubts* ». Une telle

règle de grammaire, vue dans son ensemble, s'inspire du lexique-grammaire, c'est-à-dire qu'on y trouve une règle locale. Une règle locale s'oppose à une règle globale dans la mesure où cette dernière se fonde sur une partie du discours et est censée s'appliquer à un certain nombre de mots, par exemple « $s \rightarrow$ *nom verbe-intransitif nom* », alors qu'une règle locale présuppose une syntaxe et une sémantique propres à un mot, à titre d'exemple la règle de grammaire pour le mot *doubt* ci-dessus. Les inconvénients des règles de grammaires dites globales ont été démontrés par M. Gross (1979). Enfin, Schuster modélise la relation paraphrastique à l'aide de transformations que nous expliquerons dans le prochain chapitre.

Tout mélange complexe de concepts qu'il soit, la PDG se démarque des autres grammaires formelles par son aspect non modulaire. L'auteur pense qu'une règle de grammaire ne doit pas être modulaire, autrement dit qu'elle doit contenir de l'information morphosyntaxique et sémantique car ce ne serait pas approprié de procéder d'une manière séquentielle à l'analyse de différents niveaux linguistiques. La non-modularité se voit ainsi opposée à la plupart des théories linguistiques, par exemple, la théorie sens-texte où 7 strates ont été définies ou comme chez Chomsky et Tesnière (1959) qui séparent la syntaxe et la sémantique.

Nous ne prétendons pas faire une présentation exhaustive de la PDG même si plusieurs éléments nous paraissent intéressants. Passons au sujet qui nous intéresse le plus, à savoir la paraphrase dans la PDG.

2.2 La PDG et la paraphrase

Dans la section précédente, nous avons vu que la paraphrase est au cœur de la PDG. Il est important de préciser que Schuster travaille uniquement sur la paraphrase linguistique (il l'appelle *grammatical paraphrase*), c'est-à-dire sur la

relation paraphrastique entre les phrases pouvant être modélisée à l'aide de règles linguistiques. Il laisse de côté la paraphrase pragmatique (ou « lexical paraphrase » selon l'auteur) qui implique, pour établir le lien paraphrastique entre les phrases, des connaissances encyclopédiques, et des situations d'énonciation. Par exemple :

– M. Chirac est comblé de bonheur après avoir pris sa retraite.
– Le 22ème président de la république française est comblé de bonheur après avoir pris sa retraite.

Aucun moyen linguistique proprement dit ne permet d'établir un lien d'équivalence entre « M. Chirac » et « Le 22ème président de la république française », seule la connaissance encyclopédique et/ou la situation d'énonciation le rendent possible.

Nous avons dit plus haut que Schuster modélise la relation paraphrastique au moyen de transformations. Considérons la définition de la paraphrase suivante (Schuster, 2009, p. 46).

Let s1 and s2 be sentences :

1. if there is a transformation t such that t(s1) = s2, then s1 and s2 are paraphrases of degree 1
2. if there is a chain of transformations t1, t2, ..., ti such that t1(t2(...(ti(s1))...) = s2, then s1 and s2 are paraphrases of degree i
3. if s1 has the immediate constituents a1, a2, ..., an and s2 has the immediate constituents b1, b2, ..., bn and a1 and b1 are paraphrases of degree d1 and a2 and b2 are paraphrases of degree d2 and ... and an and bn are paraphrases of degree dn ; then s1 and s2 are paraphrases of degree d1 + d2 + d3 + dn.

FIGURE 2.2 – Définition de la paraphrase par Schuster (2009, p. 46)

En quelques mots, il s'agit de prouver une relation paraphrastique existant entre deux phrases, en démontrant qu'il y a des transformations qui permettent de

paraphraser une phrase source en une autre phrase et le degré de paraphrase s'obtient en faisant la somme des transformations appliquées. Cette définition intensionnelle est concise et précise. Or, l'application d'un tel concept dans un contexte réel de TAL (par exemple la reconnaissance automatique de la paraphrase) paraît très problématique. En effet, il est sans doute difficile de déterminer l'ordre d'applications des transformations pour pouvoir mettre en évidence la relation paraphrastique. Prenons l'exemple suivant, soit deux phrases :

– J'aime vos produits.
– Je suis admirateur de vos produits.

Comment procède-t-on à une mise en relation paraphrastique de ces deux phrases ? Un fait déjà indéniable est que cette tâche est au-delà des transformations syntaxiques. En effet, les transformations se limitent normalement au mécanisme morpho-syntaxique, par exemple :

pronominalisation	j'aime ces produits. → je les aime.
passivation	Jean a mangé une pomme. → ?la pomme a été mangée par Jean.
réduction	Marie avait apporté un gateau et Paul a mangé le gateau. → Paul a mangé le gateau que Marie avait apporté.

Il est vrai que l'auteur définit le couple de paraphrases au moyen de transformations mais celles-ci n'ont pas grande importance dans la PDG car son mécanisme de paraphrasage repose principalement sur les fonctions lexicales.

Les fonctions lexicales sont issues de la théorie sens-texte. Une fonction lexicale complexe S0(Syn(aimer)) peut mettre en évidence la relation paraphrastique entre les deux phrases ci-dessus. Ainsi, la fonction lexicale synonyme Syn(aimer) donne *admirer*, ensuite la fonction lexicale « premier actant sémantique » S0(admirer)

donne *admirateur*, enfin interviennent l'ajout du verbe *être* et la préposition de enclenchés par la fonction lexicale S0. On établit alors la relation paraphrastique entre :

– J'aime vos produits.
– Je S0(Syn(aimer)) vos produits. = Je suis admirateur de vos produits.

Si notre problème est résolu facilement, c'est parce que c'est l'homme qui fait le travail d'analyse. Mettons-nous maintenant à la place d'une machine, quel algorithme doit-on appliquer ? Par quel mot de la phrase commence-t-on à appliquer les fonctions lexicales ? Comment identifie-t-on le groupe de mots sur lequel les fonctions lexicales s'appliquent ? Pour chaque mot ou chaque groupe de mots, quelles fonctions lexicales seront appliquées ? Quelles sont les fonctions lexicales et leurs valeurs associées dans un dictionnaire ?

Pour conclure, la définition de la paraphrase en termes de transformations et les fonctions lexicales sont très utiles comme outils de description théorique de la paraphrase. Par ailleurs, ces outils fonctionnent dans des systèmes de génération de paraphrases (traduction, génération de textes). Cependant, leur application dans le sens de l'analyse (la reconnaissance ou extraction automatique de la paraphrase) reste un défi.

Il n'y a d'emblée pas de discussions approfondies sur le mécanisme de paraphrasage, nous pouvons considérer les exemples de règles de PDG données dans l'ouvrage cité pour essayer de décrire le fonctionnement de ces règles paraphrastiques. Nous allons le faire maintenant.

2.3 La PDG : avantages et inconvénients

Nous donnons dans cette section des exemples de règles de PDG. Nous ferons ensuite le point sur des avantages et des inconvénients de la PDG dans le cas du traitement automatique des langues et particulièrement le traitement automatique de la paraphrase. Avant d'observer des exemples de règles de PDG tirées du (Schuster, 2009) dans l'annexe A.2, quelques remarques s'imposent :

- selon l'auteur, ce sont des règles d'analyse et non de génération ;
- nous avons corrigé des fautes d'orthographe, par exemple love(se,he) devient love(she,he) ;
- ces règles ont le même prédicat ou « propriétaire de la règle » (owner) ;
- les numéros correspondent exactement à ceux de l'ouvrage.

Listons les points forts et les points faibles de ces règles. Totalement arbitraires et personnelles, ces points de vue nous appartiennent.

Les bons côtés de telles règles de grammaire :

- les règles de PDG sont locales ;
- elles offrent une possibilité d'analyse syntaxique et sémantique en parallèle ;
- la PDG représente un cadre de travail réaliste à l'égard du TAL.

L'inconvénient :

- les règles de grammaires paraissent trop locales.

Des contradictions apparaissent entre les deux aspects. Nous allons éclaircir le problème point par point en commençant par les points positifs. Premièrement, cette grammaire est une grammaire dite locale, c'est à dire qu'elle présuppose les propriétés sémantico-syntaxiques spécifiques d'une lexie (mot sémantiquement plein). La puissance générative de cette grammaire est par conséquent faible car

elle a tendance à n'accepter que les phrases à la fois syntaxiquement et sémantiquement correctes (contrairement aux grammaires basées sur les parties du discours). Concrètement, au moment de l'analyse automatique, si la phrase d'entrée est prise en compte par une des règles de grammaire, cette phrase aura éventuellement une seule représentation syntaxique ainsi qu'une représentation sémantique correspondante non ambiguë.

Deuxièmement, les règles de PDG offrent une possibilité d'analyse syntaxique et sémantique en parallèle. En effet, nous l'avons dit plus haut que chaque règle de grammaire a sa propre représentation sémantique. Ainsi, lorsqu'une analyse syntaxique s'applique, la représentation sémantique de la phrase s'applique aussi. Pour faire un calcul sur la relation paraphrastique entre plusieurs représentations sémantiques, il suffit d'enlever toutes les parties « fonctions lexicales » de ces formules, puis de les comparer : si elles sont identiques, on a affaire à la paraphrase. Ainsi, la partie meaning des phrases 85-92 dans l'annexe A.2 est exactement la même après en avoir retiré toutes les fonctions lexicales qui se trouvent entre crochets, d'où magn(doubt, he(love(she, he))). Cette simplicité et cette efficacité qu'offre la PDG est certainement l'un des outils les plus intéressants pour le traitement de la paraphrase, notamment pour la reconnaissance automatique.

Troisièmement, la PDG offre un cadre de travail réaliste. Par réaliste, nous voulons dire que ces règles de grammaire ont été bien formalisées et de ce fait peuvent être implémentées. Il ne s'agit pas d'une simple description théorique car toutes les composantes d'une règle de PDG sont observables : un numéro indiquant le nombre d'arguments du prédicat ; la structure de la phrase sous forme de grammaire hors-contexte ; une extension de la logique des prédicats comme représentation sémantique ; le texte canonique de la règle ; le(s) prédicat(s) qui possède(s) la règle. Ces cinq éléments peuvent facilement être traités par un système informatique.

Passons maintenant au désavantage majeur de la PDG. Cela concerne son caractère local que nous avons classé plus haut parmi les points positifs. Nous n'avons surtout pas l'intention d'introduire un paradoxe. En effet, nous pensons qu'une règle de grammaire doit être locale pour, d'une part, tenir compte du fait qu'une lexie possède sa propre syntaxe et sémantique, de l'autre pour assurer la puissance générative la plus faible possible (n'accepter et produire que les phrases à la fois syntaxiquement et sémantiquement correctes). Or, si une grammaire est trop locale, elle sera difficilement opérationnelle, même dans un domaine restreint car la phase de rédaction des règles de grammaire prendra un temps considérable. Voici quelques conséquences non souhaitables de cette grammaire.

D'abord, certaines règles présentent des redondances qu'on pourrait éviter. Ainsi, parmi les phrases 85–92 ci-dessous, certains couples de paraphrases ne se distinguent que par une seule lexie *seriously* vs *severely* ou par la présence ou l'absence de la conjonction de subordination *that*.

85. He doubts gravely she loves him.
86. He doubts gravely that she loves him.
87. He doubts seriously her love for him.
88. He doubts seriously she loves him.
89. He doubts seriously that she loves him.
90. He doubts seriously her love for him.
91. He doubts severely she loves him.
92. He doubts severely that she loves him.

Ensuite, sans automatisation, la rédaction des règles PDG exigerait un temps de développement et une quantité de travail importants. En effet, dans chaque règle de PDG la représentation sémantique est assez complexe. L'incorporation

d'une telle composante dans la règle s'avère couteux à moins qu'il n'y ait un algorithme qui permette la génération automatique de la partie *meaning* à partir de *syntax*. De surcroît, le rédacteur des règles doit constamment s'assurer que la représentation sémantique, une fois les fonctions lexicales omises, reste identique pour toutes les paraphrases. Sans automatisation et abstraction, à un certain degré bien évidemment, une grammaire locale sera synonyme de chiffrement des phrases, des paraphrases, des structures ainsi que de leur sens.

Par ailleurs, construire un ensemble des prédicats constituant la grammaire n'est pas une tâche simple. En effet, les règles de grammaires sont guidées par le prédicat, équivalent d'une lexie. Il s'ensuit qu'il faut attribuer des règles à chaque lexie pour que la grammaire soit complète et opérationnelle. En d'autres termes, on doit écrire les règles pour un très grand nombre de mots comme mine/yours/his, really/possibly, superior, book/dog, love/admiration, war/party, love/like/see, nice, etc.

Prenons la phrase simple composée de neuf mots « Je suis vraiment grand amateur de vos excellents produits ». Disons que parmi ces neuf mots, les sept suivants sont des mots-pleins : *suis, vraiment, grand, amateur, vos, excellents, produits.* Pour analyser cette phrase, au moins sept règles de PDG sont probablement nécessaires et cela implique sept représentations syntaxiques et sept représentations sémantiques.

Ensuite, les sept représentations sémantiques se combinent d'une certaine façon pour former le sens de la phrase. Il en va de même pour les sept structures qui constituent la représentation syntaxique de la phrase. Nulle part Schuster n'a fourni l'algorithme d'application des règles, ceci est dû probablement au fait que la construction des règles de PDG est encore à son premier stade de développement.

Quoi qu'il en soit, trop de détails donnent souvent lieu à des complexités. Nous pensons qu'il n'est pas pratique d'écrire les règles pour un très grand nombre de lexies telles que *grand, vos, excellents, vraiment*. Certes ils sont importants mais il vaut peut-être mieux les regrouper par exemple en des classes avec leurs propriétés sémantico-syntaxiques spécifiques.

Ce qui va, selon nous, en faveur d'une généralisation raisonnable d'une grammaire locale doit être justifié, argumenté de façon systématique et méthodique. Nous le ferons dans la partie formalisation. Pour l'instant, il suffit de dire qu'une grammaire locale a besoin d'une certaine abstraction pour éviter l'explosion de règles.

2.4 Conclusion

La grammaire guidée par prédicats ou Predicate Driven Grammar (PDG) fait partie des grammaires formelles lexicalisées dans le sens où, pour un mot donné, elle présuppose des propriétés sémantico-syntaxiques spécifiques : chaque règle est composée du mot-porteur muni de sa syntaxe et de sa sémantique. Les grands postulats qui sous-tendent cette grammaire s'inspirent de la théorie sens-texte d'Igor Mel'čuk. Par conséquent, la paraphrase occupe une place centrale dans la PDG.

Nous avons d'abord donné quelques exemples de règles de PDG et avons expliqué comment la relation paraphrastique peut être mise en évidence. Ensuite, nous avons discuté les points forts de cette grammaire : les règles sont locales, d'où la puissance générative faible et le peu d'ambiguïté ; la représentation sémantique permet le calcul paraphrastique de façon simple et efficace ; toutes les composantes d'une règle de PDG sont susceptibles d'implémentation. Cependant, nous considérons que les règles de PDG sont trop locales. Le manque d'abstraction soulève

quelques inconvénients : règles redondantes, le nombre de règles trop important et le temps de développement trop long.

Chapitre 3

Le lexique-grammaire

3.1 Une brève présentation

Le terme « lexique-grammaire » que nous abordons ici appartient au linguiste français Maurice Gross (1968, 1975, 1977). Les travaux de Maurice Gross attirent notre attention d'un côté par leur lien avec la paraphrase (transformations) et de l'autre par la méthode d'analyse des données linguistiques. Nous développons le premier point dans la section suivante et y donnons des exemples de transformations paraphrastiques. Quant à la méthode d'analyse de données linguistiques adoptée par M. Gross, nous la qualifions de rigoureuse en raison de son caractère empirique.

À la différence de certains linguistes qui proposent des théories linguistiques, soi-disant universelles, en s'appuyant sur leur compétence, leur intuition ou des exemples courants, M. Gross ne fait qu'analyser des données linguistiques, exposer les résultats et en tirer les conclusions. Notre méthode de travail s'inspire en partie des travaux de M. Gross des années 75, (Gross, 1975). On pourra s'en rendre compte en faisant un survol de la partie Analyse linguistique.

M. Gross étudie les propriétés syntaxiques de verbes du français dans le but d'expliquer le rôle que joue la syntaxe dans la compréhension du langage. Les résultats de ses travaux sont des tables dites « tables de lexique-grammaire ». Une table contient un ensemble de verbes qui ont des propriétés syntaxiques similaires. On y trouve pour chaque entrée lexicale : la construction de la phrase, la nature des noms en fonction sujet et compléments, quelques transformations possibles. La figure 3.1 ci-dessous donne un exemple de tables de lexique-grammaire (Gross, 1975, p. 236).

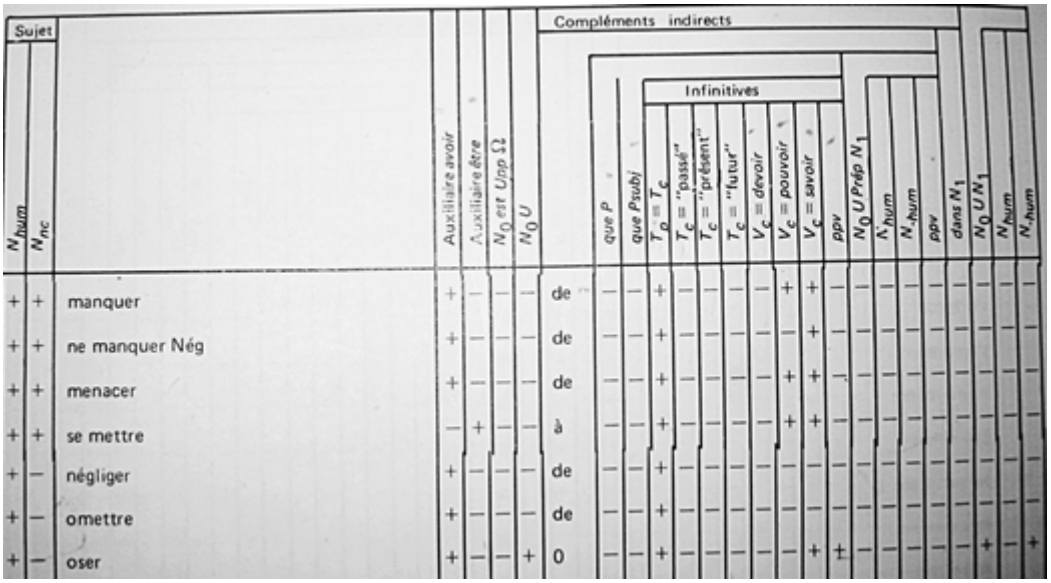


FIGURE 3.1 – Tables du lexique-grammaire

Il y a 3 000 entrées réparties dans 19 tables. Notons que ces 3 000 entrées ne représentent que 2 000 classes. La classe est ainsi définie par M. Gross (1975, p. 214) comme : « Deux éléments (i.e. deux entrées) appartiennent à la même classe lorsqu'ils possèdent les mêmes propriétés syntaxiques. ».

3.2 Les transformations et la paraphrase

Le linguiste américain Zellig S. Harris (1960, 1976, 1981) est le premier à parler de la transformation qui est en quelque sorte un processus consistant à générer à partir d'une phrase noyau d'autres phrases possibles en appliquant une série d'opérations telles que réduction, ajout, suppression. Nous nous intéressons aux transformations appliquées au français, telles qu'elles ont été utilisées par Maurice Gross, à travers ses ouvrages. Il est important de préciser que nous ne parlons pas des transformations selon le modèle génératif de Noam Chomsky (1965).

M. Gross considère la grammaire transformationnelle comme un dispositif expérimental ayant pour but de découvrir et de décrire les contraintes syntaxiques. Les transformations peuvent opérer au niveau du mot, de groupes de mots, d'une phrase ou de plusieurs phrases. Les séquences transformées peuvent être ou non des paraphrases de la phrase source. La notion de paraphrase ne fait pas partie des éléments noyaux de sa recherche, malgré des paraphrases omniprésentes dans ses ouvrages.

Considérons quelques exemples de transformations données par M. Gross (1975). Nos commentaires sur les exemples ne peuvent pas être considérés comme critiques puisque l'auteur n'avait pas l'intention d'approfondir la notion de paraphrase. Nous avons également procédé à des modifications minimes sur certains exemples pour en faciliter la lecture.

Les réductions de prépositions

- Paul est content de ce que Marie vienne. → Paul est content que Marie vienne.
- Paul débat avec Jean de ce que tu sois autorisé à venir. → Paul débat avec Jean que tu sois autorisé à venir.

Pronominalisation

- Paul veut que Jean vienne. → Paul veut ceci. → Paul le veut.
- Il y a des raisons à ce que Paul vienne. → il y a là des raisons.
- Paul le traite comme s'il était un enfant. → Paul le traite ainsi.
- Paul aime Marie. → Paul l'aime.

On peut considérer les transformées ci-dessus comme des paraphrases de la phrase source. Or, dans le contexte du TAL, il est très difficile d'établir la relation d'équivalence sémantique entre « que Jean vienne » et *le*. Même si toutes les phrases de type « X veut que Phrase » peut être paraphrasées en « X le veut », la valeur de *le* ne peut être obtenu que par une analyse de coréférence fiable.

Passif

- Paul a mangé la soupe.→ La soupe a été mangée par Paul.
- Paul a mangé la soupe. → La soupe a été mangée (passif sans agent).
- Tout le monde a reconnu Marie. → Marie a été reconnue de tout le monde.
- Paul a reçu cette garantie formelle. → Cette garantie formelle a été reçue par Paul.
- Paul a rempli le verre d'un liquide rouge. → Le verre a été rempli par Paul d'un liquide rouge.

Extraposition

- Que nous sommes tous d'accord découle de ce que nous avons réfléchi. → Il découle de ce que nous avons réfléchi que nous sommes tous d'accord.
- (De) prendre cette décision appartient à Paul. → Il appartient à Paul de prendre cette décision.

Permutations

- Paul est le chef → Le chef est Paul.
- Un certain nombre d'ennuis résulteront de ta décision → De ta décision résulteront un certain nombre d'ennuis.

– Des primes s'ajouteront à ce salaire → à ce salaire s'ajouteront des primes.

Les exemples ci-dessus montrent que certaines transformations sont en fait des règles lexico-syntaxiques de paraphrasage. Gross y fournit aussi des restrictions sur la nature du lexique, les étapes de transformations et les exceptions pour la plupart des transformations. La formalisation des règles de transformation est assez simple puisqu'une transformation représente une règle de réécriture telle que :

EXEMPLES de REGLES	**Commentaire**
N0 V Prép N1 → Prép N1 V N0	Permutation : Des ennuis résulteront de ta décision. → De ta décision résulteront des ennuis.
N0 V N1 → N1 est Vpp de N0	Passif
N0 V N1 → N1 se V de N0	Forme réflexive : Cette affaire choque Marie. → Marie se choque de cette affaire.
N0 V N1 → N1 est Vpp de N0	Ce produit déçoit Paul. → Paul est déçu de ce produit.

où N0 correspond au sujet de la phrase, N1 complément d'objet direct, Vpp participe passé du verbe V.

3.3 Le lexique-grammaire et le TAL

Jusqu'à présent, les tables du lexique-grammaire constituent une ressource lexicale intéressante pour des analyseurs syntaxiques du français (Gardent, Guillaume, Falk, & Perrier, 2005 ; Laporte, 2010) puisque sa dernière version (http :// infolingu.univ-mlv.fr/) contient :

– 13 872 entrées verbales, dont 5 738 distinctes

– 14 271 entrées pour les noms prédicatifs, dont 10 112 entrées distinctes

- 39 628 entrées pour les expressions figées verbales et adjectivales, dont 38 658 distinctes
- 10 488 entrées adverbiales, dont 9 326 distinctes.

En ce qui concerne la paraphrase, l'auteur fournit simplement un ensemble limité de transformations canoniques pour chaque entrée de la table. Comme M. Gross ne tend pas à proposer un système de paraphrasage, il n'y a pas de définition systématique de règles paraphrastiques qui pourraient être directement adaptées au traitement automatique de la paraphrase malgré l'idée de l'invariance morphémique entre deux phrases liées par une transformation (les deux phrases ont quasiment tous les mêmes mots-pleins, d'où probablement le même sens).

En revanche, nous partageons deux points de vue intéressants de l'auteur sur la méthode de recherche en linguistique et sur l'analyse automatique de phrases qui pourrait s'appliquer aux paraphrases.

Premièrement, M. Gross est convaincu qu'il est indispensable d'analyser et de décrire le langage comme tel sans recourir à tout prix à des appareils formels et à des abstractions qui se forcent à rendre compte des phénomènes linguistiques les plus tordus. En effet, il s'oppose ouvertement à la grammaire générative et à des représentations par arbre comme dans le cas des grammaires de constituants immédiats de Chomsky sur lesquels nous ne ferons pas de commentaires. Les lecteurs intéressés par cette discussion peuvent consulter (Shieber, 1985 ; Gross, 1975, p. 34-36). Nous pensons que pour décrire une langue, on doit commencer par observer un nombre raisonnable de faits (corpus), les décrire de façon simple et compréhensible, les généraliser pour arriver enfin à des conclusions ou des théorisations. Un outil formel ne serait approprié que s'il est le résultat direct de ces descriptions et de ces généralisations. Cette méthode de travail empirique est encore plus adaptée lorsqu'il s'agit de la conception d'une application de traitement automatique du langage naturel car il nous parait imprudent d'essayer de résoudre un problème

sans connaître sa nature, ni l'avoir étudié dans le moindre détail.

Deuxièmement, la conclusion de M. Gross et une de ses perspectives vis-à-vis du TAL nous intéressent plus particulièrement. Le fait que 3 000 verbes constituent 2 000 classes (cf. descriptions sur les tables de lexique-grammaire, page 31) suggère qu'en général il n'existe pas de couple de verbes qui possèdent les mêmes propriétés sémantico-syntaxiques car une classe contient en moyenne 1,5 verbe (notons que M. Gross construit des tables de verbes avec leurs propriétés syntaxiques en utilisant des étiquettes sémantiques pour marquer la nature des arguments du verbe). Nous pouvons en déduire qu'en TAL il serait difficile d'envisager une grammaire très générale qui pourrait reconnaître ou générer des phrases à la fois syntaxiquement et sémantiquement correctes. Autrement dit, on peut bien écrire une grammaire puissante qui accepte ou génère un très grand nombre de phrases syntaxiquement correctes mais rien ne garantit que ces phrases seront toutes sémantiquement correctes.

Pour que la grammaire devienne opérationnelle, un ensemble de conditions d'application et de restrictions sémantiques doit être satisfait : il y aura de ce fait la partie grammaire et la partie conditions. Souvent, à force de rendre la couverture de la grammaire plus grande, les conditions et restrictions augmentent de façon exponentielle. Bien entendu, ceci n'est pas nouveau, les chercheurs en TAL connaissent bien ce problème. Les difficultés de rédaction des règles de grammaires posent assurément des problèmes quand il s'agit de rédaction de règles paraphrastiques.

En effet, on ne peut pas avoir un ensemble de règles paraphrastiques très générales qui s'appliqueraient à un ensemble de verbes syntaxiquement apparentés sans exceptions ou contraintes. À titre d'exemple, une règle paraphrastique générale comme : « les verbes transitifs directs acceptent la construction passive » sera certes correcte du point de vue de la grammaire mais il existe des verbes transitifs

qui acceptent mal la construction passive.

On constate, en raison des propriétés sémantico-syntaxiques quasi-uniques des verbes, qu'un nombre de chercheurs en TAL se penchent vers des grammaires dites lexicalisées. L'analyse automatique à l'aide de grammaires lexicalisées repose sur des dictionnaires qui contiennent, le plus souvent, des entrées verbales munies d'informations sur les arguments régis par le verbe et sur la configuration structurelle de la phrase (constructions, frames, gabarits, cadres). Le dictionnaire dans ce cas ne se comporte pas comme une simple ressource lexicale mais assume aussi le rôle de source de règles de grammaire. Ci-dessous un passage intéressant sur l'évolution des techniques d'analyse automatique des phrases (Gross, 1975, p. 221) :

> Une éventuelle analyse automatique prendrait alors une forme nouvelle. [...] Des consultations des dictionnaires sont effectuées pour les verbes de la phrase à analyser, elles déterminent des ensembles de contextes possibles pour chaque verbe. L'exploration des contextes réels (p. ex. L'examen des substantifs et de leurs relations avec les verbes), permettent d'effectuer un choix parmi les contextes possibles et donc d'éliminer ceux qui sont incorrects.

Si M. Gross a raison, l'ère des analyseurs syntaxiques et des corpus annotés toucherait à sa fin après avoir été au centre du traitement automatique des langues pendant une cinquantaine d'années.

Cependant, une grammaire lexicalisée n'est pas sans inconvénient. Compte tenu du nombre de verbes, de constructions possibles, de la nature sémantique du sujet et des compléments (ceci implique un dictionnaire complet des substantifs, voire un thésaurus pour qu'un analyseur puisse profiter du lexique-grammaire), une table de lexique-grammaire capable de couvrir les données langagières du domaine général paraît difficilement réalisable. D'autre part, une grammaire trop plate ne

sera qu'un dénombrement de phrases existantes en langage naturel, ce qui est sans appel impossible. Il y faut alors un certain degré de généralisation pour alléger les tables du lexique-grammaire, ou alors une automatisation maximale dans le processus de construction des tables serait idéale. C'est cette idée d'automatisation qui nous intéresse.

Nous pensons qu'une extraction automatique ou semi-automatique de règles paraphrastiques est préférable à l'écriture manuelle des règles. Le processus d'extraction de règles paraphrastiques fait l'objet de formalisations plus complètes dans la partie formalisation et implémentation.

3.4 Conclusion

Les tables du lexique-grammaire fournissent quelques transformations paraphrastiques simples pour une entrée lexicale donnée. Or, la notion de paraphrase et la définition systématique des règles paraphrastiques ainsi que le lien paraphrastique entre la phrase source et ses transformées en sont absents.

Cependant, ce sont deux autres idées de M. Gross qui méritent d'être soulignées. La première concerne la méthode de description empirique des phénomènes linguistiques qui consiste à : analyser d'abord les données ; ensuite à les décrire et à les généraliser pour en exposer les résultats et en tirer des conclusions. Nous pensons que l'abstraction hâtive des phénomènes linguistiques sans analyse méthodique minutieuse risque de passer à côté d'éléments importants. En outre, ce seraient les outils formels de description qui se construiraient à la suite des analyses linguistiques et non l'inverse.

La seconde pensée se rapporte à l'analyse automatique des phrases qui s'appuie sur la grammaire lexicalisée. Il s'agit en fait de dictionnaires qui incluent à la fois

de l'information lexicale et des règles de grammaire. En effet, une grammaire basée sur les parties du discours s'avère si puissante en termes de génération qu'elle a besoin de restrictions ou de contrôle lors de son application. Les règles de grammaire et leurs conditions d'application deviennent vite ingérables face aux données linguistiques de taille importante à cause des modifications rétrospectives constantes qu'il faut faire pour éviter les conflits entre les règles. C'est là que la grammaire lexicalisée pourrait avoir des avantages car elle est plus riche en informations, et a une forme plus compacte. Nous pensons qu'une grammaire lexicalisée avec des généralisations subtiles représente une méthode efficace pour le traitement automatique des langues en général et qu'elle devrait être un bon framework pour le traitement automatique de la paraphrase.

Conclusion

Nous avons montré, à travers les trois approches linguistiques de la formalisation des règles paraphrastiques, qu'il existe un écart entre la linguistique théorique et le traitement automatique de la paraphrase. Notre but n'est pas de réfuter ces théories. Au contraire, nous en avons extrait des concepts indéniablement intéressants pour le traitement automatique de la paraphrase. Et ce faisant, il a été indispensable d'en souligner les inconvénients. Ainsi, des systèmes de règles paraphrastiques considérées dans le monde de la linguistique, seuls sont excellents les outils de description prenant en compte le lien paraphrastique, tel est par exemple le cas des fonctions lexicales de la théorie sens-texte. Or, dans un contexte réel de développement d'applications en TAL, ces formalisations demandent un temps de développement considérablement long. Par ailleurs, il y a la question de la possibilité d'implémentation qui peut remettre en cause ces règles formelles.

Sommes-nous en train de dire qu'il faut rejeter les règles paraphrastiques ? En aucun cas, parce que la paraphrase est de nature linguistique et doit être décrite par des connaissances linguistiques. Cependant, elles doivent en premier lieu être formelles de façon à ce qu'elles soient implémentables. Ensuite, ces règles formelles ne doivent pas demander un temps de développement important, ou alors, on doit trouver une technique d'automatisation, même partielle, pour la rédaction des règles. Cette automatisation dépend bien entendu de la façon dont les règles ont été formalisées : elles doivent supporter l'automatisation. Sinon, malgré une élégance théorique, ces règles paraphrastiques n'auraient qu'un statut symbolique et deviendraient difficilement exploitables dans le vrai contexte de TAL.

Deuxième partie

Analyse Linguistique et Formalisation

Chapitre 4

La portée de notre travail

4.1 Notre méthodologie

Nous avons dit dans notre première partie qu'il existait plusieurs approches à la formalisation des règles paraphrastiques. Ces formalisations présentent des avantages, aussi bien que des inconvénients vis-à-vis du traitement automatique de la paraphrase. Quant à nous, l'objectif de notre recherche n'est pas de proposer de faire de la concurrence à ce qui existe pour la formalisation de règles paraphrastiques. Il ne s'agit pas non plus de monter à tout prix une théorie qui serait nôtre. Il n'est question encore ni de suivre des méthodes existantes à la lettre, ni de faire une combinaison complexe de bonnes idées issues de plusieurs théories. En effet, ces solutions ne seront envisageables qu'après l'analyse de données linguistiques réelles.

Ainsi, nous procédons à l'analyse de notre corpus en nous détachant de toutes théories linguistiques. Cette démarche est pour nous la seule solution qui permettra de découvrir les faits importants liés à la paraphrase et au traitement automatique de celle-ci. En effet, il y a certes des études sur la paraphrase qui ont été faites

mais il semble que personne ne se soit intéressé à des questions comme : quels sont les mécanismes de paraphrasage les plus courants dans un corpus donné ? comment formaliser des règles qui puissent prendre en compte le mécanisme de paraphrasage le plus utilisé ? Ce sont ces deux questions essentielles qui doivent être posées lorsqu'on travaille la paraphrase sous l'angle du traitement automatique. Le problème est qu'en TAL on s'intéresse à la performance de systèmes informatiques, à savoir le taux de leur succès face à une multitude de textes. Sous ce prétexte, il ne faut pas que tout soit permis pour rendre le système performant, car un système qui n'est pas basé sur une philosophie sera l'équivalent d'un paquet de mouchoirs jetables (après utilisation). Nous voulons dire simplement que pour obtenir un système performant et fiable, on doit absolument identifier les problèmes récurrents d'un corpus, les décrire et formaliser les solutions. De cette façon, ces formalisations seront issues d'une analyse linguistique fine, auront un fondement théorique et pourront ainsi être réappliquées dans d'autres circonstances. Nous pensons que cette démarche convient aux travaux de recherche qui se situent à la fois dans la linguistique et dans le traitement automatique.

Nous pouvons résumer notre méthodologie de recherche en trois grandes étapes : analyser des données linguistiques, décrire les faits et problèmes liés à la paraphrase, formaliser les règles paraphrastiques. C'est dans l'ordre chronologique suivant la méthodologie décrite ci-dessus que nous présentons nos travaux de recherche. Nous discutons le corpus utilisé pour cette recherche dans la prochaine section. Ensuite, nous commençons la partie de l'analyse linguistique par une définition des types de la paraphrase traitée dans la section 4.3. Dans le chapitre 5, nous décrivons les résultats de l'analyse du corpus et les phénomènes les plus intéressants liés à la paraphrase. Nous proposons dans le chapitre 6 la formalisation qui s'inspire de nos travaux d'analyse.

4.2 Corpus

Le corpus utilisé dans cette recherche est constitué de 899 verbatims du domaine agroalimentaire. Il représente 2 376 phrases. Un verbatim ainsi contient en moyenne 2,6 phrases. Il s'agit en effet des courriels envoyés par la clientèle à une entreprise dans le secteur agroalimentaire. La plupart des cas concernent les réclamations et des demandes d'information. La figure 4.1 (page 45) donne quelques exemples de verbatims tirés du corpus.

On remarque à travers ces verbatims les trois registres de langue habituels : soutenu, standard, familier. Contre toute attente, le style d'écriture est souvent direct, émotionnel et bref malgré une situation d'énonciation assez formelle. Cela s'explique par le fait que de nombreux clients sont mécontents, voire en colère. Dans ce cas, ils ne cherchent plus à employer des tournures sophistiquées. Les facteurs sociolinguistiques sont hors de portée de notre recherche. Seuls les mécanismes de paraphrasage derrière ces phrases constituent notre objectif de recherche.

Ce corpus est riche en « structures paraphrastiques ». En effet, il est évident que les sujets de courriels se ressemblent et représentent un ensemble plutôt restreint : réclamation, appréciation, demande d'information, demande de remboursement, expression de déception, demande d'échantillons ou de marchandises, etc. Or, pour exprimer une idée, il existe un nombre infini d'expressions. Pour pourvoir discuter plus facilement la question des relations entre un sens et ses moyens d'expressions, considérons la figure 4.2 (page 46) qui esquisse une classification possible des sens selon le contenu des messages.

Le schéma indique que verbatim_2 contient des phrases classées dans sens_2 et sens_3, de même pour verbatim_3. Il est important de préciser qu'ici on a affaire aux structures paraphrastiques et non des paraphrases proprement dites. La raison qui nous empêche d'aborder directement les paraphrases se trouve dans les variables

1. 2 sortes de croquettes des NOM_du_PRODUIT et NOM_du_PRODUIT. ma chatte a eu du mal à les manger au départ, je lui mettais les deux sortes et elle allaient sur les autres, c'était du NOM_du_PRODUIT en croquettes fourrées et elle mange des sachets aussi NOM_du_PRODUIT. elle a un poil Angora et elle a souvent un genre de desquamation et la elle la trouve. nous allons souvent chez le vétérinaire. au niveau des selles c'est toujours pareil, elle ne fait pas souvent dans la caisse. je voudrais me faire rembourser.

2. fidèle depuis des années de votre produit NOM_du_PRODUIT, j'ai déchanté depuis la dernière boite acheté en effet c'est assez inadmissible il ma rappelle les cafés de la guerre. cela me déçoit beaucoup.

3. Il n'y avait pas de tomate, ni d'épices dans ce sachet, et çà m'est arrivé avec pour l'instant 2 sachets, je ne sais pas comment sont les autres que j'ai achetés. Je vous renvoie ce sachet, car je n'ai pas rajouté d'eau, à l'ouverture, il n'y avait pas le parfum habituel et pas de tomates non plus. J'espère que vous allez remplacer les sachets que j'ai achetés pour rien du tout.

4. nous avons été fort surpris de la coloration du potage où il semblait manquer des tomates et des aromates, le potage était non seulement incolore mais totalement insipide et impropre à la consommation, nous avons ouvert un autre sachet (ci-joint le sachet) et avons constaté qu'il semblait également manquer des ingrédients. Merci de nous expliquer de nous expliquer la raison de cette non-conformité et de nous indemniser pour le désagrément subi, nous avons au total 6 sachets avec les mêmes codes.

FIGURE 4.1 – Exemples de verbatims

X et Y : X renvoie à l'auteur du message ; Y désigne normalement les produits de l'entreprise. Par exemple, Y peut prendre la valeur de *yaourt, pizza*, « votre marque », *vous*, autrement dit les entités appartenant à l'univers de l'entreprise. Un sens à ce stade correspond en quelque sorte au sens du prédicat *apprécier, décevoir*, « être allergique » avec leurs arguments présentés par des variables non instanciées. Dans cette thèse, nous employons d'une façon interchangeable le terme

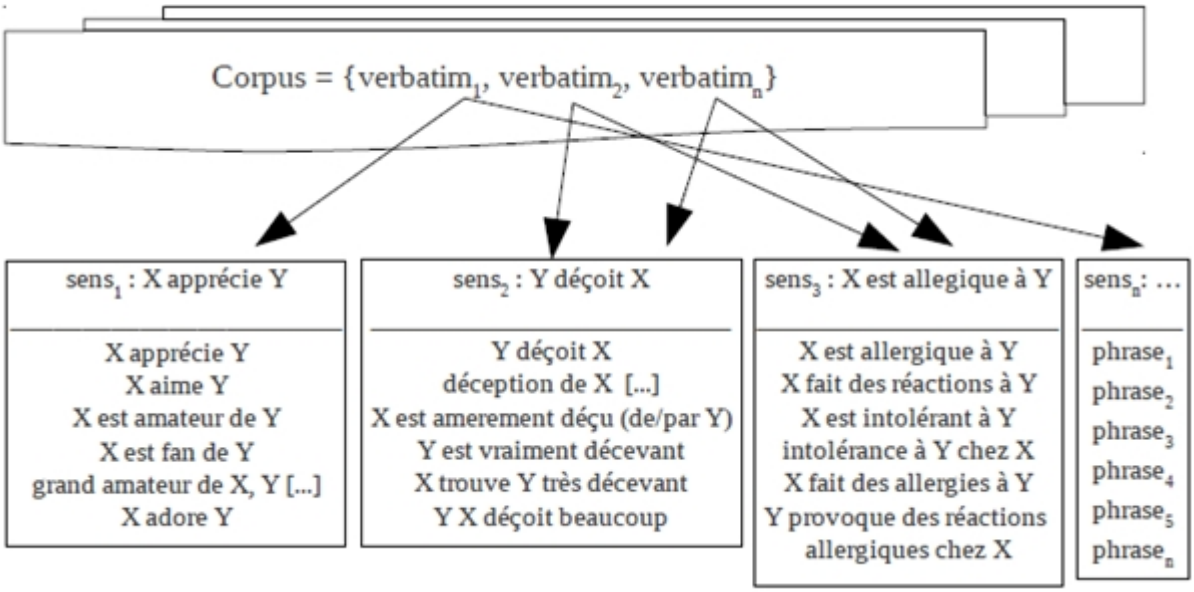


FIGURE 4.2 – Classification des sens

sens et *sème*. Le dernier n'a pas le sens de « l'unité minimale de signification », ni « l'atome de signification » telle définie respectivement par le linguiste belge Éric Buyssens et le linguiste français Bernard Pottier. Dans notre contexte, un sème correspond au « sens qui représente un ensemble de paraphrases ». Par exemple, un sème « X est allergique à Y » peut présenter la phrase : « mon fils est allergique au gluten », « l'allergie au gluten chez mon fils », « mon fils est reconnu allergique au gluten », « mon fils a développé une intolérance au gluten ». Le terme *sème* accompagne ainsi l'idée de paraphrases. Quant à la forme du sème « X est allergique à Y », celle-ci a été choisie d'une façon subjective car il aurait pu également être « allergie à Y chez X ». Selon le contexte, nous pouvons par exemple mentionner « le sème aimer » tout court, sachant qu'il revoie au sème « X aimer Y » qui serait préalablement abordé.

Une question ainsi se pose : pourquoi ne pas travailler directement sur les paraphrases ? Nous pouvons citer trois raisons majeures qui justifient le choix de

la classification en sèmes plutôt qu'en un regroupement en paraphrases proprement dit.

Premièrement, le nombre de phrases analysées paraît assez important en termes de traitement car cette phase d'analyse est entièrement manuelle. Cependant, cet ensemble de phrases n'offre pas assez de paraphrases (à l'opposition de structures paraphrastiques).

Deuxièmement, il est plus pratique de regrouper d'abord les phrases en sèmes. En effet, l'analyse manuelle est un travail minutieux qui demande beaucoup d'attention, de concentration et de mémoire. Parcourir le texte et chercher à établir les groupes de toutes les paraphrases possibles semblent outrepasser nos capacités cognitives, ou alors cette tâche nous aurait pris trop de temps pour des raisons personnelles. Au contraire, un classement en sèmes facilite la tâche en offrant ainsi une représentation simplifiée au lieu d'une multitude d'expressions linguistiques représentant chacune un cas.

Troisièmement, nous nous intéressons plus aux structures paraphrastiques qu'aux paraphrases. En effet, il est bien connu de tous que le lexique varie considérablement d'un domaine à l'autre alors que ce n'est pas le cas pour la structure qui ne diffère guère d'un texte à l'autre. De ce fait, nous mettons en avant les structures paraphrastiques ayant la forme d'un prédicat logique. Il s'ensuit que nos travaux auront un impact plus important sur le traitement automatique de la paraphrase car les structures paraphrastiques ne dépendent pas spécifiquement du domaine agroalimentaire et peuvent s'appliquer à d'autres domaines. En d'autres termes, travailler au niveau de phrases donnerait suite à des résultats qui ne seraient valables que pour le domaine dont le corpus relève. Or, en nous plaçant à un niveau structurel, la portée de nos travaux s'étend au domaine général des textes.

4.3 Typologie de la paraphrase

Avant de présenter les analyses linguistiques des phrases et leurs résultats, il nous semble indispensable d'établir explicitement la portée de notre recherche. Nous allons ici préciser les types de paraphrases traitées dans notre travail.

Le premier point que les linguistes travaillant sur la paraphrase ont dégagé est le type de paraphrase traité. Il relève de deux catégories très différentes : la paraphrase linguistique et les autres paraphrases (paraphrase cognitive, paraphrase pragmatique entre autres).

La paraphrase linguistique se réfère à la relation paraphrastique qui peut être mise en évidence par des procédés linguistiques.

Exemple 1

(a) Jean enseigne la linguistique.

(b) Jean est enseignant de linguistique.

1(a) et 1(b) sont des paraphrases de type linguistique. En effet, cette paraphrase met en jeu une substitution lexicale, *enseigner* est remplacé par le dérivé nominal *enseignant*. Ensuite, le substituant *enseignant* déclenche les modifications morpho-syntaxiques suivantes : l'ajout du copule *est* ; l'ajout d'une préposition *de* entre *enseignant* et *linguistique* ; l'omission de l'article défini *la* qualifiant *linguistique*. Ce sont des mécanismes purement linguistiques qui expliquent le lien paraphrastique entre les 1(a) et 1(b). La paraphrase linguistique peut ainsi se définir comme la relation paraphrastique qui ne met en jeu que des procédés linguistiques.

Les paraphrases autres que linguistiques sont par exemple :

Exemple 2

(a) Le président de la France se rendra aux Étas-unis ce mercredi.

(b) François Hollande sera chez Barack Obama dans deux jours.

2(a) et 2(b) seront paraphrases l'une de l'autre si toutes les conditions suivantes sont satisfaites :

1. au moment où l'énoncé est émis, François Hollande occupe la fonction du chef de l'état français ;
2. nous sommes lundi, soit moins deux jours de la date de départ du président de la France ;
3. Barack Obama habite aux États-Unis ;
4. « se rendre dans un endroit » implique « être à cet endroit ».

On peut constater que l'établissement de la relation paraphrastique entre 2(a) et 2(b) implique certaines connaissances encyclopédiques (1. et 3.), des connaissances sur les indices temporels « dans deux jours », « ce mercredi » (2.), et la preuve selon laquelle « se rendra dans un endroit » implique que « être à cet endroit » est vrai (4.). Par conséquent, on parle de paraphrase cognitive ou pragmatique car les connaissances linguistiques, seules, ne suffisent pas à démontrer la relation paraphrastique entre 2(a) et 2(b).

Nos travaux de recherche portent sur la paraphrase linguistique. Dès lors, le terme paraphrase utilisé dans cette thèse renvoie généralement à la paraphrase linguistique. Il nous reste par la suite à discuter des sous-types de paraphrase linguistique.

Les paraphrases peuvent être classées selon plusieurs critères. J. Milićević (2007, p. 139) propose une typologie intéressante de la paraphrase linguistique. Les critères de classement, comme on peut le voir dans la table 4.1 ci-dessous, sont :

– l'aspect du sens impliqué,
– l'exactitude du lien paraphrastique,

– les moyens paraphrastiques impliqués,

– la profondeur du lien paraphrastique,

– le mode de production.

Aspect du sens impliqué	**Exactitude du lien paraphrastique**	**Moyens paraphrastiques impliqués**	**Profondeur du lien paraphrastique**	**Mode de production**
Sémantiques	Exactes	Lexicaux	Sémantiques	Virtuelles
Communicatives	Approximatives	Syntaxiques	Lexico-syntaxiques	Reformulatifs
Rhétorique		Morphologies	Syntaxiques	
		Prosodies	Morphologiques	
		Constructionnels		

TABLE 4.1 – Typologie de la paraphrase de J. Milićvić

Nous reprenons tel quel le tableau de J. Milićević pour décrire les types de paraphrases qui nous intéressent.

En ce qui nous concerne, trois propriétés sont à prendre en considération : l'exactitude du lien paraphrastique, les moyens paraphrastiques impliqués et la profondeur du lien paraphrastique. Quant aux deux autres propriétés, l'aspect du sens impliqué et le mode de production, la première paraît théoriquement trop subtile pour être prise en compte du point de vue du traitement automatique ; le mode de production, lui, relève étroitement de la formalisation de la paraphrase dans la théorie sens-texte. Dans la table 4.2 ci-dessous, il reste ainsi trois colonnes de la table 4.1 à discuter.

4.3.1 Exactitude du lien paraphrastique

Observons la première colonne de la table 4.2, une paire de paraphrases peut être classée selon l'exactitude du lien paraphrastique. Sont dites paraphrases exactes les paraphrases qui ont parfaitement le même sens et peuvent ainsi se substituer

Exactitude du lien paraphrastique	**Moyens paraphrastiques impliqués**	**Profondeur du lien paraphrastique**
Exactes	Lexicaux	Sémantiques
Approximatives	Syntaxiques	Lexico-syntaxiques
	Morphologies	Syntaxiques
	Prosodies	Morphologiques
	Constructionnels	

TABLE 4.2 – Typologie de la paraphrase 2

dans tous les contextes sans changement de sens. Or, existe-il une paire de paraphrases exactes ? Considérons les exemples suivants :

Exemple 3

(a) Les pâtes n'ont aucune saveur.

(b) Les pâtes n'ont aucun goût.

(c) Les pâtes n'ont pas de goût.

(d) Les pâtes ont un goût trop fade.

(e) Les pâtes sont trop fades.

(f) Les pâtes sont sans goût.

(g) Le goût des pâtes est fade.

Ce sont certes des paraphrases, c'est le moins qu'on puisse dire. Mais, y-a-t-il une paire de paraphrases exactes parmi ces paraphrases ? Les deux premières phrases, 3(a) et 3(b), représentent une paire candidate la plus probable à être exactes :

(a) Les pâtes n'ont aucune saveur.

(b) Les pâtes n'ont aucun goût.

Ainsi, *saveur* et *goût* seraient des synonymes exacts. Cela signifierait que ces deux mots ont exactement le même sens et peuvent se substituer dans toutes les circonstances sans modifier le sens de la phrase dans laquelle ils se trouvent. À vrai dire, personne ne saurait répondre à ces questions car il est impossible de déterminer l'ensemble des phrases dans lesquelles *saveur* et *goût* apparaissent. Cependant, il est possible de prouver le contraire : il y a des contextes où les deux mots se remplacent difficilement :

Exemple 4

(a) Yaourt Summer saveur citron (publicité) vs. Yaourt Summer goût citron

(b) Nous proposons une cuisine traditionnelle, pleine de saveurs (carte de restaurant). vs. Nous proposons une cuisine traditionnelle, pleine de goûts.

Dans les deux exemples, *saveur* et *goût* sont des synonymes. Mais il est peu probable qu'on emploie les tournures « Yaourt Summer goût citron » ou « Nous proposons une cuisine traditionnelle, pleine de goûts » dans une publicité, ni dans une carte de restaurant. Une explication possible vient du fait qu'au mot saveur a été assignée une idée de luxe, une impression que l'on a affaire à quelque chose de sublime (et il y a sans doute d'autres interprétations possibles). Le mot *goût* peut être positif ou négatif et s'emploie facilement dans des énoncés négatifs comme ceux ci-dessous ; ce qui n'est pas le cas pour *saveur*, lui, étant toujours positif.

Exemple 5

(a) il y a un goût de fer dedans. vs. ?il y a une saveur de fer dedans.

(b) goût de lait tourné vs. ?saveur de lait tourné

(c) Le goût était un peu rance. vs. ?La saveur était un peu rance.

(d) J'en ai marre du goût imbuvable de ce produit. VS. ?J'en ai marre de la saveur imbuvable de ce produit.

Ces derniers exemples confirment que *goût* et *saveur* ne sont pas synonymes exacts. Revenons maintenant à la question initiale, à savoir si :

3(a) Les pâtes n'ont aucune saveur. et

3(b) Les pâtes n'ont aucun goût,

sont paraphrase exacte l'une de l'autre. Prouver une relation de non-équivalence entre *saveur* et *goût* revient-il à dire que les deux phrases ne sont pas des paraphrases exactes ? Pas nécessairement et même si l'on faisait appel aux calculs logiques pour fournir des preuves formelles, ce sera toujours discutable.

Quoi qu'il en soit, nous pensons que la plupart des paraphrases sont approximatives. Il existe probablement des paraphrases exactes mais en nombre restreint car ces dernières s'opposent au « principe d'économie » de la langue selon lequel le langage tend à mettre en avant les éléments essentiels et à se débarrasser des redondances. Ainsi, nous ne parlerons que de la paraphrase approximative, ce qui rend la table de classement 4.3 plus compacte :

Exactitude du lien paraphrastique	**Moyens paraphrastiques impliqués**	**Profondeur du lien paraphrastique**
Approximatives	Lexicaux	Sémantiques
	Syntaxiques	Lexico-syntaxiques
	Morphologies	Syntaxiques
	Prosodies	Morphologiques
	Constructionnels	

TABLE 4.3 – Typologie de la paraphrase 3

4.3.2 Moyens paraphrastiques impliqués

La deuxième colonne de la table recense les moyens paraphrastiques qui peuvent être mis en jeu lors du paraphrasage. D'une part, nous excluons les paraphrases par prosodie car on se place alors dans la langue orale, domaine hors de notre étude. D'autre part, comme les moyens paraphrastiques constructionnels n'ont pas été formellement définis par J. Milićević (2007, p. 128, 139-140), nous les laissons de côté (il s'agit vraisemblablement de simples transformations syntaxiques telles que « the decision of the gouvernment → the gouvernment's decision ». Il nous reste par conséquent trois outils principaux de paraphrasage : le lexique, la syntaxe et la morphologie. La paire de paraphrases ci-dessous illustre ces trois moyens de paraphraser.

Exemple 6

(a) Etant fidèle client de vos pizzas surgelées, cette fois, je vous écris pour vous faire part de mon insatisfaction.

(b) Je vous écris, cette fois, pour vous faire part de mon mécontentement parce que je suis un fidèle consommateur de vos pizzas surgelées.

Le moyen purement syntaxique de paraphrasage dans l'Exemple 6 est évident. On le voit à travers le déplacement du complément circonstanciel « cette fois », et la proposition « je vous écris pour vous faire part de mon insatisfaction ».

Ensuite, le moyen lexical s'exprime par la substitution d'*insatisfaction* par son synonyme *mécontentement*. Finalement, *étant* et *suis* dans « je suis/étant un fidèle consommateur de vos pizzas surgelées » est un exemple de paraphrasage morphologique.

La phrase 6(a) et 6(b) fournissent des exemples nets de chaque technique paraphrastique. Or, les choses sont souvent plus complexes du fait que les trois méca-

nismes interagissent. On parle de relation déclencheur-déclenché. Ainsi, un mécanisme de substitution lexicale se met en route, et entraîne diverses modifications syntaxiques et morphologiques. Considérons l'Exemple 7 ci-dessous :

Exemple 7

(a) Le bébé est allergique au gluten.

(b) Intolérance au gluten chez le nouveau-né.

Supposons que 7(b) est dérivé de 7(a). Il y a certainement plusieurs façons d'expliquer le procédé de paraphrasage en usage dans ces phrases : on commence par un déplacement (syntaxe), puis on a une substitution (lexique), enfin une inversion (lexique et syntaxe). Cependant, une des façons les plus naturelles serait de partir du lexique (ou prédicat) *allergique*. Nous allons montrer comment on arrive à 7(b) en partant de 7(a).

Premièrement, il est plus facile de se positionner au niveau de la structure prédicat-argument. La phrase 7(a) a la structure prédicat-argument suivante :

argument 1	prédicat	argument 2
bébé	allergique	gluten

On aperçoit ainsi les unités lexicales qui représentent les principaux acteurs participant au paraphrasage. D'abord, on nominalise *allergique* pour obtenir sa base nominale *allergie*, d'où « bébé allergie gluten ». Deuxièmement, on substitue *allergie* par son synonyme *intolérance*, la structure prédicat-argument aura ainsi le nouveau prédicat :

argument 1	prédicat	argument 2
bébé	intolérance	gluten

Revenons à la forme de surface de la phrase, nous avons procédé aux opérations suivantes :

1. Nominalisation + suppression du verbe *est* :

   Le bébé est allergique au gluten → *Le bébé allergie au gluten

2. Substitution synonymique :

   *Le bébé allergie au gluten → *Le bébé intolérance au gluten

Jusqu'ici seul un mécanisme lexical a été mis en jeu. La syntaxe intervient ensuite pour ordonner les éléments de la phrase :

3. Déplacement du sujet *bébé* :

   *Le bébé intolérance au gluten → *intolérance au gluten le bébé

Finalement, une suite de corrections morphologiques s'impose pour rendre la phrase grammaticale :

4. Ajout d'un article défini *l'* à *intolérance* :

   *intolérance au gluten le bébé → *L'intolérance au gluten le bébé

5. Ajout d'une préposition *chez* devant « le bébé » :

   *L'intolérance au gluten le bébé → L'intolérance au gluten chez le bébé

Ce n'est qu'un processus séquentiel de paraphrasage possible pour 7(a) et 7(b) en partant de 7(a). En tout cas, ce que nous voulons souligner ici, c'est le concept de déclencheur-déclenché : une substitution lexicale entraîne des reconfigurations syntaxiques ainsi que des modifications morphologiques.

Revenons au tableau de classement de la paraphrase linguistique. Nous retenons dans la deuxième colonne les moyens lexicaux, syntaxiques et morphologiques :

Exactitude du lien paraphrastique	**Moyens paraphrastiques impliqués**	**Profondeur du lien paraphrastique**
Approximatives	Lexicaux	Sémantiques
	Syntaxiques	Lexico-syntaxiques
	Morphologies	Syntaxiques
		Morphologiques

TABLE 4.4 – Typologie de la paraphrase 4

4.3.3 Profondeur du lien paraphrastique

La profondeur du lien paraphrastique nous paraît le plus problématique de tous les critères de classification de la paraphrase. Le paraphrasage à l'un de ces niveaux signifie que le mécanisme de paraphrasage concerné s'opère dans cette couche linguistique. Par exemple, le paraphrasage au niveau sémantique implique des modifications de la représentation sémantique de la phrase ; le paraphrasage lexico-syntaxique concerne la représentation syntaxique de la phrase avec des opérations lexicales (substitution, dérivation entre autres). Nous allons maintenant entrer un peu plus en détail aux niveaux sémantique, syntaxique et lexico-syntaxique mais faisons d'abord une brève présentation du niveau morphologique (cf. Exemple 8).

Example 8

(a) J'irai à Paris. == Je vais aller à Paris.

(b) Je veux bien un café. == Je voudrais bien un café.

(c) Le blouson à Paul. == Le blouson de Paul.

Le paraphrasage morphologique touche des procédés morphologiques de paraphrasage au niveau de la surface de la phrase et ne concerne pas d'autres couches linguistiques. Dans le langage naturel, les paraphrases issues de ce processus de paraphrasage sont rares. On parle par exemple des ajouts, substitutions, ou suppressions au niveau des morphèmes grammaticaux (classes fermées : me/te/se, le/la/les, et/sur/dans, etc.), de la flexion (déclinaison et conjugaison).

Prenons quelques exemples des couples de paraphrases mettant en cause des opérations morphologiques. Beaucoup considèrent ces paraphrases comme faisant partie des paraphrases lexicales. En effet, tout dépend de la façon dont on voit les choses. Si l'on considère *veux* et *voudrais* dans 8(b) comme étant un même verbe avec deux différentes désinences, ce serait là un mécanisme morphologique. Or, si voudrais et veux sont considérés comme deux synonymes dans ce contexte, il s'agit d'une opération lexicale. Autre exemple, dans « Les électeurs sont mécontents » et « L'électorat est mécontent » (exemple d'une règle lexicale, (Milićević, 2007, p. 248)) du point de vue morphologique, c'est la dérivation qui est mise en jeu car *électorat* est dérivé d'*électeurs*. Du point de vue lexical, cette règle paraphrastique relève d'une opération entre deux mots dans une phrase : substitution dérivative (fonction lexicale : substitution dérivative) comme la nomment les tenants de la théorie sens-texte.

Niveau sémantique

Voici une définition de la paraphrase sous une forme plus simple : « se dit d'une relation équivalente sémantiquement entre plusieurs phrases ». Ceci laisse penser que ce niveau peut être traité facilement. Quel que soit le nombre de paraphrases, milliers ou millions, qu'elles soient construites avec des moyens morphologiques, syntaxiques ou lexicaux, ces paraphrases auront le même sens.

De ce qui vient d'être dit, la solution la plus naturelle au problème de traitement de la paraphrase sera sémantique, plus précisément liée à une représentation sémantique. L'expression « ces phrases auront le même sens », deux lignes plus haut, est synonyme de « ces phrases auront la même représentation sémantique ». Aussi la représentation sémantique serait-elle la clé de tous les problèmes liés au traitement de la paraphrase. Mais encore faut-il définir ce qu'est une représentation

sémantique.

C'est en effet cette représentation sémantique que nombre de linguistes et de chercheurs en TAL ont tenté de modéliser. À la différence d'autres niveaux linguistiques, la sémantique ne représente pas une entité observable. L'imagerie cérébrale et les recherches en neurolinguistique ont permis de localiser l'aire de Wernické du cerveau, l'un des principaux responsables du traitement du sens. Certes on sait que le sens existe mais on ignore à quoi il ressemble, et comment il s'organise.

On distingue, dans le monde de la linguistique et du TAL, deux approches dominantes pour représenter le sens d'une phrase. D'un côté, les logiciens proposent des langages logiques que l'on appelle « la sémantique formelle ». Les plus connus sont la logique du premier ordre, esquissé par Aristote, puis développé et bien présenté au début XXème siècle par Gottlob Frege, Bertrand Russell, Kurt Gödel, Alfred Tarski entre autres (Blanché, 1984) ainsi que la Grammaire de Montague apparue dans les années 70 (Montague, 1970 ; Chambrieul, 1989).

De l'autre côté, les linguistes proposent également des représentations sémantiques moins formelles. On peut citer, par exemple, la sémantique générative principalement développée par Goege Lakoff, Jim Mcawley, Katz-Fodor qui ne séparent pas les règles syntaxiques des règles sémantiques et la sémantique interprétative tenue par Ray Jackendoff et Noam Chomsky qui ont fait la distinction entre les deux niveaux. Ces deux approches ont comme base commune la Grammaire Générative de Noam Chomsky. Malgré un point de départ linguistique, les linguistes chomskyens se rapprochent finalement de la logique comme on peut le constater dans la synthèse sur la sémantique générative faite par Michel Galmiche (Galmiche, 1975). Noam Chomsky dans les années 90, rejette la distinction structures profondes/structures de surface et propose comme base de la linguistique la forme phonologique et la forme logique dans son Minimalist Program (Chomsky, 1993, 1995). D'autre part, Aleksandr Žolkovskij et Igor Mel'čuk (1965) ainsi que Jas-

mina Milićević (2007) qui placent la paraphrase au cœur de la théorie sens-texte, proposent une représentation du sens sous forme de réseau sémantique.

Il semble que la formalisation du côté des linguistes n'ait pas d'impact significatif en matière de traitement automatique en raison sans doute de formalisations insuffisantes. En même temps, nombre de linguistes considèrent l'approche logique comme trop rigide pour prendre en compte les phrases du langage naturel. On peut néanmoins remarquer que la logique (la logique des prédicats, la logique intensionnelle de Montague) reste un des outils les plus utilisés en matière de traitement sémantique.

Après tout, à cause de la complexité du traitement de la sémantique, les approches récentes du traitement de la paraphrase ne se servent pas systématiquement de représentations sémantiques. Seules quelques applications font usage de représentations sémantiques approfondie, parmi lesquelles on peut citer un traducteur du langage parlé Spoken Language Translator (Agnäs et al., 1994) entre les paires de langues européennes telles que Anglais/Français, Anglais/Suédois, qui intègre un noyau complexe The Core Language Engine (Alshawi, 1992). Le Core Language Engine propose des modules de traitement du langage naturel en commençant par la morphologie, la syntaxe jusqu'à la transformation (mapping) des phrases en formules logiques. Néanmoins, le développement du Spoken Language Translator n'a jamais été au-delà du prototype (Nugues, 2006).

Dans les tendances actuelles, on reste au niveau de surface (morphologie) jusqu'à la syntaxe. Parfois, on parle de l'analyse sémantique, mais qui fait référence en effet à la structure lexico-syntaxique, ou à une forme basique de la logique des prédicats.

Le niveau le plus productif est indéniablement la syntaxe qui utilise des ressources lexicales riches car la position (syntaxe) et les mots (lexique) sont des

paramètres concrets donc traitables par moyens informatiques.

Niveau syntaxique et lexico-syntaxique

Nous regroupons ces deux niveaux pour une raison simple et pratique : ils partagent le même niveau opératoire qui est la syntaxe. En effet, la distinction entre le niveau syntaxique et lexico-syntaxique n'est pas claire. Le niveau lexical nous paraît vague car il n'y a pas de représentation lexicale proprement dite. Le plus souvent nous avons une représentation syntaxique et sémantique. Il semble que J. Milićević veut dire par 'niveau lexico-syntaxique' que le niveau syntaxique est impliqué par des mécanismes lexicaux lors du paraphrasage.

Quoi qu'il en soit du paraphrasage au niveau syntaxique ou lexico-syntaxique, les opérations paraphrastiques restent au niveau de la syntaxe : on travaille sur la représentation syntaxique. Nous utilisons ainsi les deux termes d'une façon interchangeable, sachant que le terme niveau lexico-syntaxique met en relief le niveau syntaxique comme profondeur d'analyse, niveau muni de mécanismes lexicaux de paraphrasage.

Nous pensons que c'est ce niveau lexico-syntaxique qui convient le mieux au traitement automatique de la paraphrase à l'heure actuelle. C'est parce que premièrement, le paraphrasage à ce niveau permet de traiter des paraphrases sophistiqués, telles celles dans l'exemple 9 qui ont été démontrées par J. Milićević (2007).

Example 9

(a) Ce fait, Balthazar l'a passé sous silence. == Ce fait, Balthazar ne l'a pas mentionné. (p. 253)

(b) Balthazar gardait le silence. == Balthazar était silencieux. (p. 254)

(c) Il s'est calmé parce qu'il a pris le médicament. == Il s'est calmé suite à la prise du médicament. (p. 256)

(d) Balthazar a fidèlement traduit le bouquin. == Balthazar a fait une traduction fidèle du bouquin. (p. 261)

(e) Il cherchait un logement. == Il était à la recherche d'un logement. (p. 265)

Deuxièmement, comme nous le clamons depuis toujours, le lexique et la syntaxe sont observables, calculables, aussi supportent-ils des informatisations.

Troisièmement, il est logique d'aller du plus simple au plus compliqué, d'aller du lexique et de la syntaxe vers une représentation sémantique. Il est du plus grand intérêt d'optimiser les ressources disponibles avant d'entamer la recherche des autres dont on est incertain (personne ne sait exactement comment le sens se présente dans notre cerveau). Cela ne signifie pas qu'on doit contourner à tout prix la représentation sémantique, c'est même un devoir des chercheurs d'y apporter quelque lumière. Mais, comme il y a encore beaucoup de progrès à faire dans le traitement automatique de la paraphrase au niveau lexico-syntaxique, nous décidons de nous y investir.

Certes, notre analyse ne met pas en avant la représentation sémantique et la sémantique ne sera en aucun cas laissée de côté. Au contraire elle est omniprésente dans les chapitres qui suivent car nul ne peut aborder la paraphrase sans prendre en compte le sens. Avant de passer au chapitre suivant où nous faisons l'analyse du corpus et où nous présentons les résultats, établissons la table de classement de la paraphrase 4.5 qui récapitule les types de paraphrases qui nous intéressent.

Exactitude du lien paraphrastique	**Moyens paraphrastiques impliqués**	**Profondeur du lien paraphrastique**
Approximatives	Lexicaux	Lexico-syntaxiques
	Syntaxiques	Syntaxiques
	Morphologiques	Morphologiques

TABLE 4.5 – Typologie de la paraphrase 5

Chapitre 5

Analyse linguistique

5.1 Introduction

Dans ce chapitre intitulé « Analyse linguistique », nous décrivons la relation paraphrastique entre des structures paraphrastiques comme étant un ensemble d'opérations de transformations textuelles. Ce n'est qu'une simple explication linguistique possible, fruit d'une observation de données tirées du corpus. Nous n'attendons pas qu'elle soit la plus complète, ni la plus cohérente de toutes autres approches. Il s'agit, là, de la façon par laquelle nous voyons la paraphrase sans complexité, sans influence d'une quelconque théorie.

Pour ce faire, nous donnons les trois sèmes suivants : aimer, inquiéter et décevoir, avec des exemples de structures paraphrastiques les plus courantes où on rencontre chacune de ces trois unités de sens. Le nombre d'exemples analysés paraît certes modeste mais ils représentent un échantillon riche, recelant d'importants phénomènes linguistiques liés à la paraphrase, et qui permettra d'extraire des principes clés de la paraphrase et du traitement automatique de cette dernière.

Après l'analyse et la description linguistique des trois sèmes en question, nous

ferons une synthèse dans la section 5.5 qui a comme objet de souligner les grandes idées dégagées à travers le travail descriptif. Ces concepts serviront, à leur tour, à esquisser notre modélisation de la relation paraphrastique présentée et formalisée dans le chapitre 6.

5.2 Sème AIMER

Il s'agit ici du deuxième sens de l'entrée lexicale AIMER tel qu'il a été défini dans le dictionnaire de la langue française Le Petit Robert (2004) :

I. 1. Éprouver de l'affection, de l'amitié, de la tendresse, de la sympathie pour quelqu'un [. . .]

II. 1. Avoir du goût pour (qqch) => affectionner, apprécier, s'intéresser (à). Aimer la lecture, le sport. Aimer la musique, aimer Mozart [...] 2. trouver agréable, être content de, se plaire à [...]

III. V. pron. 1. Être attaché à soi, amoureux de soi, [. . .]

FIGURE 5.1 – Définition d'*aimer* par Le Petit Robert (2004)

Nous ne traitons pas la question de la désambiguïsation du sens des mots car elle est trop complexe pour être présentée en quelques pages. Passons directement à l'analyse et la description des exemples de phrases tirées du corpus (figure 5.2) qui représentent le sème *aimer*.

Dans la figure 5.2, on remarque que comme les phrases relèvent d'un domaine spécifique et d'une situation de communication bien précise (verbatims), beaucoup de phrases ne diffèrent que par quelques mots. Pour apercevoir les éléments les plus intéressants, nous les regroupons en sous-ensembles de phrases ayant une structure similaire, e.g. figure 5.3.

- je les adore
- ils en raffolent
- le bébé a aimé ce lait
- ils n'aiment que la terrine
- mon chat n'aime que le thon
- nous aimons bien
- les enfants ont aimé
- il l'apprécie beaucoup
- j'apprécie énormément vos produits
- nous apprécions vos produits en général
- je aime cette nouvelle formule
- mes chats apprécient la gelée
- je suis grande admiratrice de votre Nom_du_Produit au lait en stick
- très amateur des habituellement excellents Nom_du_Produit
- grand amateur de vos produits Nom_du_Produit
- je suis fan des yaourts Nom_du_Produit
- j'ai toujours été contente de votre marque
- nous en serions ravis
- il en sera ravi
- je ai été très satisfaite
- je suis contente
- je suis satisfait de ce produit
- je suis contente du résultat
- j'étais enchanté de ces petits pots pour leur capacité et leur contenu
- je suis contente de la réduction pour une boîte de Nom_du_Produit
- ils me plaisent presque tous dont particulièrement le poulet basquaise

FIGURE 5.2 – Exemples de phrases incluses dans le sème *aimer*

Nous obtenons ainsi quatre sous-ensembles de structures paraphrastiques appartenant au sème *aimer*. Nous donnerons des descriptions linguistiques de chaque groupe et relèverons des points importants relatifs à la modélisation de la relation paraphrastique.

X aimer Y
- je les adore
- ils en raffolent
- le bébé a aimé ce lait
- ils n'aiment que la terrine
- mon chat n'aime que le thon
- nous aimons bien
- les enfants ont aimé
- il l'apprécie beaucoup
- j'apprécie énormément vos produits
- nous apprécions vos produits en général
- je aime cette nouvelle formule
- mes chats apprécient la gelée

X être amateur de Y
- je suis grande admiratrice de votre Nom_du_Produit au lait en stick
- très amateur des habituellement excellents Nom_du_Produit
- grand amateur de vos produits Nom_du_Produit
- je suis fan des yaourts Nom_du_Produit

Y plaire à X
- ils me plaisent presque tous dont particulièrement le poulet basquaise

X être satisfait de Y
- j'ai toujours été contente de votre marque
- nous en serions ravis
- il en sera ravi
- je ai été très satisfaite
- je suis contente
- je suis satisfait de ce produit
- je suis contente du résultat
- j'étais enchanté de ces petits pots pour leur capacité et leur contenu
- je suis contente de la réduction pour une boîte de Nom_du_Produit

FIGURE 5.3 – Regroupement des phrases du sème *aimer*

5.2.1 X aimer Y

Les phrases de ce groupe se caractérisent par le partage d'une configuration lexicale et structurelle similaire. Ainsi, une phrase contient un verbe transitif *aimer* ou ses synonymes *adorer, apprécier, raffoler* avec une éventuelle négation restrictive *ne...que*, ainsi que les deux éléments régis par le verbe : X en fonction de son sujet, et Y du complément.

Nous qualifions « X aimer Y » comme structure de base à partir de laquelle dérivent la plupart des structures paraphrastiques appartenant au même sème. Certes ce n'est pour l'instant que notre point de vue. Mais nous montrerons dans les pages qui suivent qu'il y a une logique à cela. Voyons maintenant quelles remarques on peut faire sur ce premier groupe de phrases.

D'abord, on observe un COD *les* et COI *en* antéposé au verbe, respectivement dans « je les adore » et « ils en raffolent ». Ces phrases de structure SOV (Sujet-Objet-Verbe) font partie du même groupe que celles de type SVO (Sujet-Verbe-Objet). En effet, les deux phrases relevant du même type sont censées avoir une structure dite paraphrastique l'une de l'autre. Ainsi, l'appartenance au même groupe des deux structures SOV et SVO se justifie puisqu'en français elles sont grammaticalement interchangeables (figure 5.4).



FIGURE 5.4 – Equivalence structurelle entre SOV et SVO

Or, pour pouvoir parler de la paraphrase proprement dite, des analyses coréférentielles et discursives fines seront nécessaires pour prouver la relation paraphrastique entre de telles phrases, car *les* et *en* peuvent référer à divers éléments autour

d'eux dans le texte concerné. Par exemple dans les phrases suivantes :

(1) Les sachets de Nom_du_Produit ne sont plus disponibles dans les magasins. C'est dommage car mes enfants les adorent.

(2) Les sachets de Nom_du_Produit contiennent de très jolies cartes de Pokémon. Mes enfants en raffolent.

le pronom les dans (1) revoient à « les sachets de Nom_Produit » alors que dans (2) on ne sait pas si les enfants raffolent des produits ou des cartes offertes dans les sachets, ou encore les deux.

D'autre part, on observe, dans le corpus, un emploi intransitif du verbe *aimer* ou de ses synonymes, qui, pris hors du contexte, donnent lieu aux phrases sémantiquement et grammaticalement douteuses : *elle adore ; *nous adorons. ; *il aime. ; * ?nous aimons bien ; *elle aime ; *les enfants ont aimé. La première phrase par exemple, une fois en contexte, devient compréhensible : « Je nourris ma fille avec vos pots depuis l'age de 4 mois et elle adore ».

Il est simple pour l'être humain de retrouver le complément du verbe *adorer*. Il n'en va pas de même pour la machine qui a tout de même quatre groupes nominaux au choix : *je*, « ma fille », « vos pots » et « l'age de 4 mois ». Il faut, encore une fois, une analyse du discours très poussée pour reconstruire la phrase complète. En général, on ne peut s'attendre à ce que le complément d'*adorer* soit le COD ou le COI de la proposition précédente. On peut par exemple avoir la phrase : « j'ai emmené ma fille au magasin pour choisir elle-même les petits pots et elle adore » où le complément zéro renvoie très probablement à la première proposition tout entière. Par conséquent, nous écartons le problème du complément zéro, de coréférence et nous intéressons aux problèmes de la paraphrase linguistique.

5.2.2 X être amateur de Y

Le deuxième groupe des phrases est caractérisé par la présence d'un nom qui a le sens de « celui/personne/ce qui aime ». Ainsi peuvent être dans cette structure les dérivés sémantiques du mot *aimer*, tels que *amateur, fan, adorateur, admirateur* entre autres. Le terme « dérivation sémantique » a été défini par Alain Polguère et Igor Mel'čuk dans (Polguère & Mel'čuk, 2006) (notons qu'une lexie désigne une unité lexicale, soit un mot simple (lexème) ou une locution) :

> – Une dérivation sémantique est une relation entre deux lexies basée sur une parenté de sens. Plus précisément, une lexie L2 est dite sémantiquement dérivée d'une lexie L1 si et seulement si les trois conditions suivantes sont satisfaites :
> – L2 entretient une relation sémantique avec L1. Dans le cas le plus typique, L2 se définit en termes de L1. Par exemple, la lexie HACHE [=L2] est définie en termes de COUPER [=L1], car le sens 'hache' = 'artefact servant à couper...'.
> – La relation sémantique entre L2 et L1 est récurrente dans la langue. Par exemple, la relation entre HACHE et COUPER, 'artefact servant à...', est récurrente en français : FRAPPER => MARTEAU, OUVRIR [une porte] => CLÉ, FUMER => PIPE, etc.
> – La relation entre L1 et L2 s'exprime fréquemment de façon morphologique. Par exemple, pour la relation 'artefact servant à...', on trouve en français les dérivations suffixales suivantes : bouch(-er) => bouch+on, ras(-er) => ras+oir, décapsul(-er) => décapsul+eur, etc. content...

FIGURE 5.5 – Définition de la dérivation sémantique (Polguère et Mel'čuk, 2006)

Dans la théorie sens-texte, le nom *amateur* est le résultat de l'application de la fonction lexicale S1(aimer) qui se lit « premier actant sémantique de la lexie aimer ». En quelques mots plus simples, dans le contexte du mot *aimer*, S1 correspond au sens du sujet du verbe : « ce qui fait l'action désignée par le verbe ».

À proprement parler, rien ne nous empêche d'utiliser la notation S1(aimer) pour désigner les dérivés sémantiques ayant le sens de « celui/personne/ce qui aime ». Or, nous préférons ne pas emprunter les termes spécifiques à une théorie pour d'une part éviter toute confusion au niveau des définitions et des terminologies, et d'autre part pour préserver notre esprit libre de toute influence. Toutefois, nous ne négligeons pas les théories linguistiques sur la paraphrase, nous les avons étudiées (Partie I) et sommes convaincu qu'il y a d'autres approches aussi intéressantes et plus convenables du problème de traitement automatique de la paraphrase. C'est d'ailleurs la raison de l'existence de cette thèse. Poursuivons nos descriptions sur la relation entre le mot *aimer* et le mot *amateur*.

Le lien entre *aimer* et *amateur* ne pourrait-il s'expliquer par une simple relation synonymique directe ? Ainsi, on recourt par exemple au dictionnaire Le Robert, Dictionnaire des Synonymes (2005) qui fournit « être amateur de » comme un des synonymes de l'entrée lexicale *aimer*. Il suffirait alors, lors d'un traitement de la paraphrase, de consulter la liste des synonymes disponibles dans le Robert pour mettre en évidence la relation paraphrastique entre deux phrases dans la figure 5.6.

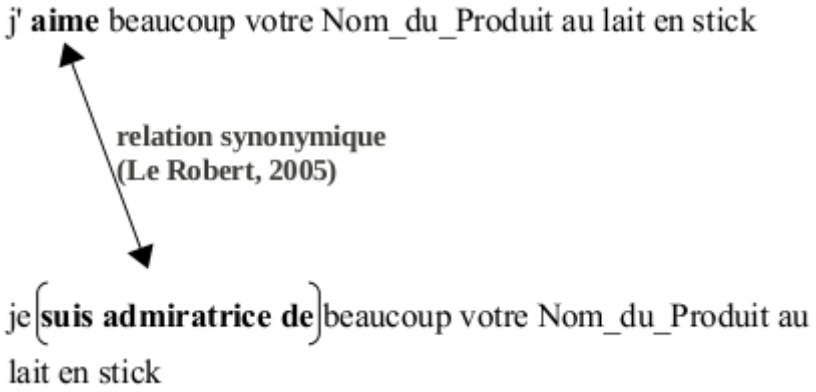


FIGURE 5.6 – Relation Synonymique entre *aimer* et « être amateur de »

Peut-on vraiment faire l'usage de dictionnaires de synonymes dans ce cas ? La réponse est oui, mais s'élèvent au moins deux inconvénients. Premièrement, dans

la figure 5.6 le lexème *aimer* entretient une relation synonymique avec l'expression « être amateur de » mais une substitution naïve du premier par le second conduit à une séquence agrammaticale « être amateur de beaucoup [...] ». On doit alors envisager une règle qui prend en compte les adverbes attachés au verbe.

Deuxièmement, les dictionnaires de synonymes n'établissent pas la relation synonymique entre *aimer* et d'autres dérivés nominaux tels que « être adorateur de », « être fan de » alors qu'*adorateur*, *fan* et *amateur* sont synonymes. De ce fait, les dictionnaires de synonymes traditionnels peuvent s'avérer utiles mais en même temps incomplets. Plus important encore, on doit être conscient du fait que les dictionnaires de synonymes sont du domaine général de la langue. Une fois dans un domaine spécifique, il est possible qu'ils ne soient plus valables. C'est aussi pour cette raison que nous insistons sur l'importance de l'analyse des données d'un corpus avant de procéder au traitement automatique de la paraphrase. Peut-être faut-il, au lieu de prendre un dictionnaire de synonyme quelconque comme allant de soi, essayer d'extraire les synonymes (ou paraphrases) du corpus.

Certes « être amateur de » est synonyme d'*aimer*. Mais au lieu de dire simplement qu'il s'agit d'une relation synonymique (ou paraphrastique), on peut aussi décrire cette relation comme étant le résultat de l'application d'une suite d'opérations de transformations morphologique, lexicale et syntaxique. Concrètement, étant donné *aimer*, on lui applique plusieurs opérations de transformations telles que substitution, ajout, suppression pour arriver finalement à « être amateur de ». La figure 5.7 ci-dessous récapitule les grandes étapes de transformation de la phrase « j'aime beaucoup votre Nom_du_Produit au lait en stick » en « je suis grande amatrice de votre Nom_du_Produit au lait en stick ».

Nous montrerons que la plupart des paraphrases linguistiques se soumettent à une description en termes d'opérations de transformation.

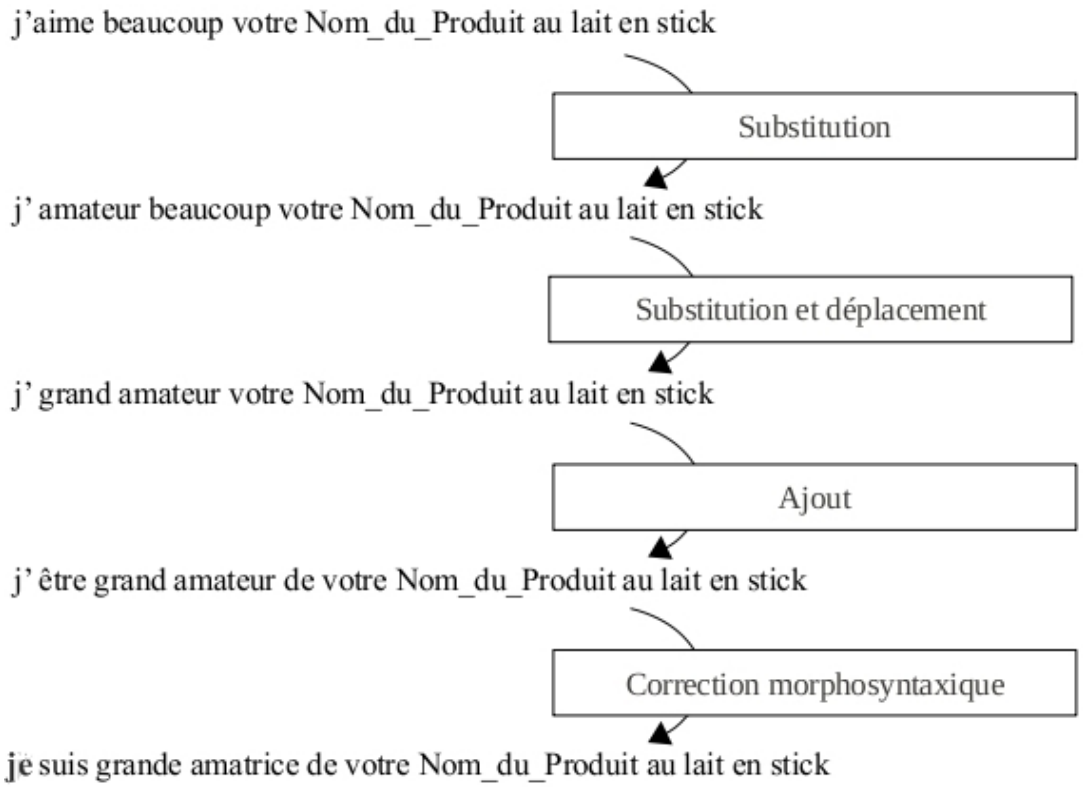


FIGURE 5.7 – Transformation *aimer* $\Longrightarrow$ « être amateur de »

5.2.3 Y plaire à X ; X être satisfait de Y

Parce qu'elles sont liées, nous regroupons la troisième structure « Y plaire à X » et la quatrième « X être satisfait de Y ». Commençons par dire que le verbe *plaire* est un conversif d'*aimer*. Le conversif est réservé aux verbes à plus de 2 valences. On doit ensuite faire une distinction entre un antonyme et un conversif. Ainsi, le conversif d'*aimer* est par exemple, *plaire, satisfaire* ainsi que les synonymes de ces derniers, alors que *détester* est un antonyme. Pour distinguer un antonyme d'un conversif à deux actants, nous avons établi un test très simple :

étant donné :

– X, V, Y tel que V est un verbe, X est sujet du verbe et Y complément du

verbe.

– S, le sens de la phrase X V Y

Un verbe W est un conversif du verbe V si et seulement si les trois conditions suivantes sont satisfaites :

(1) pour la phrase X V Y, la substitution de V par W et la permutation entre X et Y donnent lieu à une nouvelle phrase Y W X et

(2) Y W X a approximativement le sens S (autrement dit Y W X et X V Y sont paraphrases l'une de l'autre) et

(3) (X ne V pas Y) == (Y ne W pas X) est vrai.

Ainsi, *plaire* est conversif d'*aimer* puisque :

(1) « X aime Y » $\Longrightarrow$ « Y plaît à X »

(2) « X aime Y » et « Y plaît à X » sont paraphrases l'une de l'autre

(3) « X n'aime pas Y » == « Y ne plaît pas à X ».

Alors que *détester* ne l'est pas puisque la condition (2) et (3) ne sont pas satisfaites : « X aimer Y » et « Y déteste X » ne sont pas paraphrases l'une de l'autre (2) ; « X n'aime pas Y » et « Y ne déteste pas X » n'ont pas de même sens (3).

Du point de vue du traitement automatique, il est difficile de prouver la relation conversive entre deux phrases, car contrairement aux synonymes et aux antonymes pour lesquels il existe des dictionnaires, il n'y a pas de dictionnaire conversif proprement dit, à l'exception de DiCo (cf. section 1.2, page 14) qui donne aussi des conversifs d'un mot mais qui n'est pas encore exploitable. En outre, dans la pratique lexicologique, on mélange l'antonyme et le conversif comme on peut le

constater, dans le dictionnaire Le Petit Robert (2004) et Le Robert des Synonymes (2005), où *vendre* est antonyme d'*acheter*, alors qu'il s'agit aussi d'un conversif à valence 3 : « X acheter Y à Z » == « Z vendre Y à X ».

Nous venons de discuter la relation paraphrastique entre « X aimer Y » et « Y plaire à X ». La figure 5.8 (page 75) décrit le processus de transformation. Voyons ensuite comment on peut décrire la structure « X est satisfait de Y ».

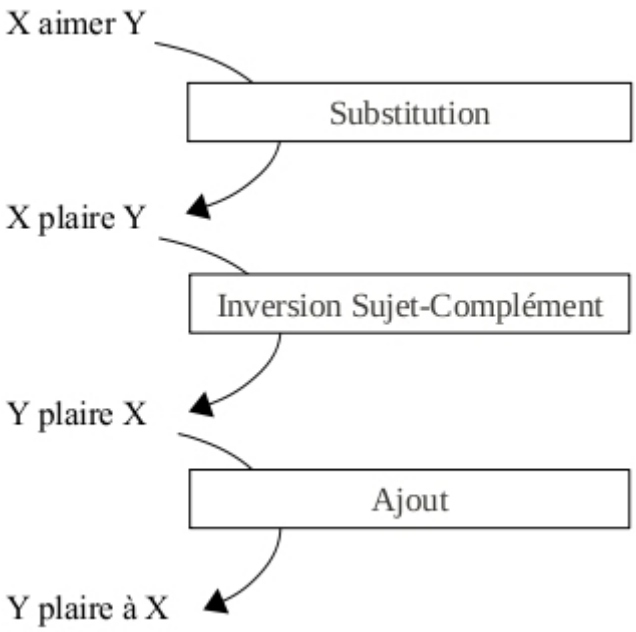


FIGURE 5.8 – Transformation « X aimer Y » $\Longrightarrow$ « Y plaire à X »

Nous avons dit plus haut que les structures « Y plaire à X » et « X est satisfait de Y » sont étroitement liées. En effet, on peut obtenir la dernière en substituant « plaire à » par un de ses synonymes *satisfaire*, suivi par une substitution de *satisfaire* par son dérivé adjectival à sens patient *satisfait* (par l'opposition à *satisfaisant* qui, lui, est un dérivé adjectival à sens agentif), nous amenant à « Y satisfait X ». Ensuite s'applique l'inversion sujet-complément qui aboutit à « X satisfait Y ». On termine par l'ajout d'une copule *être* et de la préposition *de* pour obtenir « X être satisfait de X », paraphrase de « Y plaire à X ».

Comme la relation paraphrastique est équivalente et transitive, alors si « X aimer Y » est paraphrase de « Y plaire à X » et si « Y plaire à X » est paraphrase de « X est satisfait de Y », alors « X aimer Y » est paraphrase de « X est satisfait de Y ».

Pour illustrer le passage de « X aimer Y » à « X est satisfait de Y » (figure 5.9 ci-dessous), partons de la phrase de sortie de la figure 5.8 « Y plaire à X ».

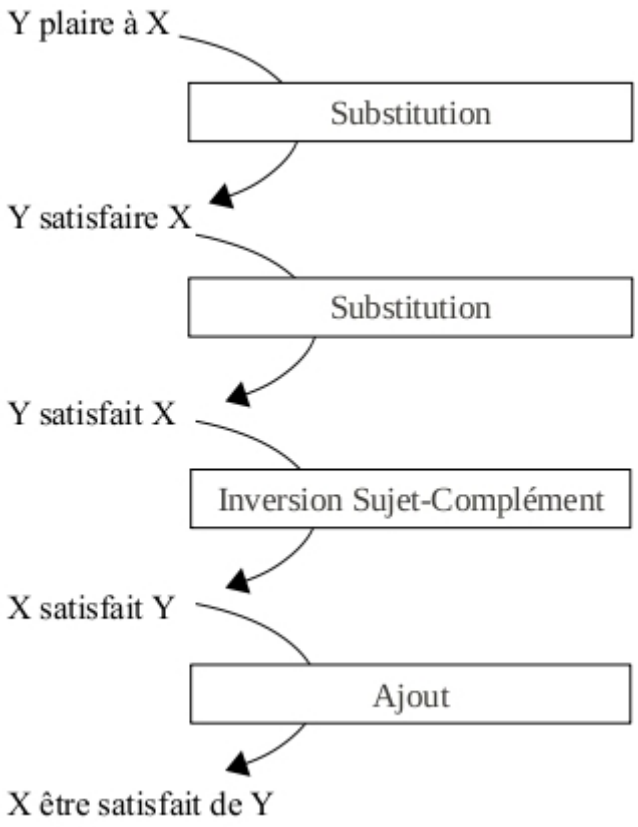


FIGURE 5.9 – Transformation « Y plaire à X » $\Longrightarrow$ « X être satisfait de Y »

Dans les exemples qui suivent, nous mettons en avant les descriptions linguistiques sur la relation paraphrastique. Ce lien paraphrastique peut être convenablement décrit comme étant un ensemble d'opérations de transformation appliquées au texte. Nous soulignons également que la relation lexicale telle que la dérivation

sémantique joue un rôle primordial dans le processus de transformation paraphrastique : c'est elle qui est l'élément déclencheur des autres changements.

Il s'ensuit que nous nous intéressons moins à d'autres questions générales dans le thème de la paraphrase, à titre d'exemple : comment remplacer un mot dans une phrase par son synonyme, métonyme, hyperonyme, négation de son antonyme, etc.

Nous fournirons des exemples avec des dérivés du mot de départ, par exemple pour *inquiéter*, on a *inquiet, inquiétude, inquiétant*. Sachant qu'à côté de ces exemples, il y a des dérivés sémantiques dont la morphologie diffère d'*inquiéter* tels que *angoissé, angoissant, préoccupant, agité, préoccupation, troubler, troublé, s'alarmer*, etc. Ceci dit, nous sommes conscients du caractère créatif et infini de la langue. Nous savons qu'une multitude de sujets liés à la paraphrase sont à résoudre. Mais concentrons-nous d'abord sur la question : comment expliquer la relation paraphrastique entre deux phrases ? quelles sont les indices les plus visibles de la paraphrase ?

5.3 Sème INQUIÉTER

Le présent sème *inquiéter* correspond au deuxième sens de l'entrée lexicale INQUIÉTER dans le Petit Robert (2004).

> **I.** 1. vx ou litté. Troubler la quiétude, la tranquillité de, ne pas laisser en repos. => agiter, troubler [...]
>
> **II.** (1645) cour. Remplir d'inquiétude, rendre inquiet (qqn). => alarmer, angoisser, ennuyer, tourmenter, tracasser, [...]. Sa santé m'inquiète. [...]
>
> **III.** S'INQUIÉTER v. pron. 1. Commencer à être inquiet. => s'alarmer [...]. 2. (1662) S'INQUIÉTER DE : se préoccuper, prendre soin, s'enquérir de. => se soucier [...]

FIGURE 5.10 – Définition d'*inquiéter* par Le Petit Robert (2004)

Nous suivons la même démarche appliquée au sème *aimer*. La figure 5.11 ci-dessous présente un extrait des phrases tirées du corpus et classées par similarité structurelle.

Le premier groupe « X inquiéter Y » représente la structure de base pour le sème *inquiéter*. Les exemples de phrases dans ce premier groupe ont une structure typique : « X Y inquiéter », tel que X est quelque chose, Y pronom personnel *me* désignant l'énonciateur, suivi du verbe *inquiéter*. La structure « X inquiéter Y » est absente, probablement du fait qu'il s'agit ici de verbatims et non de textes narratifs. Passons à la première structure paraphrastique « Y s'inquiéter de X ».

X inquiéter Y
- ce produit m'inquiète
- cela m'inquiète
- la couleur du produit m'inquiète
- la solidité du produit m'inquiète

Y s'inquiéter de X
- je m'inquiète sur la qualité de vos produits
- je m'inquiète donc de la qualité du produit
- je ne m'inquiète pas quant à la qualité du produit
- je m'inquiète à propos des frais de données

Y avoir une quiétude sur X
- votre inquiétude sur la qualité du Nom_du_Produit
- l'inquiétude du public quant à la qualité du lait en poudre
- j'avais une inquiétude quant à la qualité du produit
- j'ai une inquiétude au niveau de mon PC
- j'ai une inquiétude sur ma santé
- j'ai une inquiétude avec un produit de Nom_du_Produit

Y être inquiet de X
- elle est inquiète car elle allaite son bébé
- je suis inquiet sur la qualité du produit
- je suis inquiète par rapport à l'hygiène alimentaire
- je suis inquiète quant aux symptômes de mon chien
- je suis inquiète quant à la composition des menus
- je reste inquiète quant à mon avenir
- maman inquiète quant à la croissance de son fils

FIGURE 5.11 – Regroupement des phrases du sème *inquiéter*

5.3.1 Y s'inquiéter de X

Le deuxième groupe « Y s'inquiéter de X » représente l'emploi pronominal du verbe *inquiéter*. La différence entre l'emploi pronominal et l'emploi transitif simple du verbe *inquiéter* réside en trois changements (figure 5.12 ci-dessous) : (1) la substitution d'inquiéter par sa forme pronominale correspondante *s'inquiéter*, (2) l'inversion de Y, X, et (3) l'ajout d'une préposition *de*.

On a ici un emploi pronominal à sens réfléchi du verbe *inquiéter*, plus un complément d'objet indirect. Nombre de verbes transitifs directs ont un emploi pronominal similaire :

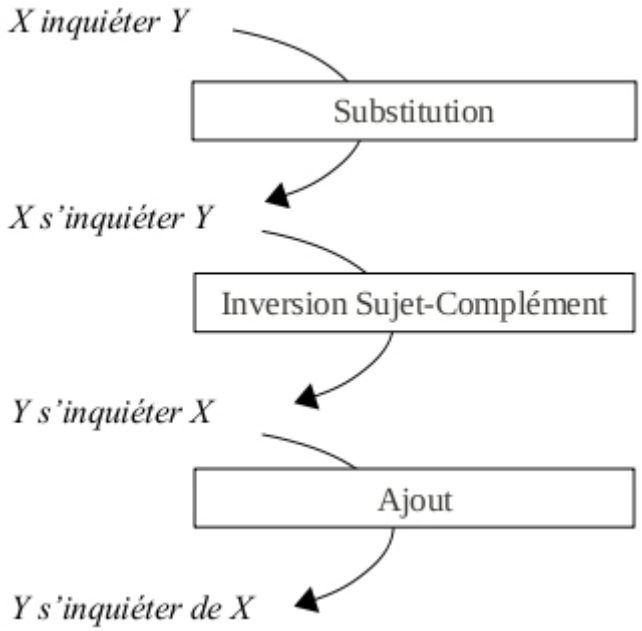


FIGURE 5.12 – Transformation « X inquiéter Y » $\Longrightarrow$ « Y s'inquiéter de X »

(a) X préoccuper Y $\Longrightarrow$ Y se préoccuper de X

(b) X fâcher Y $\Longrightarrow$ Y se fâcher de X

(c) X inspirer Y $\Longrightarrow$ Y s'inspire de X

Or, notons qu'il existe également des verbes transitifs directs dont l'emploi pronominal existant accepte mal le complément d'objet indirect en fonction d'agent :

(d) X casser Y $\Longrightarrow$ Y se casser

(e) X abîmer Y $\Longrightarrow$ Y s'abîmer

(f) X noyer Y $\Longrightarrow$ Y se noie

Il est ainsi impossible de déduire qu'une règle générale d'équivalence structurelle (structure paraphrastique) serait :

(g) X verbe Y $\Longrightarrow$ Y se_verbe préposition X

On pourrait éventuellement essayer de proposer un classement sémantique des verbes en disant que d'un côté les exemples (a), (b), et (c) sont des verbes de sentiment et acceptent la règle paraphrastique (g) ci-dessus. De l'autre côté, les verbes dans (d), (e), et (f) ne sont pas des verbes de sentiment et par conséquent n'admettent pas la règle paraphrastique (g).

Cependant, il est très difficile de réaliser à la main un classement sémantique complet de tous les verbes en vue d'un traitement automatique à large échelle. En tout cas, puisque l'on ne cherche pas à établir des règles paraphrastiques générales pour un grand ensemble de verbes, il n'est plus question de procéder au classement sémantique des verbes. Nos descriptions montrent que chaque verbe a ses propres structures paraphrastiques.

Concernant la préposition *de* dans la structure type, on peut en effet mettre à sa place sa partie du discours, d'où « Y s'inquiéter préposition X ». Or celle-ci est trop vague, nous préférons utiliser *de* pour faciliter la compréhension des lecteurs car « s'inquiéter de » représente l'emploi le plus standard par rapport à d'autres emplois : « s'inquiéter au niveau de », « s'inquiéter à propos de », « s'inquiéter quant à », « s'inquiéter sur », « s'inquiéter par rapport à », « s'inquiéter pour » entre autres.

5.3.2 Y avoir une inquiétude sur X

La règle paraphrastique permettant de relier « X inquiéter Y » à « Y avoir une inquiétude sur X » met en jeu trois principales opérations de transformation. Premièrement, le mécanisme lexical consiste à remplacer *inquiéter* par son dérivé nominal inquiétude ayant le sens de « état d'être inquiet ». Deuxièmement, le processus syntaxique, déclenché par la première substitution lexicale, permute le sujet et le complément. Troisièmement, l'opération morphosyntaxique intervient pour

assurer la grammaticalité de la phrase en ajoutant le verbe *avoir*, un déterminant *une* et la préposition *sur*. La figure 5.13 résume la suite des transformations séquentielles.

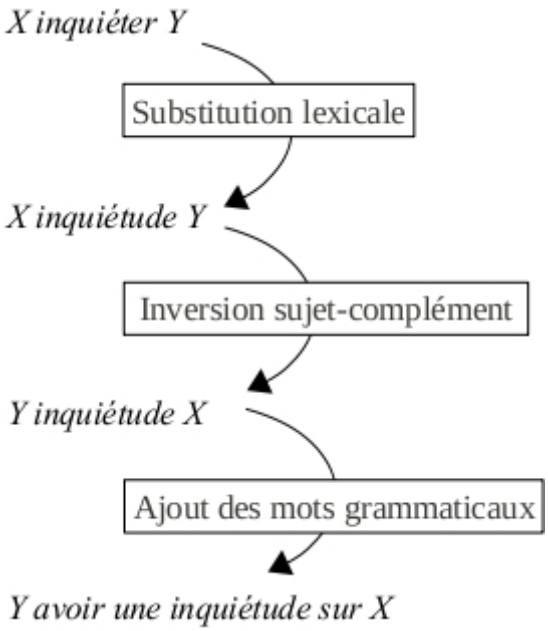


FIGURE 5.13 – Transformation « X inquiéter Y » $\Longrightarrow$ « Y avoir une inquiétude sur X »

Pour cette dernière opération, certains pourraient penser qu'il s'agit d'un mécanisme lexical et non un simple processus morphosyntaxique, car on ajoute tout de même le verbe *avoir*. Or, *avoir* ne serait qu'un verbe support dont le sens est faible, et qui, par conséquent, ne représente pas un élément essentiel dans la relation paraphrastique entre les deux structures. Ainsi, pour s'en convaincre, un simple test de suppression suffit. Étant donné deux paraphrases :

– La crise d'adolescence inquiète les Parents.

– Les Parents ont des inquiétudes sur la crise d'adolescence.

Supprimer *avoir* dans la deuxième phrase donne lieu à « Parents des inquiétudes sur la crise d'adolescence ». Un locuteur français natif pourrait sans souci

deviner le sens de la phrase. Au contraire, si on y retire *inquiétudes*, le sens de la phrase résultante « *Parents ont sur la crise d'adolescence » devient très opaque, voir incompréhensible, et s'écarte considérablement du sens de la phrase de départ « la crise d'adolescence inquiète les Parents ». Du point de vue sémantique, l'importance du lexème *avoir* n'est pas comparable à *inquiétude*, ce dernier est essentiel et le premier, lui, périphérique. Sur le plan grammatical en revanche, le verbe support *avoir* n'est pas moins important.

Cette observation suggère que, quand on travaille sur la comparaison des sens, ou par extension sur le calcul de la relation paraphrastique, il est indispensable d'identifier les mots sémantiquement pleins. Ainsi, si deux phrases partagent les mots pleins qui sont sémantiquement proches, on a affaire à une paire de paraphrases candidates. D'autres analyses plus poussées pourraient être envisagées. Si, au contraire, les deux phrases n'ont pas en commun les mots sémantiquement semblables, il y a peu de chance qu'elles soient paraphrases l'une de l'autre et par conséquent d'autres analyses plus approfondies ne seront pas nécessaires.

Dans le domaine de la paraphrase linguistique, ce que nous venons de dire est un fait puisque toutes paraphrases partagent des mots qui ont le même sens, sinon elles n'ont pas de lien sémantique. Et c'est ce trait fondamental de la paraphrase linguistique qui alimente notre formalisation de la relation paraphrastique dans le chapitre 6 Formalisation.

5.3.3 Y être inquiet de X

Partant de la structure « X inquiéter Y », le schéma de transformation en « Y être inquiet de X » ressemble à celui de la structure « Y avoir une inquiétude sur X » dans la figure 5.13. La seule différence est la substitution lexicale d'*inquiéter* par son dérivé adjectival *inquiet* ayant le sens de « qui est agité par la crainte,

l'incertitude » (Le Petit Robert, 2004). Les mécanismes enclenchés par cette substitution lexicale sont bien connus (cf. figure 5.14 ci-dessous) : inversion entre le sujet et le complément ; l'ajout d'un copule *être* et d'une préposition *de*.

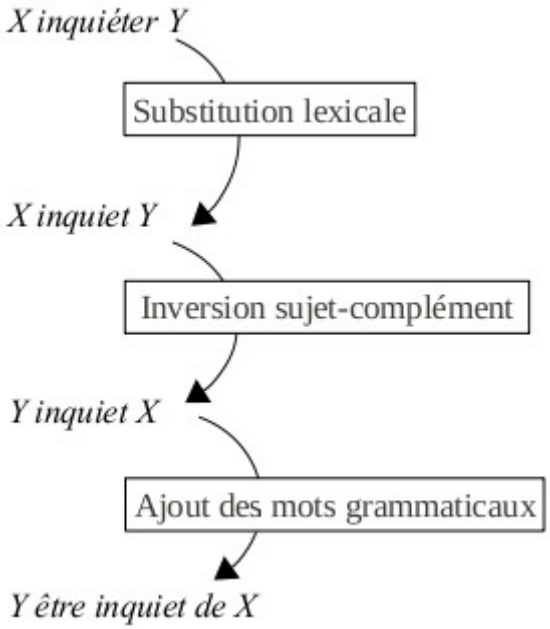


FIGURE 5.14 – Transformation « X inquiéter Y » $\Longrightarrow$ « Y être inquiet de X »

5.4 Sème DÉCEVOIR

Il s'agit ici du deuxième sens, celui le plus courant du verbe décevoir tel qu'il a été défini dans le Petit Robert 2004 :

> DÉCEVOIR [...] 2. Tromper quelqu'un dans ses espoirs ; donner une impression moins agréable que l'impression attendue => désappointer. Cet élève m'a déçu. j'ai été très déçu par ce voyage. [...] CONTR. Contenter, enchanter, satisfaire, répondre (à l'attente).

FIGURE 5.15 – Définition de *décevoir* par Le Petit Robert (2004)

La figure 5.16 ci-dessous fournit quelques exemples des structures paraphrastiques du sème *décevoir*, regroupés par similarité structurelle.

X décevoir Y ; X Y décevoir
- cela me déçoit beaucoup
- celle-ci m'a beaucoup déçu car ce produit manque de noisettes et de sauce chocolat
- La_Marque me déçoit

X être décevant ; Y trouver X décevant
- je trouve ça très décevant
- ce produits est assez décevants

déception de Y vis-à-vis de X ; déception de Y ; Y déception
- à ma grande déception, 1 des boîtes était immangeable
- ce courrier pour vous faire part de ma déception
- quelle déception, même pas une décoration sur les bûches
- je vous fais part de ma déception
- je vous fais par de ma déception vis-à-vis du Nom_du_Produit
- je vous laisse imaginer la déception d'un enfant
- quelle n'est pas ma déception au moment de déguster

Y être déçu par X
- j'ai été déçu lors de mon dernier achat
- j'ai été déçue par le côté âpre de cet article
- je me suis trouvée très déçue de ce produit
- je suis amèrement déçue et espère ne plus avoir ce genre de désagréments
- je suis déçue de leur remarque
- je suis déçu de mes deux derniers achats identiques, jambon au torchon en pack avait un problème
- je suis déçue de votre marque La_Marque
- je suis déçue par une tablette de chocolat Nom_du_Produit
- je suis déçue pour la 1ère fois depuis 25 ans, que j'achète des Nom_du_Produit à mes enfants de 35 et 32 ans
- je suis donc très déçue quant à l'éthique de votre société
- je suis très déçu des truffes noir 70
- je suis très déçue de ce produit
- je suis très déçu par le menu
- je suis une consommatrice très déçue de votre grande marque
- je suis vraiment déçu de vos pizzas surgelées Nom_du_Produit

Y être déçu ; Y être déçu X (X = Proposition)
- je suis déçue,
- je suis très déçue
- j'ai été très déçu ce jour en voulant déguster ma dernière boite achetée
- j'ai été déçue vu le prix du produit
- je suis déçu car en achetant ce paquet de céréales mon fils attendait sa carte téléphonique Nom_du_Produit
- je suis très déçu car le goût est différent
- je suis très déçue car dans mes deux derniers achats, le poulet était plein de cartilage
- je suis très déçue car dans mes deux derniers achats, le vendeur a oublié de mettre des morceaux de chocolat sur la glace
- je suis très déçue car j'ai envoyé des vœux et les personnes concernées n'ont pas reçu les SMS
- je suis particulièrement déçue de mon dernier achat
- nous avons été déçus de constater que ces céréales avaient un goût de périmé

FIGURE 5.16 – Regroupement des phrases du sème *décevoir*

5.4.1 X décevoir Y ; X Y décevoir

Nous considérons cette structure « X décevoir Y » comme structure de départ de laquelle dérivent d'autres possibles structures paraphrastiques. Cependant, la fréquence d'apparition la plus élevée de la structure « Y être déçu de X » dans le corpus pourrait laisser penser que celle-ci représente la structure de départ. Or, les statistiques n'ont rien de pertinent sur ce point. C'est le verbe *décevoir* qui est le plus porteur de sens car il permet de dériver d'autres sens noyaux de la phrase : *déçu, décevant, déception*.

Certains peuvent arguer que ce n'est pas toujours vrai. Ainsi, on a par exemple *contenter* qui vient du *content* (Le Petit Robert, 2004). Certes, *contenter* est un dérivé verbal de *content* mais nous défendons notre thèse selon laquelle la structure de base « préférentielle » d'un ensemble de structures paraphrastiques serait celle avec une forme verbale conjuguée.

Nota bene, nous ne sommes pas en train de généraliser l'idée que le verbe est toujours le plus important, ni l'élément le plus significatif dans toutes les circonstances. En effet, nous observons que dans le contexte de la paraphrase par dérivation morphologique ou sémantique, le verbe donne lieu à plusieurs dérivés pour la plupart des cas. Cela semble être la règle générale dans notre corpus. Deuxièmement, considérer les choses de cette façon offre un cadre de travail plus cohérent du point de vue théorique, et cela est indispensable car l'absence de règle générale est synonyme de manque total de point de repère pour toutes tentatives de modélisation d'un système quelconque.

En guise de conclusion préliminaire, nous postulons que pour un ensemble de structures paraphrastiques d'un sème, par exemple *décevoir*, la phrase de départ serait celle avec le verbe *décevoir* conjugué. Le verbe, considéré étroitement dans la perspective de la transformation paraphrastique, est l'élément le plus important.

En dehors de ce contexte, autrement dit les phrases prises individuellement, tout élément peut en représenter le noyau sémantique : un nom, un adjectif, un adverbe, etc.

5.4.2 X être décevant ; Y trouver X décevant

Suivant notre thèse d'existence d'opérations de transformations textuelles entre les structures paraphrastiques, nous pouvons expliquer le passage de « Y décevoir X » à « X être décevant » de la façon suivante. Premièrement, commençons par une substitution de *décevoir* par son dérivé adjectival *décevant* qui, du point de vue sémantique, assume le rôle agentif du verbe *décevoir*. Par exemple, étant donné le verbe *décevoir*, *décevant* est la forme adjectivale qui signifie « celui/ce qui déçoit ». Nous obtenons de cette première opération de transformation la phrase « X décevant Y ». Le deuxième processus déclenché par le premier consiste à ajouter une copule, donnant lieu à la phrase « X être décevant Y ». Troisièmement, on obtient « X être décevant » en procédant à la suppression de Y.

Si l'on suit nos descriptions, on se demanderait très probablement quelle serait la raison qui justifierait l'omission d'un complément Y de la phrase. En effet, on pourrait s'attendre à ce que la structure obtenue dans cette deuxième étape soit le résultat de l'ajout de deux mots pour rendre la phrase grammaticale : une copule *être* devant *décevant* et d'une préposition *pour* devant Y, d'où « X être décevant pour Y ». Quelle explication de cette disparition ?

C'est en effet la nature du texte qui permet la suppression du complément dans une telle structure : il s'agit d'un courriel. Dans le message, l'expéditeur s'adresse au destinataire en disant : « X être décevant ». Le contexte de communication permet de déduire que X est décevant, très probablement du point de vue de l'énonciateur Y. Remarquons aussi que cela n'est valable que pour les pronoms

personnels de la première personne.

En utilisant le terme « contexte de communication », nous entrons dans l'univers de la paraphrase pragmatique. Certes, nous avons dit dans la section 4.3 typologie de la paraphrase que seule la paraphrase linguistique fait l'objet de l'analyse. Or ce type de paraphrase est courant dans le corpus, nous sommes, en tant que scientifique, contraint de décrire ces faits.

Faisons, dans la figure 5.17 ci-dissous, le résumé des opérations qui transforment « X décevoir Y » en « X être décevant » ou éventuellement « X être décevant pour Y ».

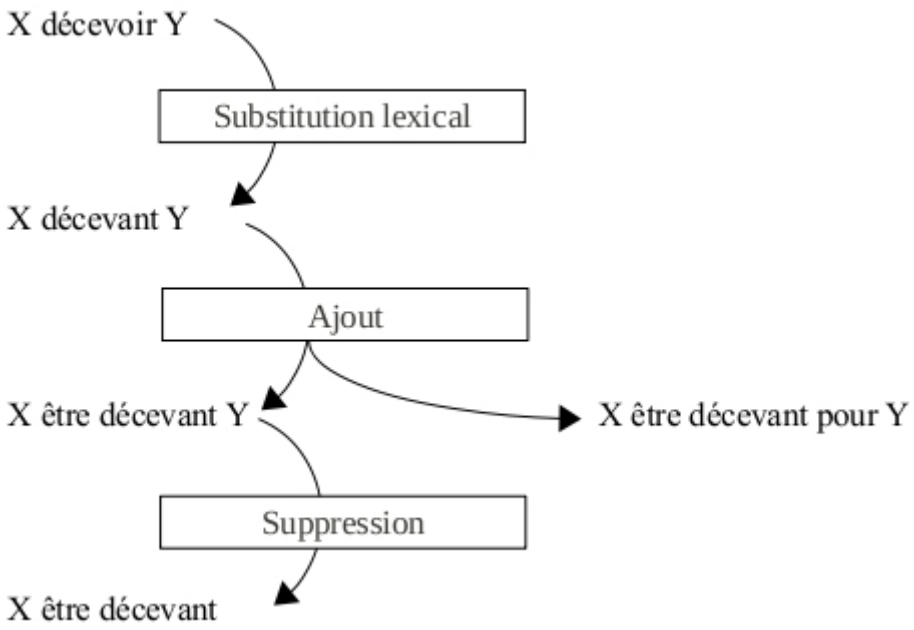


FIGURE 5.17 – Transformation « X décevoir Y » $\Longrightarrow$ « X être décevant (pour Y) ».

Dans ce même paragraphe, on a une autre structure « Y trouver X décevant » qui serait également dérivé de « X décevoir Y ». Cette fois, le verbe *trouver* est verbe principal de la phrase et non plus un simple copule *être* comme dans beaucoup d'autres exemples. Mais, poursuivons notre description qui met en avant le

mécanisme lexical de dérivation sémantique et comme nous avons dit plus haut que tout élément dans une phrase, prise individuellement, peut être un noyau sémantique, nous considérons ainsi *décevant* comme élément clé de la phrase.

Étant donné la structure de départ « X décevoir Y », pour arriver à « Y trouver X décevant », une suite d'opérations suivantes sont appliquées. D'abord, on substitue *décevoir* par son dérivé adjectival *décevant*, cela donne « X décevant Y ». Deuxièmement, le déplacement de Y en début de phrase est déclenché, conduisant ainsi à une nouvelle configuration « Y X décevant ». Enfin, pour rendre la phrase grammaticale, le verbe *trouver* est inséré après Y qui devient ainsi le sujet de la phrase. La figure 5.18 ci-dessous récapitule les étapes de transformation de « X décevoir Y » en « Y trouver X décevant ».

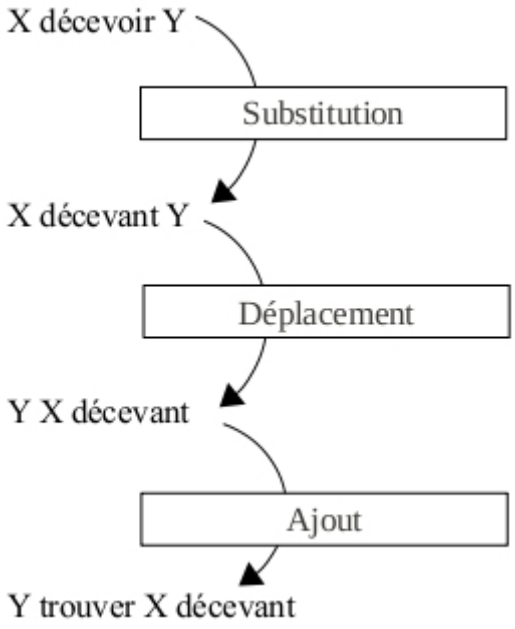


FIGURE 5.18 – Transformation « X décevoir Y » $\Longrightarrow$ « Y trouver X décevant ».

D'après nos descriptions, le verbe *trouver* est sémantiquement faible mais indispensable sur le plan grammatical. La preuve : on peut aisément supprimer *trouver*, et ajouter un mot grammatical tout en conservant le sens de la phrase (L'idée est

de montrer qu'un mot sémantiquement faible peut être facilement supprimé ou remplacé par un autre mot sémantiquement faible.). En supprimant *trouver* et ajoutant *pour* devant Y, on obtiendra par exemple « *pour Y, X décevant ». Certes, la séquence obtenue est grammaticalement incomplète mais au moins le sens de la phrase reste explicite. Ce test de suppression vient appuyer notre thèse selon laquelle, le plus important élément de la phrase isolée, du point de vue de la sémantique, n'est pas nécessairement le verbe.

Il est à noter que *trouver* a le même comportement syntaxique que les verbes comme *juger* et *considérer* qu'un grand nombre de linguistes appellent « verbe z » (ces verbes n'ont pas d'appellation spécifique). En effet, *trouver* attribue un adjectif à son objet par opposition à un verbe d'état qui attribue l'adjectif au sujet. Ainsi, il serait possible de généraliser ce sous-ensemble de structures comme contenant des phrases ayant la structure « X verbe_z Y décevant », structure caractérisée par la présence de ce « verbe z ».

5.4.3 Déception de Y vis-à-vis de X ; déception de Y ; Y déception

Le mot *déception* est un nom dérivé de *décevoir*, qui a le sens de « fait d'être déçu ; sentiment pénible causé par un désappointement, une frustration » (Le Petit Robert, 2007).

La structure « déception de Y vis-à-vis de X », parmi les deux autres « déception de Y » et « Y déception », semble être une structure paraphrastique la plus directe et complète de « X décevoir Y ». Nous décrirons d'abord comment passer de « X décevoir Y » à « déception de Y vis-à-vis de X ».

Premièrement, on remplace *décevoir* dans « X décevoir Y » par son dérivé nominal *déception*, d'où « X déception Y ». On déplace ensuite X, on le met à la

fin de la phrase, donnant « déception Y X ». On ajoute enfin : une préposition *de* devant Y pour indiquer que c'est à ce dernier qu'apparient la déception ; une préposition complexe « vis-à-vis de » devant X pour le marquer comme origine de la déception, d'où « déception de Y vis-à-vis de X ». Le passage de « X décevoir Y » à « déception de Y vis-à-vis de X » se résume en trois opérations de transformation comme le montre la figure 5.19 ci-dessous.

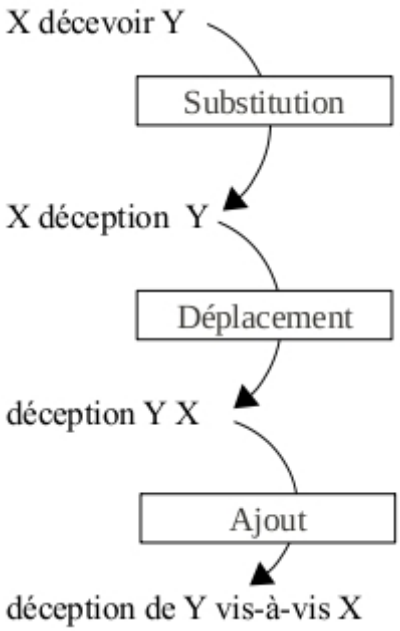


FIGURE 5.19 – Transformation « X décevoir Y » $\Longrightarrow$ « déception de Y vis-à-vis de X ».

Voyons maintenant de plus près les deux autres structures qui paraissent problématiques. La structure « déception de Y » concerne par exemple le groupe nominal « la déception d'un enfant » alors que « Y déception » sera représenté par « ma déception ». Dans le dernier groupe nominal, le déterminant possessif *ma* correspond sémantiquement au pronom personnel *je* dans « je suis déçu ». À la différence de « déception de Y vis-à-vis de X », ces deux structures sans X ne sont pas des structures paraphrastiques proprement dites de « X décevoir Y ». En effet,

on a vu dans la sous-section 5.4.2 que l'absence de Y est compensé par le contexte de communication (Y est l'émetteur du message : le client « je »). Or, l'absence de X modifie le sens de la phrase, à tel point que l'on ne sait plus l'origine de la déception. Néanmoins, on peut dire que « déception de Y » ou « Y déception » serait paraphrase de « Y être déçu » (voir prochaine sous-section 5.4.4), elle-même structure incomplète de « Y être déçu par X » (voir sous-section 5.4.5 plus bas).

5.4.4 Y être déçu par X

L'adjectif *déçu* est un dérivé de *décevoir*, qui se définit comme : « personne qui a éprouvé une déception. => dépité, désappointé. [...] être déçu par qqch, qqn, de qqch ». La structure « Y être déçu par X » est assez facile à mettre en relation paraphrastique avec la structure de départ « X décevoir Y ». Cela ressemble à un simple changement de voix active-passive. Le mot *déçu* peut être considéré comme adjectif ou participe passé. Trois opérations de transformations seront nécessaires pour accomplir le changement de « X décevoir Y » en « Y être déçu par X ».

La première opération consiste à substituer *décevoir* par *déçu*, ceci donne « X déçu Y ». Ensuite intervient l'opération de permutation entre X et Y pour obtenir « Y déçu X ». Troisièmement, l'ajout d'une copule (si *déçu* est considéré comme un adjectif) ou d'un auxiliaire *être* (*déçu* comme un participe passé), ainsi qu'une préposition introduisant le complément X en fonction d'agent, donnant « Y être déçu par X ».

Cependant, il sera plus pratique de considérer *déçu* comme un adjectif car l'emploi adjectival de *déçu* est plus courant : « Paul a l'air déçu » ; « les spectateurs déçus [...] ». La figure 5.20 ci-dessous illustre les trois étapes de transformation de « X décevoir Y » en « Y être déçu par X ».

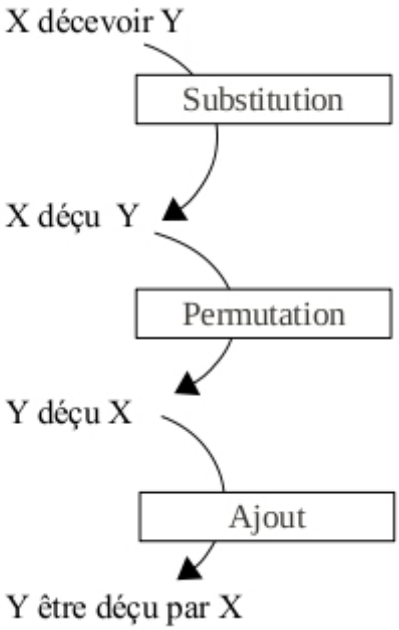


FIGURE 5.20 – Transformation « X décevoir Y » $\Longrightarrow$ « Y être déçu par X ».

5.4.5 Y être déçu ; Y être déçu X (X = Proposition)

Dans un corpus, on trouve souvent la phrase toute court « je suis déçu », une structure incomplète de « Y être déçu par X » discuté dans la sous-section 5.4.4 plus haut. Sans contexte ou analyse du discours, cette proposition est sémantiquement très incomplète. Ainsi, il n'est d'aucun sens de dire qu'on est déçu sans donner l'origine de ce sentiment, autrement dit sans préciser X qui fait l'objet de la déception. Il existe deux grands cas de figure qui explique la présence de la phrase simple « Y être déçu ».

Le premier cas, l'objet de déception X renvoie à des propositions, précédant la phrase comme dans l'exemple suivant.

– La pizza Nom_du_Produit ne contient que deux morceaux de poulet. Je suis déçu.

L'objet qui déçoit peut être la première proposition toute entière, c'est à dire, le fait que le produit n'est pas à la hauteur des espérances. Ou alors, c'est précisément « La pizza Nom_du_Produit » qui est la cause de la déception. Les deux interprétations semblent possibles. Il paraît que le problème est facile à résoudre mais en général les choses sont plus compliquées, surtout à l'égard du traitement automatique. Ainsi, considérons l'exemple ci-dessous :

– La pizza Nom_du_Produit ne contient que deux morceaux de poulet. Mes enfants n'ont pas du tout apprécié. Je suis déçu.

Il devient difficile de dire si le client est déçu par la pizza ou le fait que les enfants n'ont pas aimé le produit. Il faut une bonne analyse du discours pour retrouver le X sous-jacent dans la structure « Y être déçu ».

Dans le deuxième cas, l'origine de la déception se trouve après la phrase comme dans les exemples suivants.

- Je suis déçu. En fait, la pizza Nom_du_Produit ne contient que deux morceaux de poulet. Mes enfants n'ont pas du tout apprécié.
- Je suis déçu, la pizza Nom_du_Produit ne contient que deux morceaux de poulet. Mes enfants n'ont pas du tout apprécié.
- Je suis déçu car la pizza Nom_du_Produit ne contient que deux morceaux de poulet. Mes enfants n'ont pas du tout apprécié.
- Je suis déçu en voulant déguster la pizza Nom_du_Produit mais il n'y avait que deux morceaux de poulet. Mes enfants n'ont pas du tout apprécié.

On a la cause de la déception X qui peut renvoyer à beaucoup de choses. Comme nous ne faisons pas d'analyse de discours, nous considérons que « Y être déçu » est une structure paraphrastique de « déception de Y » ou « Y déception ». Ces trois structures constituent un sous-ensemble des structures paraphrastiques du sème « X décevoir Y » mais ne sont pas paraphrase proprement dite de ce dernier.

5.5 Récapitulatif : les idées extraites de l'analyse

5.5.1 Structure source - structures dérivées

Les descriptions dans la partie Analyse disent que pour tout ensemble de structures paraphrastiques (linguistiques), l'une de ces dernières pourrait avoir le statut de la structure dite « source » potentielle et les autres seraient « les structures dérivées directes ou indirectes ». Une structure dérivée directe est le résultat de l'application d'un nombre d'opérations de modifications textuelles sur la structure source sans passer par une autre structure dérivée. La structure dérivée indirecte, elle, ne peut provenir directement de la structure source et dérive ainsi d'une autre structure dérivée intermédiaire.

Ayant dit cela, il s'ensuit que la relation paraphrastique qu'entretiennent les

structures paraphrastiques se décrit comme étant une suite d'opérations de transformations textuelles. Par conséquent, étant donné deux structures paraphrastiques, un certain nombre de transformations ainsi que d'éventuels structures dérivées justifient la relation paraphrastique entre elles.

L'ordre de l'application des opérations est un autre point à souligner. Comme on peut le constater dans les figures d'exemples du précédent chapitre, l'ordre des transformations ne s'appliquent pas d'une façon systématique. Par exemple, le déplacement peut avoir lieu avant l'ajout ou vice-versa. Toutefois, nous sommes convaincu que la substitution lexicale et de classe grammaticale, en d'autres termes remplacer un sens par un autre sens étroitement lié, représente le mécanisme déclencheur qui intrigue et qui régit les autres modifications : c'est ce que nous avons montré à travers les exemples. Par ailleurs, certains peuvent considérer que toutes les opérations se produisent simultanément mais cela ne donnerait aucune lumière sur le mécanisme du paraphrasage.

5.5.2 Noyau sémantique de la phrase

Selon notre thèse, une structure paraphrastique dérivée peut être obtenue par l'application des opérations de transformations textuelles à la structure source et la substitution lexicale est le mécanisme porteur parmi les autres transformations. Cette substitution n'est pourtant pas aléatoire : le mot remplacé doit en effet représenter le noyau sémantique de la phrase.

Dans le contexte de notre étude de la paraphrase, le noyau sémantique de la phrase est abordé sous deux angles différents. Dans le premier cas de figure, s'agissant des paraphrases prises collectivement, le sens central de la phrase source est le verbe, car en général c'est ce qui permet de découvrir les autres dérivés sémantiques. Or, si l'on considère une phrase isolée des autres, le sens noyau de la phrase

peut faire partie d'autres parties du discours que le verbe. C'est cette observation qui joue un rôle important dans notre formalisation du modèle de la paraphrase et dans la modélisation du processus d'extraction de structures paraphrastiques.

5.5.3 Un sème = un ensemble de structures paraphrastiques

Le noyau sémantique de la phrase source discuté plus haut correspond en effet à un sème sous forme de « argument1 noyau argument2 », où *noyau* revoie par exemple à *aimer, décevoir, inquiéter*. Dans nos descriptions, un sème possède un certain nombre de structures paraphrastiques. D'une façon générale, on peut dire qu'il s'agit ici d'une analyse lexicalisée. On parle d'une lexicalisation car c'est nettement le lexique qui impose d'autres contraintes : modification lexicale déclenchant une reconfiguration structurelle.

Nous n'avons pas proposé une règle paraphrastique générale qui s'appliquerait à un grand nombre de lexies (lexème et locutions). En effet, il serait imprudent de dire qu'il y a une règle « nom agent du verbe » qui se décrit : verbe $\Longrightarrow$ « être nom_agent_du_verbe de ». Cette règle appliquée au verbe *aimer*, on obtient « être amateur de », « être fan de », « être admirateur de », etc. Prenons maintenant le verbe *décevoir*, d'où « être *déceveur de », « ?être trompeur de ». Pour la paraphrase, une règle générale exigerait beaucoup de contraintes et d'exceptions.

Par conséquent, il n'est pas faux de dire que nous essayons de trouver les structures paraphrastiques en extension, les structures paraphrastiques pour un sème donné. Mais, il ne faut pas confondre le résultat et la définition car les structures paraphrastiques en extension (toutes les formes possibles) seraient le résultat d'une définition en intension du sens donné. Cette définition correspondrait

aux règles générales (i.e. l'ensemble de postulas dans le chapitre 6 Formalisation) qui permettent de découvrir les structures paraphrastiques d'un sème donné.

5.5.4 Un modèle de la paraphrase formalisé

Ayant analysé des structures paraphrastiques et en ayant extrait les principes clés, nous proposons un modèle général de la paraphrase formalisé comme une base théorique à partir de laquelle peuvent dériver un nombre de concepts liés à la paraphrase et au traitement automatique de la paraphrase (cf. chapitre 6 Formalisation).

5.5.5 Extraction des structures paraphrastiques

Suite à l'analyse et à la formalisation de notre modèle de la paraphrase, nous voulons établir une collection de sèmes avec leurs structures paraphrastiques. Cette tâche sera accomplie par un système d'extraction basé sur les connaissances linguistiques sur la paraphrase, exposées précédemment dans la partie analyse. Cette extraction a trois grands objectifs.

Le premier sert à valider nos différentes thèses sur la paraphrase. Le deuxième objectif a pour but de donner un exemple d'une application réelle qui s'appuie sur notre formalisation du modèle de la paraphrase. Notre troisième intention est de créer une base de données de structures paraphrastiques pouvant servir à divers applications de TAL qui intègre des modules de traitement de la paraphrase.

Par ailleurs, ce module d'extraction des structures paraphrastiques répondrait aux questions soulevées dans la partie analyse. Premièrement, nous avons expliqué à la page 72 que les dictionnaires de synonymes traditionnels sont incomplets

et parfois inconsistants à l'égard du traitement automatique de la paraphrase en contexte réel (corpus de domaines spécifiques). Comme les synonymes d'un mot donné, fournis dans un dictionnaire des synonymes, ont tendance à être trop généraux, on doit envisager une mesure de validation. Ainsi, on prend par exemple les synonymes du dictionnaire, puis on les cherche dans le corpus. On valide un synonyme s'il se trouve à la fois dans le dictionnaire et dans le corpus. On obtient ainsi une nouvelle liste de synonymes plus fiable. Le corpus doit toutefois être assez grand pour que cette méthode soit efficace.

Deuxièmement, l'extraction automatique aidera aussi à retrouver divers emplois d'un verbe. En effet, le verbe *s'inquiéter* accepte une variété de prépositions : « s'inquiéter de », « s'inquiéter au niveau de », « s'inquiéter à propos de », « s'inquiéter quant à », « s'inquiéter sur », etc. Et il en va de même pour beaucoup d'autres verbes. Souvent, des emplois très courants de ces verbes, mais non attestés, ne figurent pas dans les dictionnaires. Les extraire automatiquement du corpus représente ainsi un grand avantage pour prendre en compte ces variations et cela rendrait un système de TAL plus robuste et plus complet face à un corpus très hétérogène du point de vue du registre des langues.

5.6 Conclusion

Nous avons fait une analyse de phrases tirées de notre corpus. Nous les avons regroupés en sèmes qui correspondent en effet à un ensemble de structures paraphrastiques. Entre ces structures se trouve une relation paraphrastique que nous avons décrite comme une suite d'opérations de transformations textuelles telles que substitution, ajout, déplacement, inversion (permutation), et suppression.

Ensuite, nous avons fait un rappel des grands concepts issus de l'analyse et avons envisagé des solutions pour des questions intéressantes posées dans la partie

analyse. Ce résumé sert aussi comme transition d'une analyse linguistique descriptive vers son équivalence sous forme d'un système formel que nous allons exposer dans les pages qui suivent.

Chapitre 6

Formalisation

6.1 Structures paraphrastiques comme structure source-structure dérivée

Étant donné un quintuple ⟨ **D, M, T, S, R** ⟩ et leur définition :

- **D** = $\{d_1, d_2, d_3, d_4, d_i\}$ un ensemble de mots dérivés
- **M** = $\{m_1, m_2, m_3, m_4, m_i\}$ un ensemble de mots
- **T** = {*substitution, ajout, déplacement, inversion, suppression*} un ensemble de transformations textuelles. Par exemple :
  - substitution = substituer d_i par d_j, et $d_i \neq d_j$
- **S** = $\{s_0, s_1, s_2, s_3, s_i\}$ un ensemble de structures, tel que :
  - s représente une séquence ordonnée $\langle m_1, m_2, d_j, m_3, m_k \rangle$, $j > 0$, $k \geq 0$
  - s_0 est structure source
- **R** = $\{Identité, R_D, RI_1, RI_2\}$ un ensemble de relations paraphrastiques, telle que :
  - *Identité* = s_i est la même que s_j

- R_D = La Relation paraphrastique Directe entre s_0 et s_i, $i > 0$. Par exemple :
  - s_i est le résultat de l'application de $\mathbf{T}$ sur s_0, $\mathbf{T} \neq \varnothing$, $\mathbf{T} \ni$ substitution
- RI_1 = Relation paraphrastique Indirecte entre et s_0 et s_j, j > 0, tel que
  - s_j est le résultat de l'application d'un ensemble d'ensembles de transformations textuelles $\mathbf{T}' = \{T_1,\ T_2,\ T_i\}$ sur un ensemble $\mathbf{S}' = \{s_1, s_2, s_k\}$, tel que $\mathbf{S}' \subset \mathbf{S}$, $i = k$, tout $T_i \ni$ substitution
  - L'application de $\mathbf{T}'$ sur $\mathbf{S}'$ se fait d'une façon ordonnée = $\{\langle \mathbf{T}_1, s_1\rangle, \langle \mathbf{T}_2, s_2\rangle, \langle \mathbf{T}_i, s_k\rangle\}$
- RI_2 = Relation paraphrastique Indirecte entre s_i et s_j, i et $j > 0$, $i \neq j$, s'explique par R_D ou RI_1. Par exemple :
  - si s_i peut dériver de s_0 au moyen de R_D ou RI_1, s_i est structure paraphrastique de s_0 ; et si
  - s_j peut dériver de s_0 au moyen de R_D ou RI_1, s_j est structure paraphrastique de s_0
  - alors s_i est structure paraphrastique de s_j

Un ensemble de structures $\mathbf{S}$ est dit ensemble de structures paraphrastiques si et seulement si pour toute paire $\langle s_i, s_j\rangle$ de $\mathbf{S} \times \mathbf{S}$, s_i et s_j sont reliées par au moins une relation appartenant à $\mathbf{R} = \{\textit{Identité}, R_D, RI_1, RI_2\}$.

Une propriété fondamentale de l'ensemble de structures paraphrastiques $\mathbf{S}$ est que le produit cartésien $\mathbf{S} \times \mathbf{S}$ représente la relation d'équivalence qui est réflexive (une structure quelconque est paraphrase d'elle-même), symétrique (une structure s_i est paraphrase de s_j, et vice-versa) et transitive (si s_i est structure paraphrastique de s_j, et si s_j est paraphrase de s_k, alors s_i est paraphrase de s_k).

La figure 6.1 illustre les différentes relations paraphrastiques $\mathbf{R} = \{\textit{Identité}, R_D, RI_1, RI_2\}$ définies plus haut. La figure 6.2 est une instance de la figure 6.1 mais avec des phrases comme exemples.

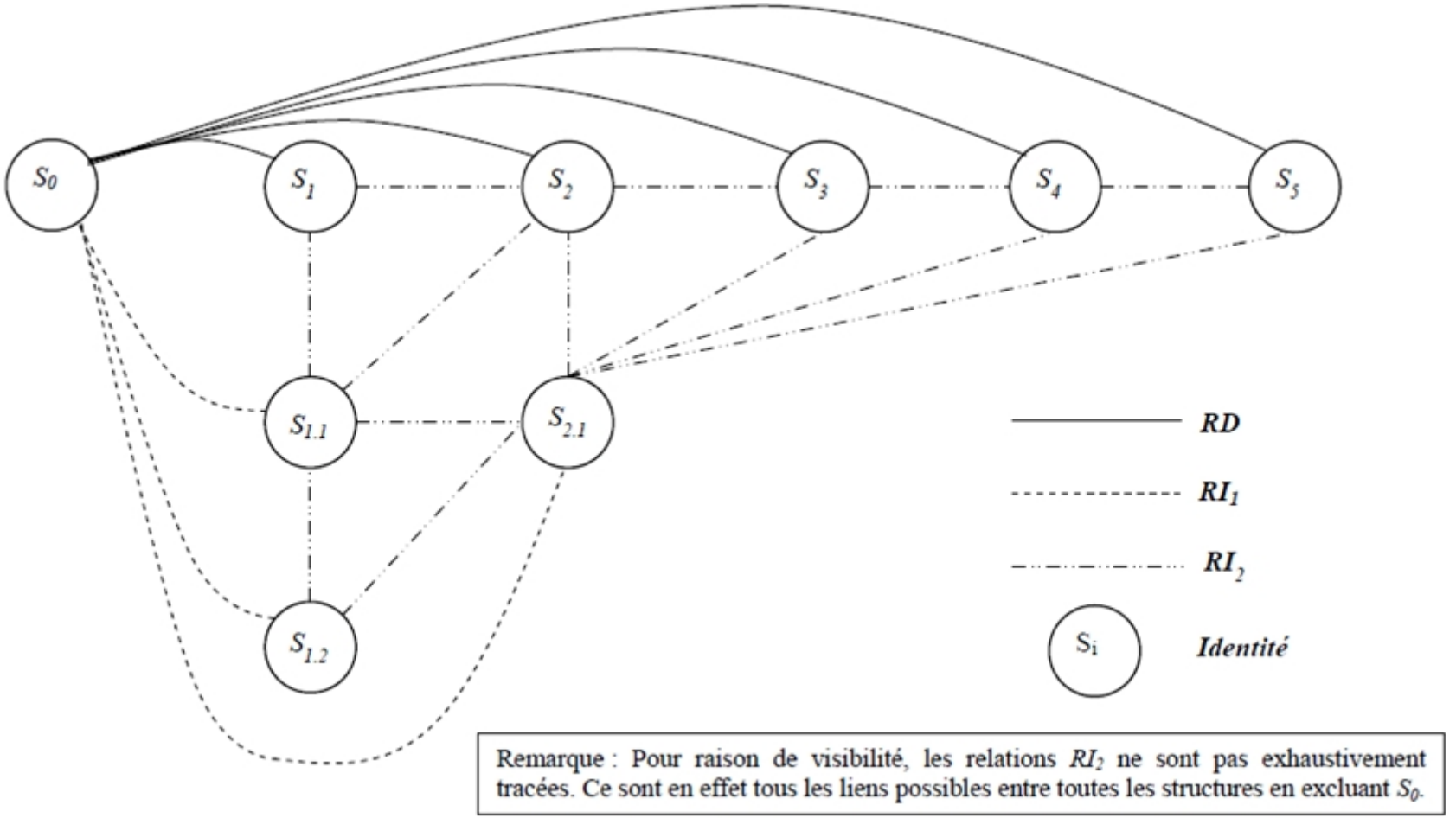


FIGURE 6.1 – Relations paraphrastiques $\mathbf{R} = \{Identité,\ R_D, RI_1, RI_2\}$

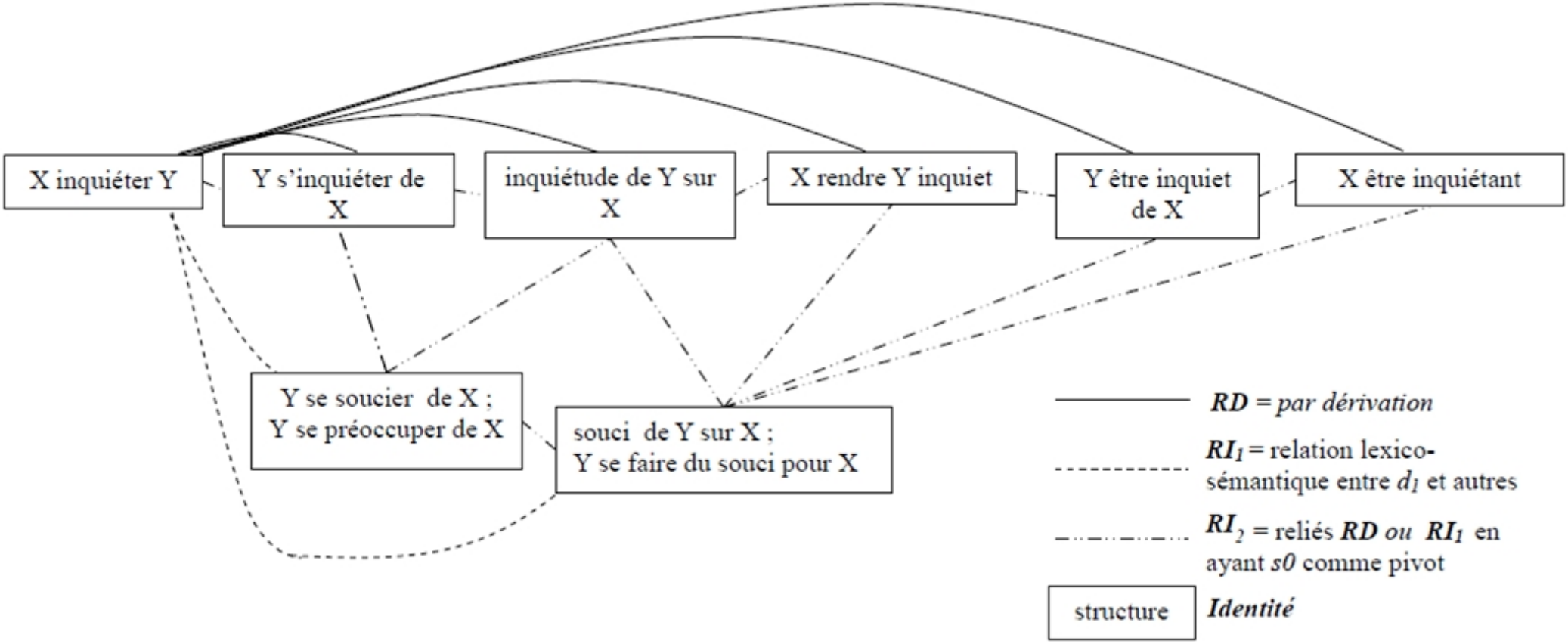


FIGURE 6.2 – Relations paraphrastiques $\mathbf{R} = \{Identité, R_D, RI_1, RI_2\}$ avec exemples

Nous allons reprendre la formalisation plus haut avec les phrases dans la figure 6.2 comme exemples.

Étant donné un quintuple ⟨ **D, M, T, S, R** ⟩ et leur définition :

- **D** = un ensemble de mots dérivés {*inquiéter, s'inquiéter, inquiet, inquiétude, inquiétant, souci, se soucier, se préoccuper*}
- **M** = un ensemble de mots {*de, rendre, sur, pour, du, X, Y*}
- **T** = un ensemble de transformations textuelles {*substitution, ajout, déplacement, inversion, suppression*}. Par exemple :
  - *substitution* = substituer *inquiéter* par *inquiétude*
- **S** = un ensemble de structures (ou séquence ordonnée de mots) {⟨*X, inquiéter, Y*⟩, ⟨*Y, s'inquiéter, de, X*⟩, ⟨*X, rendre, Y, inquiet*⟩, ⟨*inquiétude, de, Y, sur, X*⟩, ⟨*X, être, inquiétant*⟩, ⟨*Y, se soucier, de, X*⟩, ⟨*souci, de, Y, sur, X*⟩, ⟨*Y, se faire, du, souci, pour, X*⟩, ⟨*Y, se préoccuper, de, X*⟩}
- **R** = un ensemble de relations paraphrastiques {*Identité*,R_D, RI_1, RI_2}, reliant les structures paraphrastiques entre elles :
  - *Identité* : {
    [⟨*X, inquiéter, Y*⟩, ⟨*X, inquiéter, Y*⟩],
    [⟨*Y, s'inquiéter, de, X*⟩, ⟨*Y, s'inquiéter, de, X*⟩],
    ...,
    [⟨*Y, se préoccuper, de, X*⟩ ⟨*Y, se préoccuper, de, X*⟩]}
  - R_D : {
    [⟨*X, inquiéter, Y*⟩, ⟨*Y, s'inquiéter, de, X*⟩],
    [⟨*X, inquiéter, Y*⟩, ⟨*inquiétude, de, Y, sur, X*⟩],
    [⟨*X, inquiéter, Y*⟩, ⟨ *X, rendre, Y, inquiet* ⟩],
    [⟨*X, inquiéter, Y*⟩, ⟨*Y, être, inquiet, de, X*⟩],
    [⟨*X, inquiéter, Y*⟩, ⟨*X, être, inquiétant*⟩]}

On peut remarquer que les structures dérivées sont obtenues en substituant *inquiéter* par leur dérivés et en modifiant le lexique et la syntaxe comme conséquence de l'opération de substitution : *inquiéter* → *inquiet ; inquiéter* → *s'inquiéter ; inquiéter* → *inquiétant ; inquiéter* → *inquiétude ; inquiéter* → *inquiétant*. Nous avons illustré le processus de transformation dans le chapitre Analyse linguistique.

- $RI_1 = \{$
  $[\langle X,\ inquiéter,\ Y\rangle, \langle Y,\ se\ soucier,\ de,\ X\ \rangle]$,
  $[\langle X,\ inquiéter,\ Y\rangle, \langle souci,\ de,\ Y,\ sur,\ X\rangle]$,
  $[\langle X,\ inquiéter,\ Y\rangle, \langle Y,\ se\ faire,\ du,\ souci,\ pour,\ X\rangle]$,
  $[\langle X,\ inquiéter,\ Y\rangle, \langle Y,\ se\ préoccuper,\ de,\ X\rangle]\}$

  Nous qualifions cette relation paraphrastique comme étant indirect car il paraît imprudent de dire que $\langle X,\ inquiéter,\ Y\rangle$ donne lieu directement au $\langle souci,\ de,\ Y,\ sur,\ X\rangle$. Il serait plus raisonnable d'expliquer cette transition comme indirecte en ayant $\langle Y,\ s'inquiéter,\ de,\ X\rangle$ comme structure pivot. De la même manière, les trois autres paires auraient les structures pivot comme suite :
  - $\langle X,\ inquiéter,\ Y\rangle \Longrightarrow \langle \boldsymbol{inquiétude,\ de,\ Y,\ sur,\ X}\rangle \Longrightarrow \langle souci,\ de,\ Y,\ sur,\ X\rangle$
  - $\langle X,\ inquiéter,\ Y\rangle \Longrightarrow \langle \boldsymbol{inquiétude,\ de,\ Y,\ sur,\ X}\rangle \Longrightarrow \langle Y,\ se\ faire,\ du,\ souci,\ pour,\ X\rangle$
  - $\langle X,\ inquiéter,\ Y\rangle \Longrightarrow \langle \boldsymbol{Y,\ s'inquiéter,\ de,\ X}\rangle \Longrightarrow \langle Y,\ se\ soucier,\ de,\ X\rangle$
- RI_2 peut être représentée par un ensemble de paires de structures paraphrastiques, hors structure source, comme le montre la figure 6.3. Cet ensemble de structures paraphrastiques reliées par RI_2 représente en effet la relation symétrique, transitive et non identique (« non identique » pour

éviter la redondance avec la relation *Identité*).

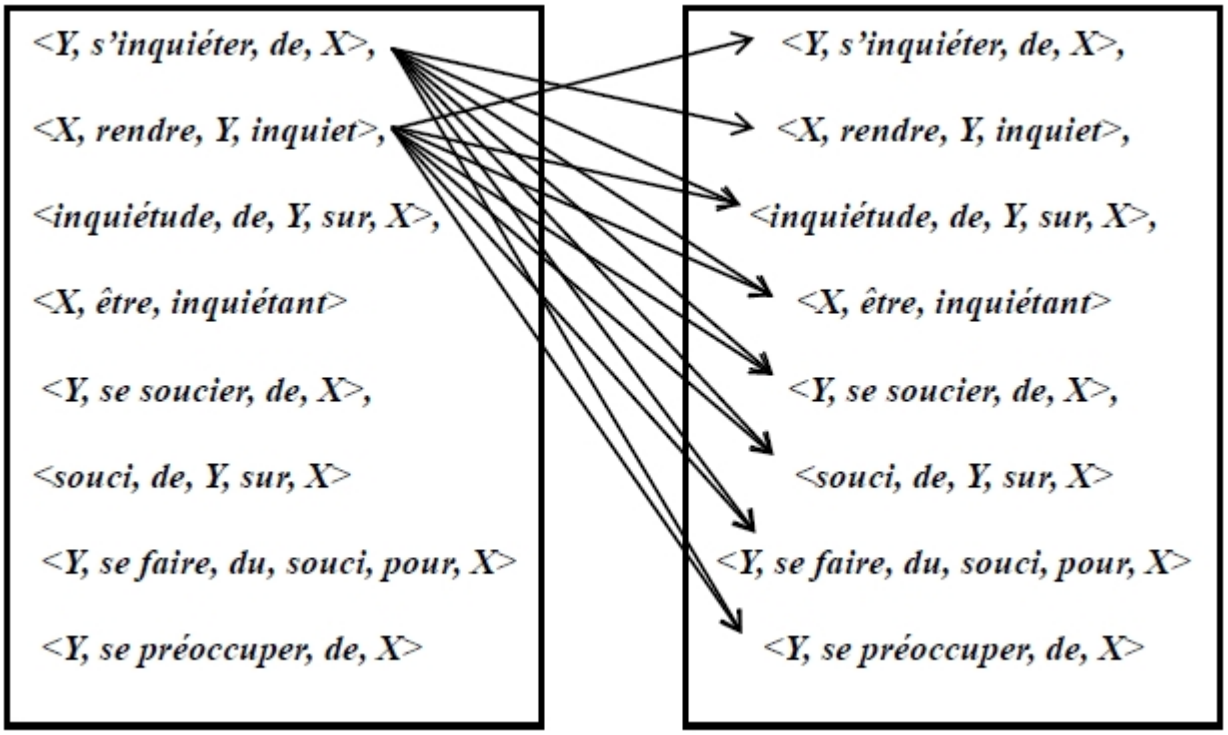


FIGURE 6.3 – RI_2 : Relation Paraphrastique Indirecte entre les structures paraphrastiques dérivées de la structure source.

Cette relation est symétrique, transitive et non identique. Pour garder une meilleure lisibilité, les liens n'ont pas été tracés en totalité (flèche partant d'une structure dans la table de gauche vers toutes les structures de la table de droite, à l'exception de sa copie).

Sachant que, pour certains, quelque unes de ces paires de structure paraphrastique pourraient être aperçues comme entretenant une Relation Paraphrastique Directe (R_D) dans le sens où $\langle$ *Y, se soucier, de, X*$\rangle$ donnerait suite au $\langle$*souci, de, Y, sur, X*$\rangle$. Ceci n'est pas faux. Or, notre formalisation met en relief les relations entre la structure source et ses structures dérivées plutôt que celles entre les structures dérivées elles-mêmes. Par conséquent, ces structures dérivées sont reliées par

la relation RI_2 qui dit tout simplement : $\langle X,\ rendre,\ Y,\ inquiet \rangle$ et $\langle souci,\ de,\ Y,\ sur,\ X \rangle$ sont toutes les deux dérivés de la structure source $\langle X,\ inquiéter,\ Y \rangle$ quelle soit directe ou indirecte, elles sont deux structures paraphrastiques.

Notre tentative de formalisation plus haut a un but pédagogique et théorique. Cette formalisation reflète notre point de vue sur les relations paraphrastiques entre des structures et sur le mécanisme de paraphrasage guidé par substitution lexicale. On entend par « pour but pédagogique et théorique » une intention de décrire le processus de paraphrasage d'une façon semi-formelle sans impliquer une mise en pratique directe de cette formalisation dans des systèmes de TAL. Pourquoi ne serait-elle convenable à l'application directe au TAL ?

Parce qu'une telle formalisation certes fournit les éléments composants de base du système de paraphrasage mais paraît complexe et probablement ingérable dans le contexte réel du traitement automatique. Elle implique en effet une recherche préalable des données linguistiques nécessaires à la complétude de ce système de paraphrasage. Sont indispensables dans cette perspective :

- Une liste exhaustive de dérivés pour un verbe donné
- Une liste exhaustive des mots qui constituent le sème
- Les règles de transformation lexicale et les règles morphosyntaxiques permettant le changement d'une structure à d'autres structures paraphrastiques
- L'ordre de l'application des opérations de transformation pour chaque passage d'une structure à l'autre.

La complexité de ces tâches est considérable. Investir dans cette direction nous semble irréalisable. Nous nous orientons par conséquent vers une définition simple d'un ensemble de structures paraphrastiques. Ensuite, les questions comment obtenir chaque composant de cette définition trouveront une réponse dans nos méthodes d'extraction automatique basées sur les connaissances linguistiques.

6.2 Formalisation en vue d'extraction automatique

6.2.1 Postulas

1. Une paire de paraphrases linguistiques sont reliées par un certain nombre de sens similaires.
2. L'une des deux peut être considérée comme phrase source, et l'autre dérivée de cette dernière.
3. Un sens dans la phrase source est plus important que les autres dans le sens où il permet de dériver des paraphrases. C'est la substitution de ce sens par un autre sens similaire qui est l'opération porteuse du processus de paraphrasage.
4. Ce sens plus important, appelé « sens source » dans la phrase source est souvent représenté par un verbe. Le « sens dérivé » dans la paraphrase dérivée, lui, peut appartenir à plusieurs parties du discours dont essentiellement verbe, nom, adjectif.
5. Le sens source et le sens dérivé sont appelés ici « prédicat » quel que soit leur partie du discours.
6. Dans une phrase, le prédicat régit d'autres unités de sens qui participent à la construction du sens de la phrase.
7. Ces sens régis par le prédicat sont appelés « arguments ». Les arguments sont essentiellement des groupes nominaux.
8. Les arguments dans la phrase source d'un côté, et de l'autre dans la paraphrase dérivée sont, hors cas d'ellipse, du même nombre. Ils peuvent être mis en correspondance par une relation lexico-sémantique telle que synonymie, hyperonymie, métonymie.

9. Un ensemble de « structures paraphrastiques linguistiques potentielles » présente alors un ensemble de phrases qui ont chacune :
   (a) un prédicat
   (b) les arguments de ce prédicat
10. La configuration ou position des arguments qui diffèrent selon la forme du prédicat est déterminée par l'hypothèse suivante. Dans notre corpus, un grand nombre de clients écrivent à l'entreprise pour un même objet. Leur expression est différente mais les sens les plus significatifs (les prédicats et arguments) resteraient similaires. Par exemple pour l'objet d'allergie :

Phrases	**Position des unités de sens**
mon enfant est allergique au gluten	enfant-allergie-gluten
le gluent déclenche une allergie chez mon enfant	gluten-allergie-enfant
intolérance au gluten chez mon enfant	intolérance-gluten-enfant
mon enfant fait l'allergie au gluten	enfant-allergie-gluten

FIGURE 6.4 – Les unités de sens les plus significatifs dans des structures paraphrastiques

Ainsi, le corpus pourra fournir l'information sur le mouvement des arguments au niveau de la structure de surface de la phrase.

6.2.2 Formalisation : ensemble de structures paraphrastiques

Nous allons maintenant formaliser l'ensemble de structures paraphrastiques en suivant la logique des dix postulas.

Au niveau plus abstrait démuni de la syntaxe et de la morphologie, un ensemble de paraphrases ressemblerait en quelque sorte à l'ensemble de groupes de sens. La figure 6.5 ci-dessous illustre un ensemble de groupes de sens.

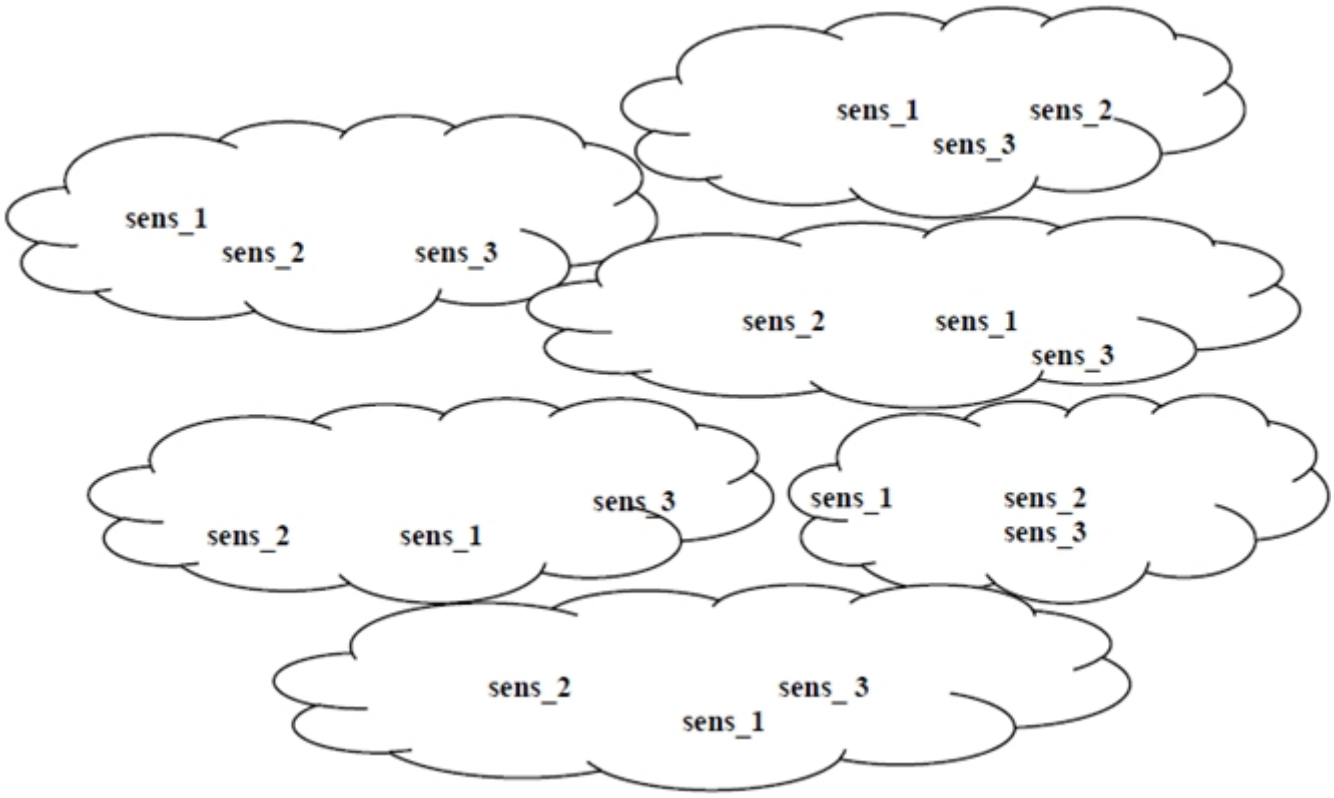


FIGURE 6.5 – Perception de l'ensemble de groupes de sens à un niveau plus abstrait, représentant un ensemble de paraphrases

La figure 6.5 bien-entendu ne ressemble à rien et c'est bien ce que nous voulons montrer. En effet, à ce niveau le plus abstrait, les sens ne sont pas ordonnés, ni tangibles, ni visibles à l'œil (Saussure les appelle « concepts » ou peut-être « la représentation sémantique » pour certains). Cette figure elle-même n'a rien d'intéressant. Or, voyons à quoi ressemblerait cette figure si chaque groupe de sens, i.e. une phrase, présente un prédicat et ses arguments (figure 6.6).

Les choses sont un peu plus intéressantes dans la figure 6.6. On y voit le lien entre le prédicat et ses prédicats dérivés, et également entre les arguments. Le plus important est dit. En effet, l'image elle-même n'est pas intéressante mais révèle déjà un principe clé à l'extraction de la paraphrase : pour extraire un ensemble

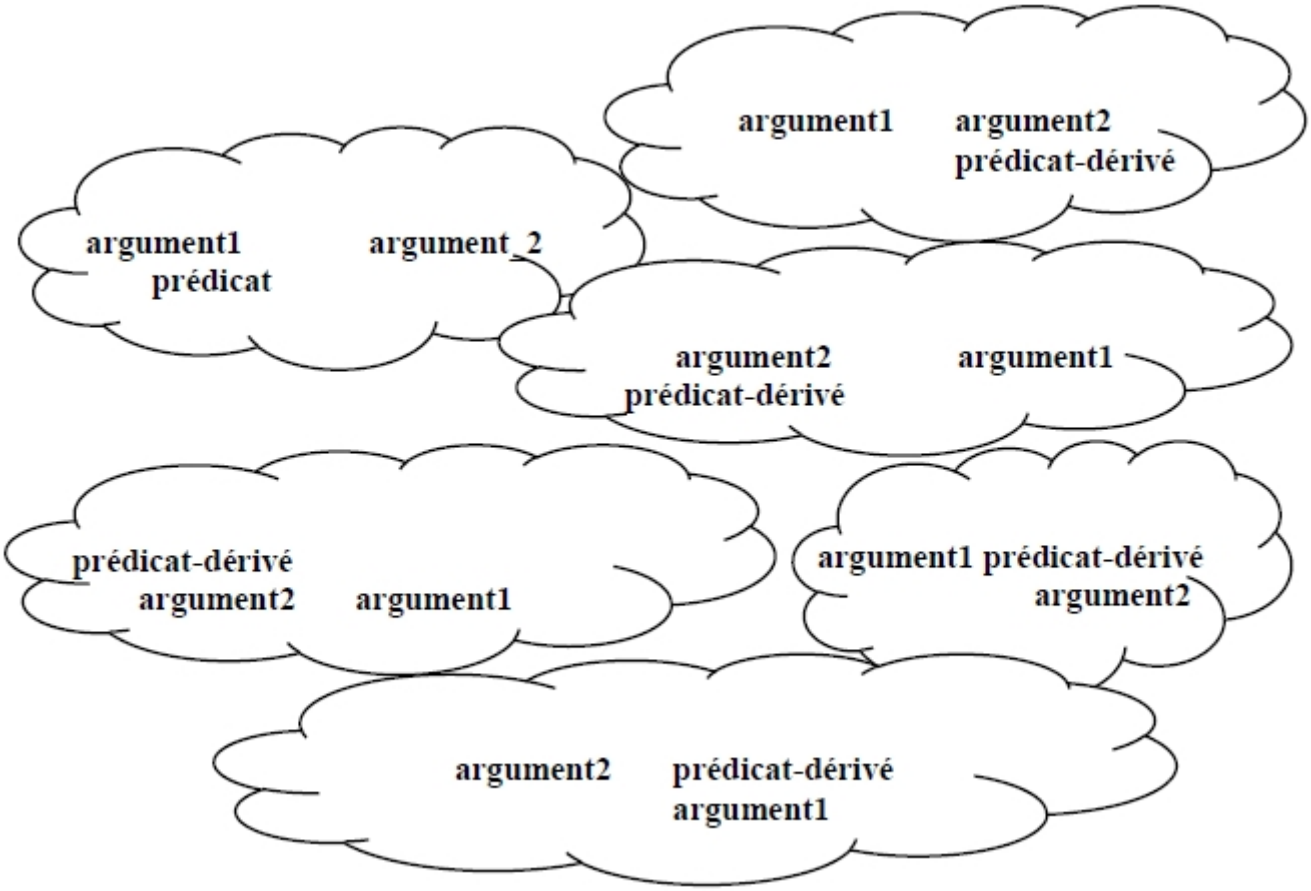


FIGURE 6.6 – Groupes de sens comme groupes de prédicat et ses arguments

de structures paraphrastiques, on devrait commencer par identifier les prédicats dérivés.

En langage naturel, ces sens sont représentés par une combinaison entre les mots (lexique), la forme de ces mots (morphologie) et la façon dont ces mots sont agencés (syntaxe). En évoquant ce trio, nous sommes maintenant sur terre, autrement dit au niveau de surface, le niveau que le TAL peut réellement exploiter. Considérons la figure 6.7 montrant, au niveau de surface, les paraphrases possibles qui correspondent aux groupes de sens dans la figure 6.6.

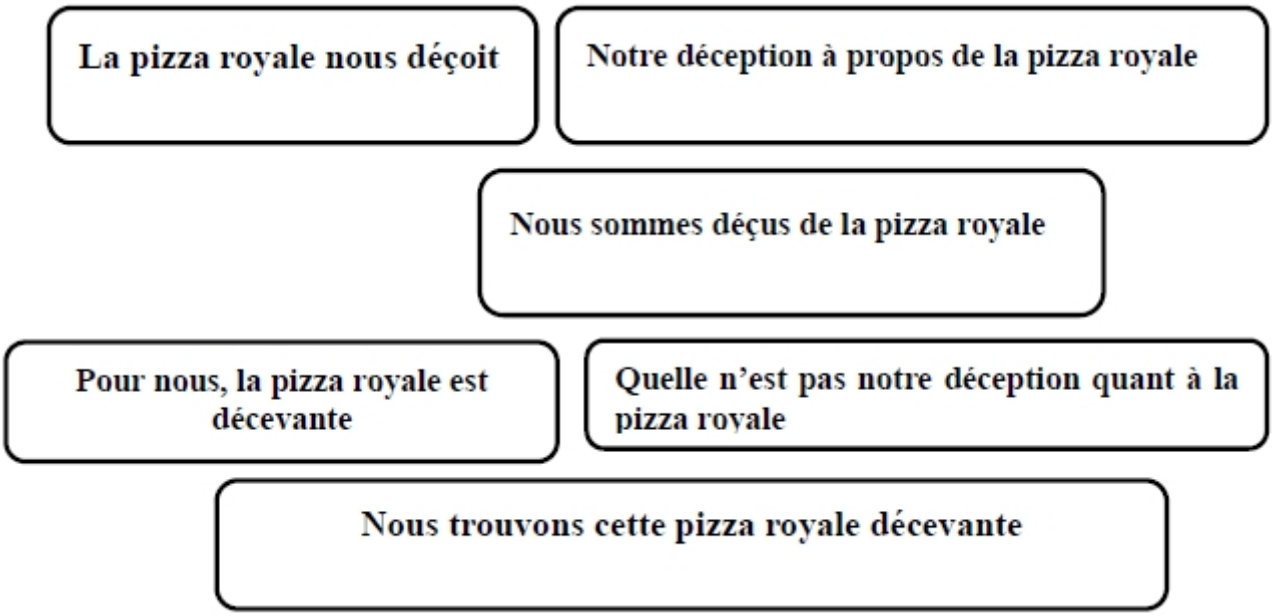


FIGURE 6.7 – Phrases de surface possibles des groupes de sens dans la figure 6.6

Nous pouvons maintenant procéder à la fusion de la figure 6.6 et la figure 6.7. En d'autres termes, il s'agit d'un mapping de la phrase vers la structure prédicat-argument.

La **Argument1(pizza royale) Argument2(nous) Prédicat(déçoit**)
Argument2(Nous) sommes **Prédicat-dérivé(déçus)** de la **Argument1(pizza royale)**
Pour **Argument2(nous)**, la **Argument1(pizza royale)** est **Prédicat-dérivé(décevante**)
Argument2(Nous) trouvons cette **Argument1(pizza royale) Prédicat-dérivé(décevante)**
Argument2(Notre) Prédicat-dérivé(déception) à propos de la **Argument1(pizza royale)**
Quelle n'est pas **Argument2(notre) Prédicat-dérivé(déception)** quant à **Argument1(la pizza royale)**

FIGURE 6.8 – Phrases de surface étiquetées des fonctions prédicat-argument

La formalisation dans la figure 6.8 s'opère au niveau de la phrase (syntaxe + lexique + morphologie) qui ne fait que refléter un niveau plus abstrait : la sémantique représentée ici par le prédicat et ses arguments.

Cependant, ces structures devraient être plus générales en vue du traitement automatique. Nous procédons ainsi à la suppression des articles (à l'exception des pronoms possessifs en raison de leur importance sémantique), à la déflexionnalisation (la transformation des formes flexionnelles en forme canonique non fléchie). Les autres éléments comme les verbes et propositions sont préservés car ils participent à la configuration de la phrase. La figure 6.9 donne une formalisation des structures paraphrastiques sous forme de prédicat-argument.

Note :

– « Argument1 » est noté « arg1 »

– « Argument2 » est noté « arg2 »

– « Prédicat » et « Prédicat-dérivé » sont toutes deux notées « préd ».

– La valeur de l'argument1 est remplacé par X, l'argument2 par Y.

arg1(X) arg2(Y) préd(décevoir)
arg2(Y) être préd(déçu) de arg1(X)
pour arg2(Y) arg1(X) être préd(décevant)
arg2(Y) trouver arg1(X) préd(décevant)
arg2(Y) préd(déception) à propos de arg1(X)
quel ne être pas arg2(Y) préd(déception) quant à arg1(X)

FIGURE 6.9 – Formalisation : ensemble de structures paraphrastiques sous forme de prédicat-argument

Il est maintenant temps d'adresser la question comment extraire ces structures paraphrastiques en TAL. La partie III Implémentation propose une méthodologie d'extraction des structures paraphrastiques guidée par les résultats de nos travaux d'analyse et la formalisation présentés plus haute. Les différentes techniques seront basées sur les connaissances linguistiques.

Troisième partie

Implémentation

Introduction

Le raison principale de l'existence de cette partie implémentation est de donner une réponse à la question soulevée par nos analyses linguistiques et leur formalisation dans la partie précédente et qui pourrait être utile en traitement automatique des langues. Le système d'extraction des structures paraphrastiques que nous allons présenter n'est qu'une mise en pratique possible de nos travaux théoriques précédemment exposés. Nous ne prétendons pas avoir créé une application en vue d'une commercialisation immédiate, ou une application opérationnelle sur des millions de phrases. Loin de là, dans notre perspective, nous visons plutôt à constituer un système informatique de démonstration du point de vue du linguiste.

Nous nous sommes efforcé de réaliser un système de traitement en cascade, pas vraiment simple, mais dont la logique est facile à comprendre. Nous mettons à disposition des lecteurs, et surtout de nos collègues chercheurs ou étudiants, tous les codes du programme et les ressources utilisées dans l'annexe B, parce que nous voulons que ce système soit avant tout compréhensible, transférable, adaptable à d'autres contextes. Cependant, il n'est pas obligatoire de suivre nos méthodes à la lettre pour obtenir des résultats similaires. Comme plusieurs chemins peuvent mener à la même destination, il en va de même pour la programmation : on peut recourir à d'autres méthodes de traitement ou à d'autres ressources linguistiques, selon bien entendu les conceptions théoriques du chercheur.

Certes, c'est un système compact car il comprend seulement une centaine de pages de codes de programme, y compris les ressources linguistiques utilisées, mais il est assez complet. En effet, il comprend toutes les composantes nécessaires au bon fonctionnement du programme du point de vue à la fois technique et théorique. Ce système est réellement opérationnel pour un ensemble de quatre-vingt-douze phrases (annexe B.1).

Nous allons, dans un premier temps, esquisser le schéma général de notre système d'extraction des structures paraphrastiques en mettant en relief ses composantes principales. Ensuite, nous expliquerons un par un comment fonctionnent ces modules de traitement. Finalement, en guise de démonstration, nous allons tracer le flux de traitement sur quelques exemples dans la section « Démonstration point par point ».

Chapitre 7

Extraction des structures paraphrastiques

7.1 Processus d'extraction

En quelques lignes, le processus d'extraction de structures paraphrastiques dans la figure 7.1 se résume comme suite. En données initiales du système il y a notre corpus. Le taggueur Labelgram dans le module de traitement 1, qui a déjà opéré sur des corpus avec succès, découpe les phrases en mots. Le module 2 prend comme entrée des phrases coupées en mots et rend en sortie des phrases candidates constituant des structures paraphrastiques pour un sème donné. Ces phrases candidates sont passées au module 3 qui les découpe en « groupes de mots » [1] ayant comme tête des noms, pronoms, adjectifs ou verbes. Le module 4 prend ces phrases constituées en chunks comme données d'entrée. Seront supprimés dans ce module 4 les

1. Nous évitions le terme syntagme car son concept ne s'applique pas dans notre travail. Le syntagme verbal, par exemple, peut être composé de verbe comme tête et de syntagme nominal. Notre approche s'inspire plutôt des arbres de dépendance de Lucien Tesnière. Ainsi, le groupe nominal complément d'un verbe constitue un groupe de mots non incorporé dans le verbe.

mots sémantiquement vides se trouvant dans les chunks. Une fois nettoyées, les phrases seront transformées en structures prédicat-argument dans le module 5.

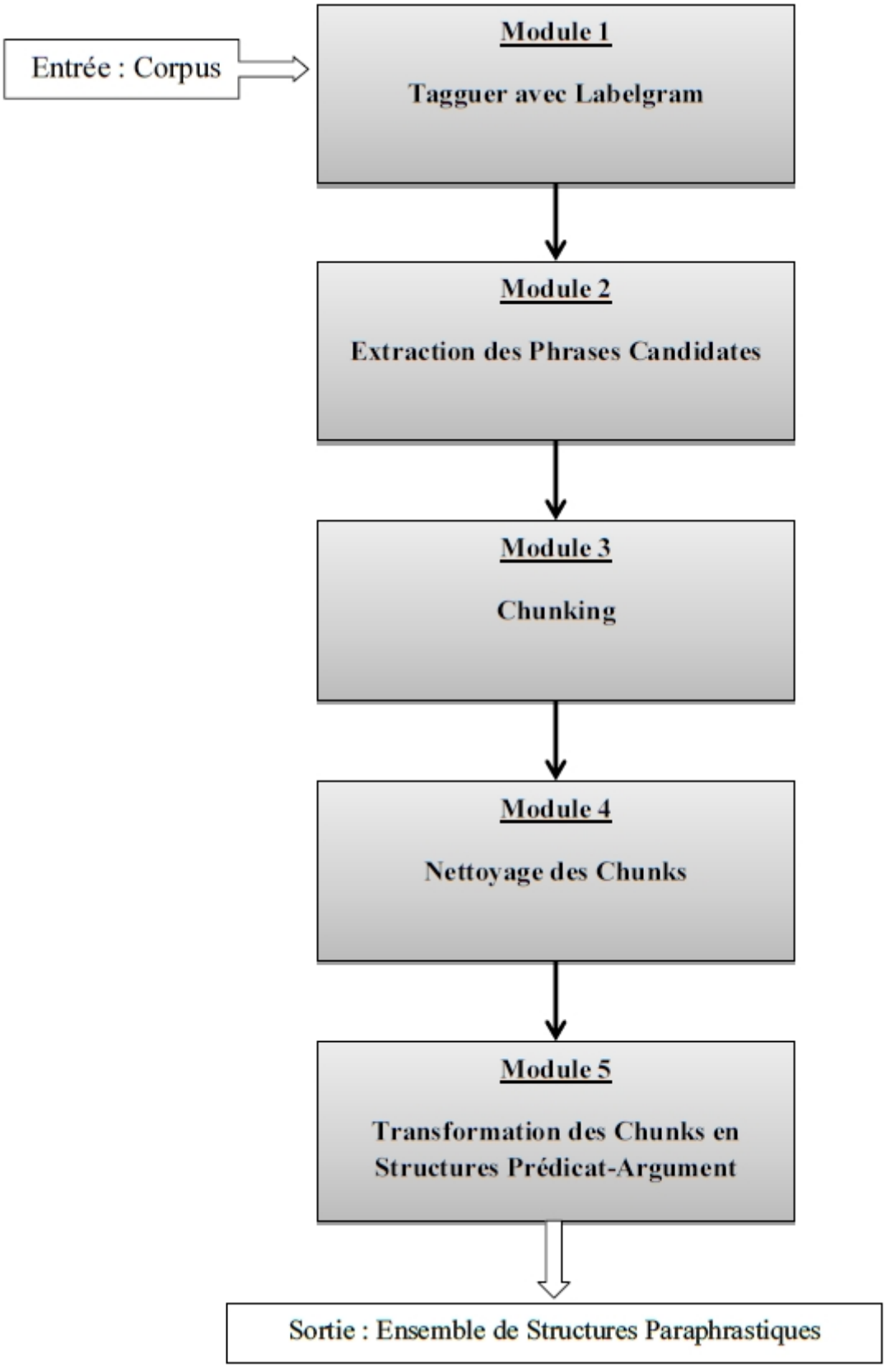


FIGURE 7.1 – Processus d'extraction de structures paraphrastiques et ses composantes

7.2 Composantes du système

7.2.1 Labelgram

Labelgram est un système de désambiguïsation morphologique de textes, dont le POS taggueur selon le terme courant, a été développé au Centre de Recherche Lucien Tesnière. Le système est fondé sur le principe de linguistique microsystémique (Gentilhomme, 1985 ; Cardey, 1987) selon lequel un système peut être segmenté en plusieurs systèmes plus petits qui influent les uns les autres. Labelgram fait l'objet d'implémentation pour plusieurs langues : le français (Cardey & Greenfield, 2005 ; El Harouchy, 1997) ; l'anglais (Birocheau, 2000) ; l'allemand et l'espagnol (Morgadinho & Venet, 2003).

Pour notre modèle d'extraction de structures paraphrastiques, nous partons de l'hypothèse que le corpus a été taggué par Labelgram. En effet, notre corpus est un sous-ensemble du corpus utilisé dans le projet Classificatim (Cardey et al., 2006), un système de classification de sèmes employé à large échelle dans le domaine de l'industrie agroalimentaire. Classificatim a connu un taux de précision de l'ordre de 84 % sur du texte brut et 99 % sur du texte normalisé. Puisque Labelgram y a accompli un taggage avec succès et que des publications assez complètes ont été faites sur cet outil, il serait nécessaire selon notre point de vue de mettre l'accent sur les modules de traitement qui peuvent en dériver. La figure 7.2 ci-dessous fournit quelques-unes des quatre-vingt-douze phrases que nous avons étiquetées manuellement et avec l'aide d'expressions régulières (cf. annexe B.1 pour la totalité des phrases). Ces phrases constituent les données d'entrée du module de traitement 2.

```
(1) sentence([pron('je'), v('suis'), adv('très'), adj('déçue'),
    prep('de'), det('ce'), n('produit')]).
(2) sentence([pron('je'), v('utilise'), adv('habituellement'),
    det('ce'), n('lait')]).
(3) sentence([det('mes'), n('chiens'), adv('ne'), v('mangent'),
    adv('que'), pron('ça')]).
(4) sentence([pron('je'), v('suis'), det('une'), adv('très'),
    adj('grande'), n('consommatrice'), det('de'), pron('vos'),
    n('barres')]).
(5) sentence([prep('après'), det('les'), aux('avoir'),
    v('consommés'), det('à'), adj('fameux'), n('repas'),
    pron('ils'), aux('ont'), v('eu'), det('des'), n('rillettes')]).
(6) sentence([det('mes'), det('trois'), n('chats'), v('manger'),
    det('nos'), n('restes'), det('de'), n('table')]).
(7) sentence([pron('je'), v('suis'), det('une'), adj('grande'),
    n('consommateur'), det('de'), n('Nom_du_Produit')]).
(8) sentence([pron('ce'), v('est'), det('une'), n('lait'),
    pron('que'), pron('je'), v('utilise'), prep('depuis'), det('1'),
    n('mois')]).
(9) sentence([pron('je'), v('suis'), det('une'), adj('fervente'),
    n('consommatrice')]).
(10) sentence([det('les'), n('Français'), v('mangent'), adv('trop'),
    det('de'), n('sel')]).
(11) ...
```

FIGURE 7.2 – Exemples de phrases d'entrée initiales au système d'extraction

7.2.2 Extraction des phrases candidates

Le module de traitement 2, extraction de phrases candidates, prend comme données d'entrée l'ensemble des phrases tagguées produites par le module 1. Comme son nom l'indique, cette deuxième phrase consiste à sélectionner les phrases candidates qui représentent un ensemble de structures paraphrastiques d'un sème donné. Ceci implique que les phrases sélectionnées dépendent du sème auquel on s'intéresse.

Le processus d'extraction des phrases candidates se fait sans difficulté. Etant donné :

- un ensemble de phrases étiquetées comme donnée d'entrée :
  K = $\{phrase_1, phrase_2, phrase_3, phrase_i\}$, tel que $phrase_i$ présente une liste de mots $\langle mot_1, mot_2, mot_3, mot_i\rangle$
- un ensemble de mots dérivés du sème décevoir : D = $\{dérivé_1, dérivé_2, dérivé_3, dérivé_i\}$;

Le système parcourt les phrases dans K une par une, ne garde que celles qui contiennent au moins un dérivé dans D. Autrement dit, une phrase est une phrase candidate si et seulement si au moins un mot dans $phrase_i$ est le même que $dérivé_i$ dans D ($mot == dérivé$). Le traitement donne finalement un nouvel ensemble de phrases candidates destinées au troisième module de traitement.

Il s'ensuit que ce module de traitement inclut un certain nombre de données linguistiques prêtes à être exploitées. La plus importante de toutes est sans doute la liste des dérivés qui nous intéresse. Rappelons que le terme dérivé employé ici comprend les mots dérivés ou reliés par une relation sémantique comme la synonymie, l'homonymie, la métonymie, l'antonymie entre autres. Le choix des types de mots à inclure dans la liste dépend des idées, des buts du concepteur du système. Ce dernier peut ne prendre en compte qu'un certain type de dérivés

et leurs synonymes s'il vise une qualité et une exactitude des traitements. Par exemple, on peut s'intéresser aux :

- dérivés nominaux ayant le sens d'agent du verbe (*fabriquer* → *fabricant, producteur, constructeur*)
- dérivés adjectivaux en –able signifiant « qui peut être participe_passé_du_verbe » (*manger* → *mangeable, consommable*).
- Dérivés nominaux ayant le sens de « état d'être verbe » (*décevoir* → *déception*)
- dérivés adjectivaux dans le sens d'agent du verbe signifiant « qui présente_indicatif_du_verbe » (*dégoûter* → *dégoûtant*).
- synonymes.

Sinon, en visant la quantité, il peut recourir à des synonymes du mot de départ, à des dérivés d'un dérivé et ainsi de suite. Cependant, plus on s'éloigne du mot de départ, plus on court le risque d'avoir à la fin du traitement des paraphrases assez lointaines. En d'autres termes, l'ensemble des structures paraphrastiques final sera peu fidèle car il contiendra trop de structures paraphrastiques possibles mais trop lointaines. Passons maintenant à la question que nous considérons être au cœur de ce module de traitement 2 : comment obtenir cette liste de dérivés.

Pour le français, il n'existe pas de dictionnaire de dérivés complet, prêt à être exploité tel quel. Il y a par exemple, le Verbaction, dictionnaire de noms dérivés en –ion (Hathout, Namer, & Dal, 2002). En outre, il existe des dictionnaires généraux intéressants tels que WiktionaryX (Sajous, Navarro, Gaume, Prévot, & Chudy, 2010), une version plus légère du Wiktionary pour le français ; le Wolf, une version de Wordnet (Fellbaum, 1998) pour le français. Ces dictionnaires proposent des mots dérivés, des mots reliés par une relation sémantique mais d'une façon non systématique, ce qui provoquerait des incohérences en cas d'usage direct. Le WiktionaryX, par exemple, est une ressource lexicale riche mais, étant une copie

de la ressource libre Wiktionnaire (http ://fr.wiktionary.org) que n'importe qui peut modifier, il serait peu prudent de l'exploiter tel quel. Une autre possibilité existe ; (Kampeera & Cardey, 2012) propose une méthodologie pour extraire les mots dérivés morphologiquement et sémantiquement du WiktionaryX en passant par la définition des mots et leur étymologie.

Extraire les dérivés pour construire un dictionnaire de dérivés complet est une tâche épuisante. Nous voulons bien entendu le faire mais devons garder les pieds sur terre vu notre savoir-faire et le temps dont nous disposons. Ainsi, nous ne pourrons que construire un modèle de traitement plus abstrait pour bien montrer la méthodologie d'extraction des structures paraphrastiques. Pour ce faire, supposons que nous ayons la liste de dérivés suivante pour le sème *décevoir*. Cette liste doit normalement comprendre toutes les formes fléchies d'un dérivé pour assurer qu'aucune phrase candidate ne passera à côté.

```
décevoir, déçoit, déçois, décevons, décevez, déçoivent,
déçu, déçue, déçus, déçues, décevant, décevante,
décevants, décevants, déception, ennui, désappointement,
désappointer, déconvenue, désenchantement, désillusion,
chagrin, insatisfaisant, insatisfaisante, frustrer,
frustré, frustrés, frustrée, frustrées, frustrant,
frustrantes
```

FIGURE 7.3 – Liste de dérivés pour le sème *décevoir*

7.2.3 Chunking

Le Chunking consiste à réorganiser la phrase tagguée en plusieurs plus petites structures. Selon le but du concepteur, le chunker peut distinguer un groupe verbal, nominal, adjectival, adverbial, voire prépositionnel. Un chunker n'a pas la même fonction qu'un analyseur syntaxique puisque sa tâche n'est pas d'élucider

les relations syntaxiques entre les groupes de mots composant la phrase. D'une façon générale, un chunker peut faire partie d'un analyseur syntaxique.

Ce troisième module de traitement reçoit les phrases candidates tagguées produites par le deuxième module. Concrètement, il comprend l'algorithme et les règles linguistiques de découpage des phrases en chunks. Nous pensons que les groupes nominaux, verbaux et adjectivaux sont les plus importants pour mettre en évidence des structures prédicat-arguments. De même, nous voulons commencer par le plus simple composé seulement de quelques mots pour éviter que notre travail ne prenne des proportions non souhaitables. Nous nous inspirons en effet du principe de l'adaptabilité. Ainsi, on cherche d'abord à construire un système limité mais opérationnel pour tester nos idées et pour en découvrir les composantes principales. Ensuite, on peut améliorer le système en fonction des besoins pour qu'il soit capable de traiter des données à plus grande échelle.

Ci-dessous les exemples des grammaires (règles) de chunking. Les grammaires complètes se trouvent dans l'annexe B.9.

```
chunk(gv, [aux(_), adv(_), v(_)]).
chunk(gv, [aux(_), v(_)]).
chunk(gv, [pron('me'), aux(_), v(_)]).
chunk(gn,[det(_), n(_), adj(_), prep(_), det(_), n(_)]).
chunk(gn,[det(_), adj(_), adj(_), n(_), adj(_)]).
chunk(gn,[det(_), n(_), prep(_), det(_), n(_)]).
chunk(gprep, [prep('vis-à-vis'),prep('de')]).
chunk(expression, [v('faire'),n('part'),prep('de')]).
```

FIGURE 7.4 – Exemples de règles de chunking

Ainsi, la première règle se lit « le group verbal `gv` est composé d'un auxiliaire, d'un adverbe et d'un verbe ». La règle numéro 5, elle, stipule qu'un groupe nominal `gn` est constitué d'un déterminant, de deux adjectifs, d'un nom et d'un adjectif. Pour la huitième règle, il s'agit d'une locution verbale « faire part de ».

7.2.4 Nettoyage des chunks

Les données produites par le module 3 comprennent les phrases candidates découpées en chunks. Avant de les transformer en structures prédicat-argument, il est nécessaire de procéder au « nettoyage » qui consiste à enlever les mots sémantiquement vides comme les déterminants et certains adjectifs et adverbes car nous nous intéressons plutôt à la structure prédicat-argument et non à la structure de chaque chunk en particulier.

Cette étape de traitement comprend également des règles de décomposition des chunks dans le cas où les mots-clés (mots dérivés) font partie d'un chunk mais ne sont pas la tête du chunk. Ce travail a pour but d'extraire les prédicats car ceux-ci sont, de notre point de vue, l'élément noyau des structures paraphrastiques. Nous fournissons ci-dessous les règles de suppression des éléments non-essentiels et des règles de décomposition (cf. annexe B.14 pour l'ensemble des règles).

```
(1) transform_to(gn([det(_Det1),n(N1),adj(_Adj), prep(Prep),
    det(_Det2),n(N2)]), gn([n(N1),prep(Prep), n(N2)]))
(2) transform_to(gn([det(_Det1),n(N1),adj(_Adj),prep(Prep),
    det(_Det2),n(N2)]), gn([n(N1),prep(Prep),n(N2)])).
(3) transform_to(gn([det(_Det),n(N1),prep(Prep),n(N2),n(N3)]),
    gn([n(N1),prep(Prep), n(N2), n(N3)])).
(4) transform_to(gn([det(_),n(N1),prep(Prep),det(_Det),n(N2)]),
    gn([n(N1), prep(Prep), n(N2)])).
(5) transform_to(gn([det(_Det),n(N1),prep(Prep),det(_Det2),n(N2)]),
    gn([n(N1), prep(Prep), n(N2)])).
(6) ...
```

FIGURE 7.5 – Règles de transformation des groupes nominaux

La première règle transforme un groupe nominal représentant une séquence $\langle déterminant, nom, adjectif, préposition, déterminant, nom\rangle$ en une séquence $\langle nom,$

préposition,nom⟩

Règle de décomposition

```
transform_to(gn([det(_Det),n(N),adv(_Adv),adj('déçue')]),
[gn([n(N)]), gadj([adj('déçue')])]).
```

Cette règle de décomposition ci-dessus transforme un groupe nominal composé d'un déterminant, d'un nom, d'un adverbe et d'un adjectif *déçue* (mot-clé) en deux structures plus petites : un groupe nominal composé d'un nom et un groupe adjectival avec l'adjectif *déçue*.

7.2.5 Transformation des chunks en structures prédicat-argument

Dans la phase précédente, nous avons obtenu les structures des phrases candidates, composées d'un prédicat (sème ou ses dérivés) et les arguments potentiels de ce dernier. Afin de transformer ces structures en prédicat-argument, une ontologie spécifique à ce corpus sera utilisée. Il s'agit en effet de deux grandes catégories d'entités : celles appartenant à l'univers de la clientèle d'un côté et, de l'autre, celles appartenant à l'entreprise à qui s'adressent à ces clients (cf. fiugure 7.6).

Domaine client	me, je, ma, moi, mon, mes, nous, notre, enfant, consommatrice, consommateur, famille, ...
Domaine entreprise	'La_Marque', vous, votre, vos, bûche, 'Nom_du_Produit', chocolat, 'société', truffe, produit, menu, marque, article, pizza, entreprise, ...

FIGURE 7.6 – Deux grandes catégories d'entités

Du côté entreprise, 'La_Marque' renvoie au nom de la société et à ses différentes appellations. De même, 'Nom_du_Produit' comprend les noms propres des produits de cette entreprise. Nous avons procédé ainsi pour ne pas révéler le nom de la marque.

L'étiquetage des arguments repose sur ces deux classements ; par exemple, `argument1` appartient à l'univers de la clientèle et `argument2` à celui de l'entreprise. Notre hypothèse stipule que dans le cas le plus courant les groupes nominaux qui se trouvent autour du prédicat représentent les arguments de ce dernier. Ainsi, les verbes et les prépositions qui ne font pas partie du prédicat seront laissés comme tels pour avoir les structures les plus grammaticales possibles. Le résultat final de la transformation est un ensemble de structures paraphrastiques formalisées sous forme de structures prédicat-argument (cf. annexe B.22).

```
arg2(client:pron(je)),
arg1(entreprise:pron(vous)),
faire,part,de,
arg2(client:det(ma)),
pred('déception')
```

FIGURE 7.7 – Exemple d'une structure de sortie finale

Nous allons maintenant essayer de présenter le processus de traitement du début jusqu'à à fin pour bien montrer les données d'entrée/sortie à chaque étape.

Chapitre 8

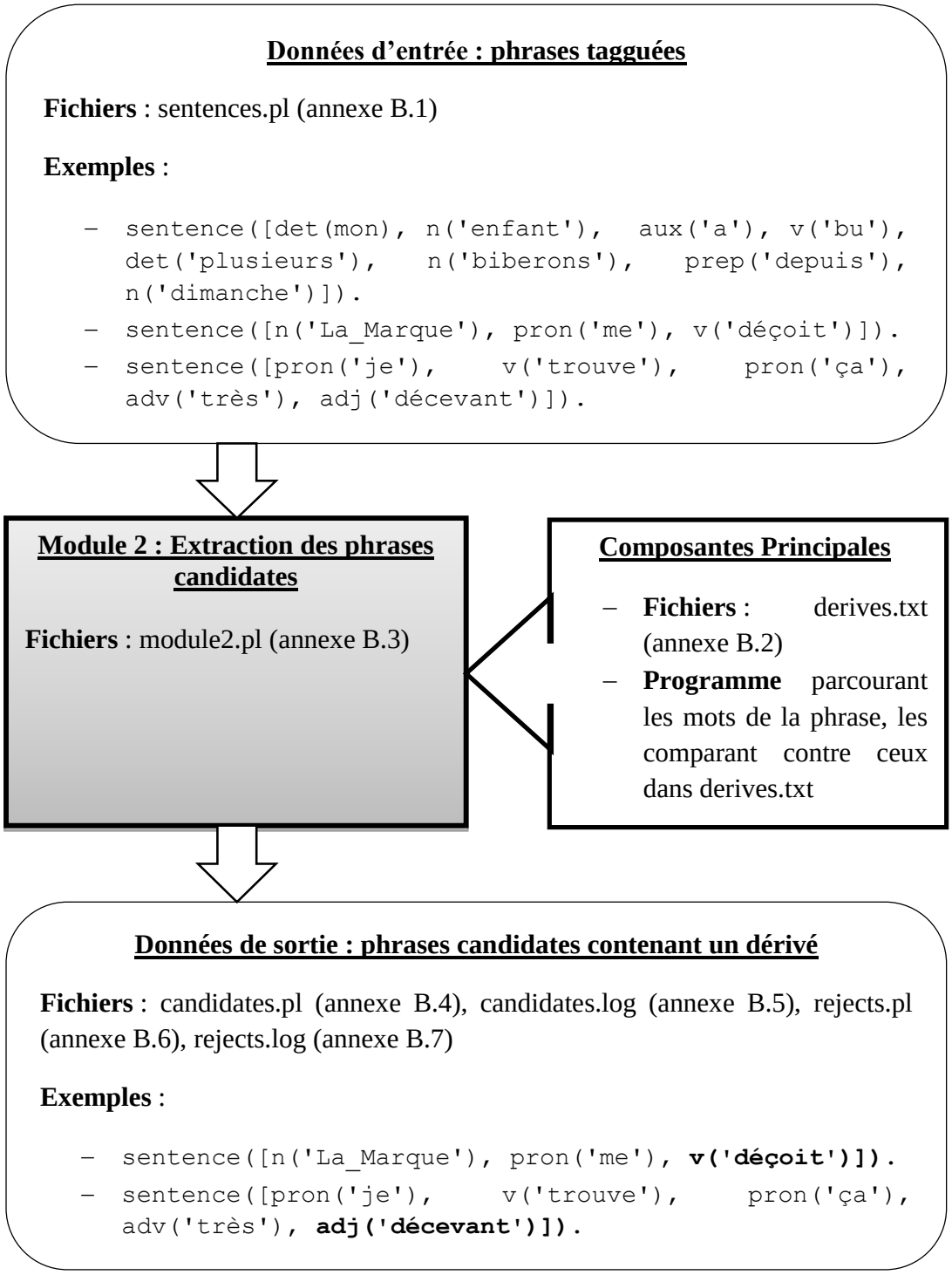
Traitement point par point

L'objectif de cette simulation est de montrer le processus d'extraction des structures paraphrastiques étape par étape après avoir décrit chaque module de traitement. Faute d'espace et de visibilité et comme la tradition le veut, les données relatives à l'implémentation, s'agissant principalement des codes du programme et les ressources linguistiques utilisées, seront en annexes. Nous fournissons quelques phrases comme exemples ainsi que les différentes composantes du système. Chaque élément des schémas dans les pages qui suivent est accompagné d'un renvoi à une forme donnée dans les annexes.

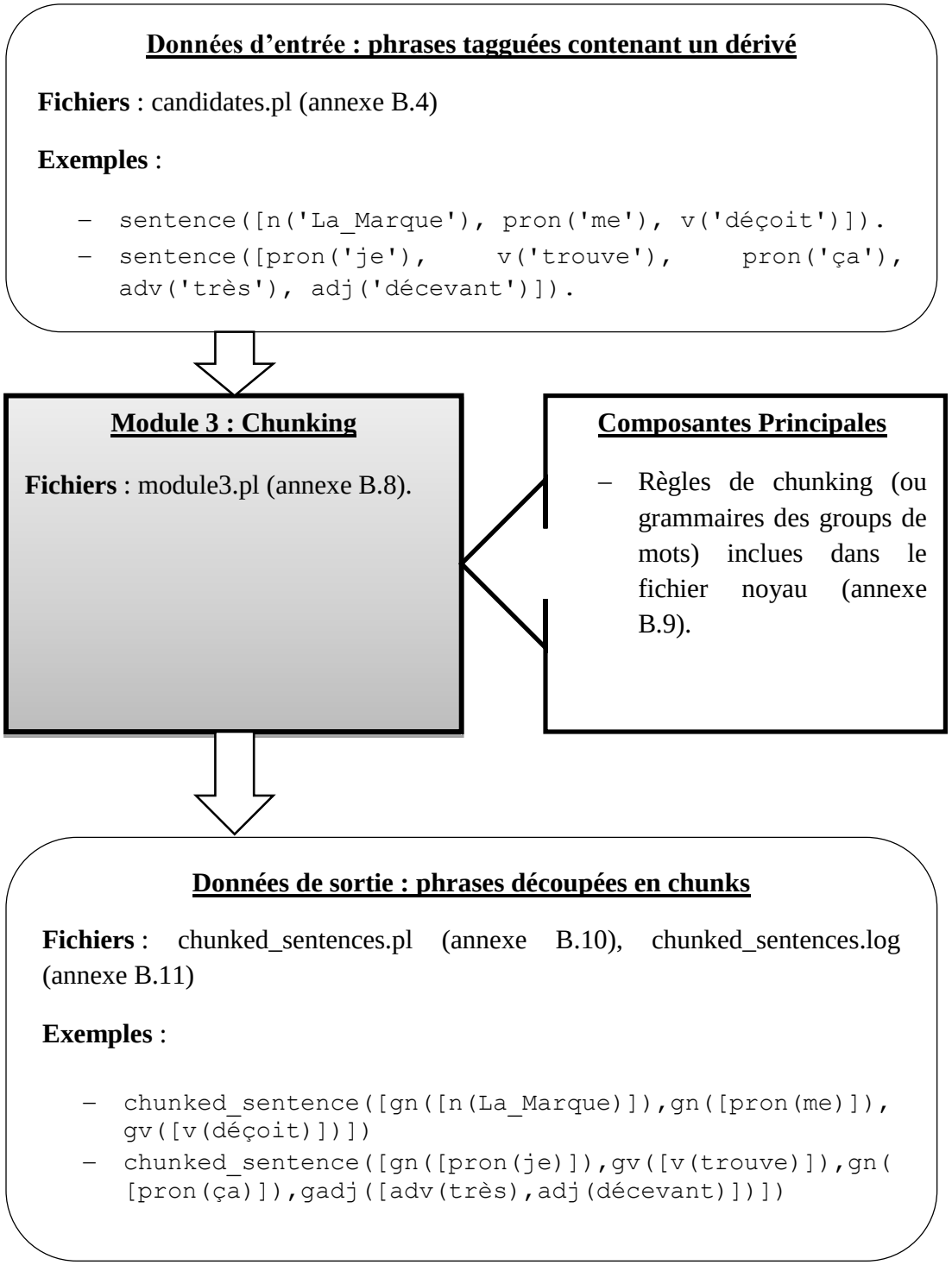
Les figures ci-dessous se composent de trois parties. Elles se lisent de haut en bas, en commençant toujours par les données d'entrée en haut de page. La deuxième partie en milieu de page présente le noyau du traitement libellé Module 1, Module 2, ..., Module 5 ainsi que ses composantes principales. La dernière partie en bas de chaque figure constitue les données de sortie après traitement. Nota bene, si un renvoi est fait aux fichiers en extension « `fichier.pl` » ou « `fichier.log` » comme dans `candidates.pl` (annexe B.4), `candidates.log` (annexe B.5) , cela signifie que les deux fichiers contiennent des données plus ou moins similaires mais

que les fichiers `.pl` sont destinés à la machine alors que ceux en `.log`, aux humains. Pour simplement observer les données, il est conseillé de consulter les fichiers `.log`.

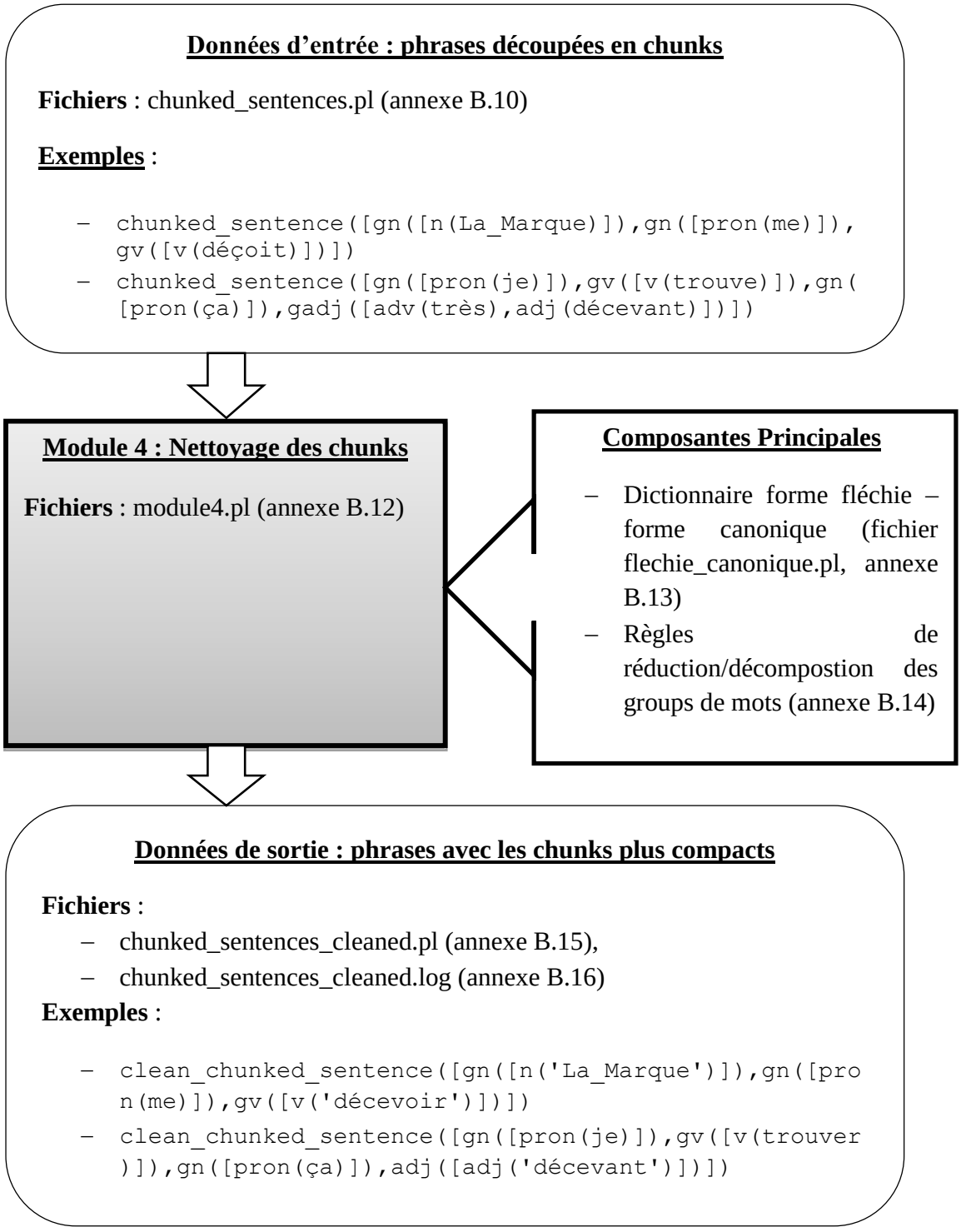
8.1 Module 2 : Extraction des phrases candidates



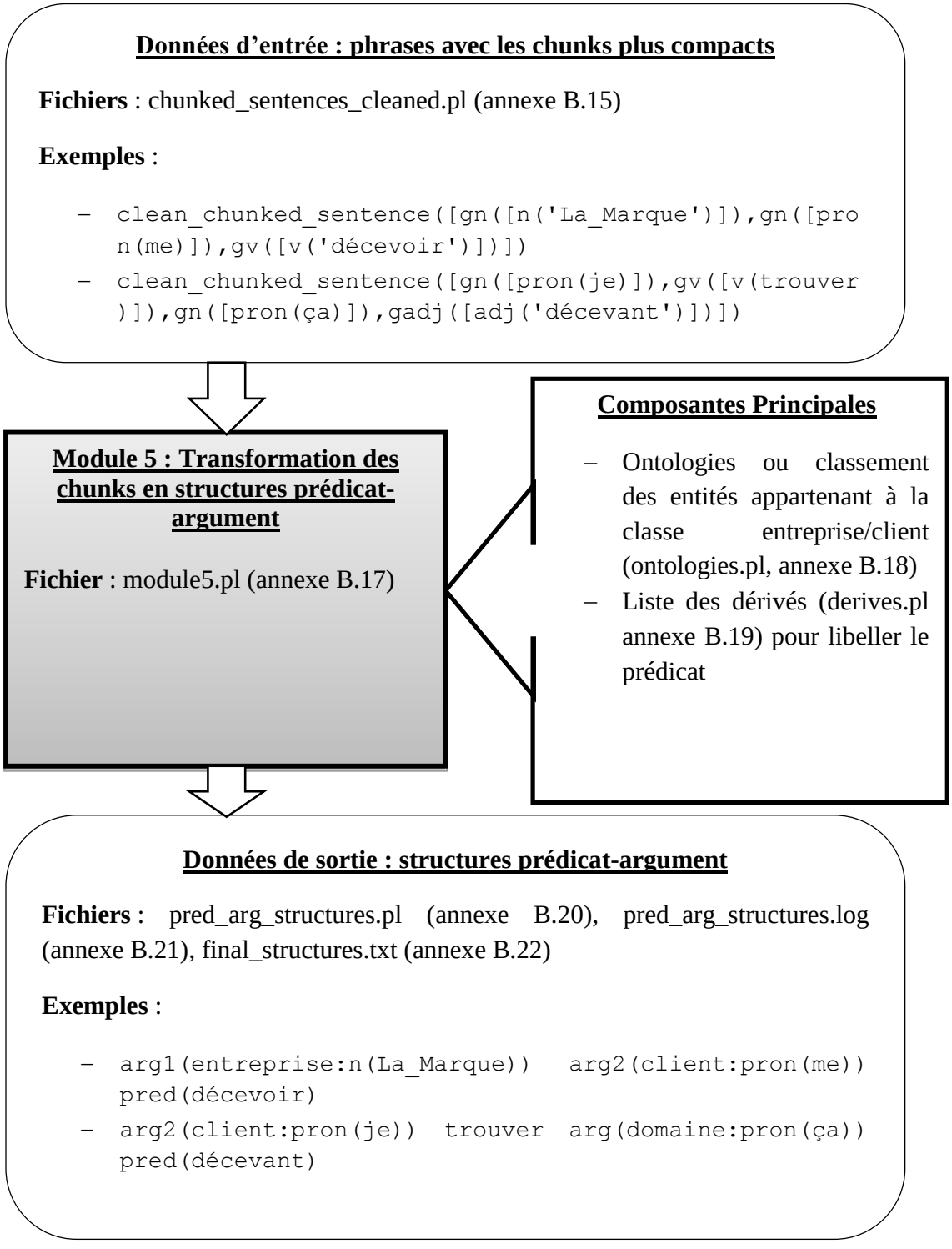
8.2 Module 3 : Chunking



8.3 Module 4 : Nettoyage des chunks



8.4 Module 5 : Transformation des chunks en structures prédicat-argument



8.5 Résultats et discussions

Nous avons montré notre processus d'extraction des structures paraphrastiques étape par étape. Cette implémentation peut être considérée comme étant une application dérivée de nos analyses et formalisations. Le système est fondé sur des règles linguistiques descriptives, simples et sans complexité.

Parmi les vingt-neuf structures paraphrastiques finales (`final_structures.txt`, annexe B.22), quatorze ont le status « `OK` », ce qui signifie qu'elles ont été transformées en structures prédicat-argument avec succès ; les autres quinze phrases ont le statut « `to be reviewed, undefined domaine` », ce qui veut simplement dire qu'il y a un souci. En effet, la plupart des problèmes que présentent les structures problématiques sont faciles à décrire et donc à résoudre. Notre intention de les laisser comme telles a pour but de montrer les éléments qui seraient nécessaires à un traitement spécifique en cas d'implémentation des applications réelles.

En effet, quand le système marque une phrase comme « `to be reviewed, undefined domaine` », il échoue à étiqueter au moins un élément dans la phrase ; cela est très probablement dû à deux raisons : problème de vocabulaire non connu dans l'ontologie ou alors problème syntaxique.

Pour ce qui est de la syntaxe, le problème vient du fait que le système n'a pas fait une analyse syntaxique profonde comme le font généralement les analyseurs syntaxiques. Considérons ainsi la structure ci-dessous dont la phrase initiale est « je suis déçue pour la première fois depuis 25 ans » :

- `arg2(client: pron(je)) être pred(déçu) pour arg(domaine: n(fois)) depuis arg(domaine: n(ans))`

On constate que si le système avait été composé d'un module d'analyse syntaxique de type constituants immédiats, la séquence « pour la première fois depuis

25 ans » aurait été analysée comme un complément circonstanciel et donc supprimée. C'est aussi le cas pour la structure 27, ayant comme phrase originale « j'ai été déçu ce jour » :

– `arg2(client:pron(je)) être pred(déçu) arg(domaine:n(jour))`

On peut ainsi tirer une première conclusion selon laquelle le chunking est efficace sur des phrases simples, plutôt courtes et de style direct. Lorsque des compléments circonstanciels sont présents, une analyse syntaxique plus fine serait nécessaire.

Une autre catégorie de problème est liée aux mots non reconnus dans l'ontologie, la structure 4 et 28 ci-dessous en sont des exemples :

– `arg2(client:pron(je)) trouver arg(domaine:pron(ça))`
`pred(décevant)`
– `arg2(client:pron(je)) arg(domaine:pron(en)) être pred(déçu)`

Le système ne trouve pas *ça* et *en* dans le classement des entités, ce qui est certain. Comme nous l'avons dit dans la partie analyse, ces pronoms sont difficiles à traiter car une bonne analyse du discours est indispensable. La solution la plus sûre à un tel obstacle serait de valider ces structures manuellement. Un autre exemple apparaît sous un simple problème de vocabulaire :

– `arg2(client:pron(je)) être pred(déçu) de arg1(entreprise:`
`n(truffe)) arg(domaine:n(pourcent)).`

Le système a détecté *truffe* comme un des produits de l'entreprise et un autre nom *pourcent* est non reconnu. En effet, dans la phrase originale « je suis déçu des truffes noir 70 % », la séquence « truffes noir 70 % » dans son entier devrait faire partie de l'univers de l'entreprise car c'est un nom propre plutôt que deux groupes nominaux distincts. Il suffit ainsi d'ajouter ce nom propre « truffes noir 70 % » à l'univers de l'entreprise.

D'autre part, des groupes nominaux complexes peuvent être problématiques, tel que « l'éthique de la société » dans la phrase suivante.

– `arg2(client:pron(je)) être donc pred(déçu) quant à arg(domaine:n(éthique)) de arg1(entreprise:n(société))`

Le mot *éthique* n'appartient ni au domaine du client, ni à celui de l'entreprise. Une solution au problème posé par cette structure sera par exemple une simple règle qui impose la suppression des noms hors de l'ontologie dès la phase chunking. Cette démarche entraîne la suppression de *éthique* et par conséquent, la suppression de la préposition *de* qui se trouve entre les deux noms. La structure sera ainsi correctement étiquetée dans la phase finale car seul le nom *société* est présent :

– `arg2(client: pron(je)) être donc pred(déçu) quant à arg1(entreprise: n(société))`

Mais, en faisant ainsi, on risque de supprimer des mots, certes, inconnus mais qui peuvent en effet être intéressants, voire qui devraient être ajoutés dans l'ontologie. Une solution plus prudente serait de laisser *éthique* non libellé comme tel et ensuite de demander à l'utilisateur de le rejeter ou de le valider/ajouter à l'ontologie.

Notre système d'extraction des structures paraphrastiques, si compact et simple qu'il soit, montre déjà qu'il peut être très utile pour des applications de classification de sens comme le Classificatim (Cardey et al., 2006), conçu comme un système basé sur des règles visant la qualité. Ce classificateur de sens comporte un grand nombre de règles paraphrastiques dont la fonction est de détecter les phrases constituant le même sème. Notre système peut y avoir le rôle d'un module de traitement qui extrait les règles paraphrastiques d'une façon automatique ou semi-automatique (dans ce dernier cas, les données de sortie seront validées par l'humain et on écrira les règles manuellement).

Conclusion

Plusieurs formalisations des règles paraphrastiques ont été proposées. Ces grammaires de paraphrase sont, dans la plupart des cas, lexicalisées ; c'est-à-dire que, pour un mot sémantiquement plein donné, on essaie de lui attribuer une ou plusieurs règles paraphrastiques. Certaines constituent un excellent outil de description théorique sur des relations paraphrastiques, tel est le cas du système de paraphrasage de Aleksandr Žolkovskij et Igor Mel'čuk ainsi que la grammaire guidée par prédicat de Jörg Schuster.

Néanmoins, ces formalisations présentent des inconvénients à l'égard du traitement automatique : le temps de développement trop long ou l'implémentabilité mise en cause. Il serait alors intéressant de trouver de nouvelles méthodes de traitement automatique de la paraphrase, qui permettait en quelque sorte un développement plus rapide des ressources lexicales ou phrastiques. En même temps, elles doivent garantir l'informatisation. Ainsi, au lieu de créer une ressource lexicale complète destinée au traitement automatique de la paraphrase, tâche impossible jusqu'à nos jours, nous avons proposé une formalisation linguistique qui a ces deux caractéristiques.

Pour ce faire, nous avons commencé par entamer une analyse linguistique d'un corpus du domaine agroalimentaire pour dégager les mécanismes de paraphrasage les plus courants ainsi que les propriétés fondamentales de structures paraphras-

tiques. Ainsi, pour paraphraser, une substitution lexicale initiale se met en route et déclenche d'autres modifications syntaxiques, lexicales et morphologiques. Substituer un mot par un autre revient à dire qu'il y a remplacement d'un sens par un autre, à priori similaire. Dans cette perspective, pour un couple de paraphrases, il est possible d'identifier au moins deux unités lexicales qui, d'après nous, entretiennent la relation « sens source – sens dérivé ». Les autres sens majeurs dans la phrase sont alors considérées régis par ces deux sens.

Les résultats de l'analyse et la synthèse de ceux-ci ont amené à une généralisation des propriétés de la paraphrase. Cette généralisation nous a permis d'abord de proposer une première formalisation des relations paraphrastiques. Cette formalisation a un but théorique et pédagogique et vient compléter la deuxième formalisation, celle des structures paraphrastiques sous-forme de prédicat-argument. On peut aussi dire que la première formalisation sert à décrire l'action, à savoir les mécanismes de paraphrasage (et les différentes relations paraphrastiques) alors que la deuxième formalisation capte le résultat. Les prédicats correspondent en effet aux « sens sources – sens dérivés », les arguments, eux, à « d'autres sens majeurs régis par les prédicats », comme décrits plus haut. La formalisation des structures paraphrastiques prédicat-argument s'avère adaptée au traitement automatique de la paraphrase car elle est simple, facile à implémenter et assez générale pour être adaptée dans d'autres domaines.

Finalement, nous avons mis en pratique nos formalisations en implémentant un système d'extraction des structures paraphrastiques. Nous y avons montré comment la formalisation des structures paraphrastiques sous forme de prédicat-argument peut contribuer à découvrir d'autres structures paraphrastiques.

La voix de recherche sur la paraphrase et son traitement automatique restera ouverte. Nul ne saurait combien de règles paraphrastiques sont nécessaires pour rendre un système informatique de traitement de la paraphrase quelconque assez

complet. De plus, la question sur le type de règles paraphrastiques le plus approprié au traitement automatique reste à adresser. Nous en avons néanmoins proposé un, celui qui n'est pas simplement une règle paraphrastique mais qui permet d'en découvrir. Nous espérons que cette thèse apportera aux étudiants et à nos collègues chercheurs une nouvelle perspective sur la paraphrase et son traitement automatique.

Références Bibliographiques

Références

Agnäs, M.-S., Alshawi, H., Bretan, I., Carter, D., Ceder, K., Collins, M., ... Svensson, T. (1994). *Spoken language translator, first-year research report R94 :03* (Rapport technique). Kista, Sweden : SICS.

Alshawi, H. (1992). *The Core Language Engine*. ACL-MIT Press series in Natural Language Processing.

Apresjan, J. D., Boguslavskij, I. M., Iomdin, L. L., Lazurskij, A. V., Sannikov, V. Z., & Tsinman, L. L. (1992). ETAP-2 : The Linguistics of a Machine Translation System. *Meta : journal des traducteurs / Meta : Translators*, *37*(1), 97–112.

Birocheau, G. (2000). Morphological Tagging to Resolve Morphological Ambiguities. In *Proceedings of the second international conference on language resources and evaluation (lrec2000).*

Blanché, R. (1984). *La Logique et son histoire : d'Arsitote à Russel.* Paris : Armand Colin.

Cardey, S. (1987). *Traitement algorithmique de la grammaire normative du français pour une utilisation automatique et didatique, Thèse de Doctorat d'Etat.* Université de Franche-Comté, France.

Cardey, S., & Greenfield, P. (2005). Systemic Linguistics with Applications. In *Proceedings of the 9th international symposium on social communication* (pp. 649–653). Santiago de Cuba.

Cardey, S., Greenfield, P., Bioud, M., Dziadkiewicz, A., Kuroda, K., Marcelino, I., ... Vienney, S. (2006). The Classificatim Sense-Mining System. *Advances in Natural Language Processing*, *LNAI 4139*, 674–684.

Chambrieul, M. (1989). *Grammaire de Montague : langage, traduction, interprétation.* ADOSA.

Chomsky, N. (1957). *Syntactic Structures*. Paris : Mouton.

Chomsky, N. (1965). *Aspects of the thoery of syntax* (Cambridge éd.). MIT press : Massachusetts Institute of Technology.

Chomsky, N. (1993). A minimalist program for linguistic theory. *The view from Building 20 : Essays in linguistics in honor of Sylvain Bromberger*, 1–52.

Chomsky, N. (1995). *The Minimalist Program. Cambridge*. MA : The MIT Press.

CoGenTex. (1992). *Bilingual Text Synthesis System for Statistics Canada Database Reports : Design of Retail Trad Statistics (RTS) Prototype. Technical Report 8.* (Rapport technique). Montreal : CoGenTex Inc.

Culioli, A. (1976). *Transcription du séminaire de D.E.A. (1975-1976) : Recherche en linguistique ; théorie des opérations énonciatives*. Paris : Université de Paris VII.

Culioli, A. (1990). *Pour une linguistique de l'énoniation*. Paris : Ophrys.

El Harouchy, Z. (1997). Dictionnaire Automatique et Grammaire pour le traitement automatique des ambiguïtés morphologiques des mots simples en français. In *Actes du colloque international fractal* (pp. 141–151). Besançon.

Fellbaum, C. (1998). WordNet : An Electronic Lexical Database. Cambridge, MA : MIT Press.

Fiser, D., & Sagot, B. (2008). Combining multiple resources to build reliable wordnets. In *Tsd 2008.* Brno.

Fluhr, C., & the contributors SAIMSI. (2013). Suivi Adaptatif Interlingue et MultiSources des Informations. In *Workshop wisg2013.* Troyes.

Fuchs, C. (1980). *Paraphrase et théories du langage : contribution à une histoire des théories linguistiques contemporaines et à la construction d'une théorie énonciative de la paraphase*. Thèse de doctorat non publiée, Paris VII, Editeur : Paris, Centre de documentation sciences humaines.

Fuchs, C. (1982). *La Paraphrase*. Paris : Presses Universitaires de France.

Fuchs, C. (1984). *Paraphrase et énonciation.* Paris : Ophrys.

Galmiche, M. (1975). *Sémantique générative.* Collection. Langue et Langage. Paris : Larousse.

Gardent, C., Guillaume, B., Falk, I., & Perrier, G. (2005). Le lexique-grammaire de M. Gross et le traitement automatique des langues. *Communication à la jounée ATALA : Interface lexique-grammaire et lexiques syntaxiques et sémantiques.*

Gentilhomme, Y. (1985). *Essai d'approche microsystémique, Théorie et pratique, Application dans le domaine des sciences du langage.* Peter Lang.

Gross, M. (1968). *Grammaire transformationnelle du français : syntaxe du verbe.* Collection. Langue et Langage. Paris : Larousse.

Gross, M. (1975). *Méthodes en syntaxe : régime des constructions complétives* (N° 1365). Collection. Actualités scientifiques et industrielles. Paris : Hermann.

Gross, M. (1977). *Grammaire transformationnelle du français : Syntaxe du nom.* Paris : Larousse.

Gross, M. (1979). On the failure of Generative Grammar. *Language, Journal of the Linguistic Society of America Baltimore*, *5*, 859–885.

Harris, Z. (1972). The two systems of grammar : Report and Paraphrase. In S. Plötz (Ed.), *Transformationelle analyse* (pp. 155–241). Frankfurt : Athenäum.

Harris, Z. S. (1960). *Structural Linguistics* (Phoenix Bo éd.). Chicago : The University of Chicago press.

Harris, Z. S. (1970). *Mathematical structures of language.* New York : Interscience Publishers John Wiley & Sons.

Harris, Z. S. (1976). Notes du cours de syntaxe. In M. Gross (Trad.), (Seuil éd.). Collection. Travaux linguistiques. Paris.

Harris, Z. S. (1981). The elementary transformations. *Papers on Syntax*, 211–235.

Hathout, N., Namer, F., & Dal, G. (2002). *An Experimental Constructional Database : The MorTAL Project* (P. Boucher, Ed.). Cascadilla, Somerville, Mass.

Iordanskaja, L. (1992). Communicative Structure and its Use during Text Generation. *IFID*, *17*(2), 15–27.

Kampeera, W., & Cardey, S. (2012). Building a Lexically and Semantically-Rich Resource for Paraphrase Processing,. In *Proceedings of the 8th international conference on natural language processing* (pp. 138–143). JapTAL 2012, Kanazawa : Springer-Verlag Berlin Heidelberg.

Laporte, E. (2010). Le lexique-grammaire est-il exploitable pour le traitement des langues ? *Les Tables. La grammaire du français par le menu. Mélanges en hommage à Christian Leclère*, 207–218.

Martin, R. (1976). *Inférence, autonomie et paraphrase : éléments pour une théorie sémantique*. Strasbourg : Klincksieck.

Mel'čuk, I. (1999). *Dictionnaire explicatif et combinatoire du français comtamporain. Recherhces lexico-sémantiques IV*. Montreal : Les Presses de l'Université de Montréal.

Mel'čuk, I., Arbatchewsky-Jumarie, N., Iordanskaja, L., & Mantha, S. (1992). *Dictionnaire explicatif et combinatoire du français comtamporain. Recherhces lexico-sémantiques III*. Montreal : Les Presses de l'Université de Montréal.

Mihajlovska, K. (2011). *Anotación en corpus de las funciones léxicas oper, real, caus y sus combinaciones*. Mémoire de recherche. Universitat Autònoma de Barcelona.

Milićević, J. (2007). *La paraphrase, modélisation de la paraphrase langagière* (Peter Lang éd.). Collection. Sciences pour la communication.

Montague, R. (1970). *Universal Grammar* (Theoria éd.).

Morgadinho, H., & Venet, A.-L. (2003). Construction de dictionnaires microsys-

tèmiques. In *"microsystèmes et tal", special number of the bulag.* Besançon : PUFC.

Nasr, A. (1996). *Un modèle de reformulation automatique fondé sur la théorie Sens-Texte : applications aux langages contrôlés.* Thèse de doctorat non publiée, Université Paris 7.

Nugues, M. P. (2006). *An introduction to language processing with Perl and Prolog : an outline of theories, implementation, and application with special consideration of English, French, and German.* Collection. Cognitive technologies. Berlin : Springer.

Partee, B. H., Meulen, A. T., & Wall, R. E. (1990). *Mathematical Methods in Linguistics* (Student éd.). Dordrecht/Holland : Kluwer Academic Publishers.

Polguère, A., & Mel'čuk, I. (2006). Dérivations sémantiques et collocations dans le DiCo/LAF. *Langue française*, *2*(150), 66–83.

Sajous, F., Navarro, E., Gaume, B., Prévot, L., & Chudy, Y. (2010). Semi-automatic Endogenous Enrichment of Collaboratively Constructed Lexical Resources : Piggybacking onto Wiktionary. *Advances in Natural Language Processing, Lecture Notes in Computer Science*, *6233*, 332–344.

Schuster, J. (2009). *Towards Predicate Driven Grammar* (F. Guenthner, Ed.). LINCOM GmbH.

Shieber, S. M. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 334–343.

Smaby, R. M. (1971). *Paraphrase grammars.* Formal linguistics, series 2. New York : Humanities Press.

Tesnière, L. (1959). *Éléments de syntaxe structurale.* Paris : C. Klincksieck.

Žolkovskij, A., & Mel'čuk, I. (1965). ovozmožnom metode i instrumentax semantičeskogo sinteza (One possible method and instruments for semantic synthesis of texts). In *Naučnotexničeskaja informacija* (pp. 23–28).

Annexes

Annexe A

Données linguistiques

A.1 Dictionnaire électronique Dicouèbe

DiCouèbe LE DICO EN LIGNE

48 résultats

Chercher | Chercher (nouvelle fenêtre) | Effacer tout | Effacer tous les champs

Mode expert

Ø↓

↓ **LEXIE** Ø↓

- **nom vocable** [lexie:vocable] Ø→ amour
- **no. acception** [lexie:num] Ø→
- **carac. grammaticales** [lexie:cgs] Ø→
- **étiquette sém.** [lexie:formuleEtiquette] Ø→

↓ **LIENS DE FONCTIONS LEXICALES** Ø↓ + FL(2)...

- **fonction lexicale** [FL:formuleFL] Ø→
- **glose** [FL:glose] Ø→
- **valeur fusionnée?** [FL:estFusionnee] Ø→
- **marque d'usage** [FL:marqueDUsage] Ø→
- **valeur** [FL:lexie] Ø→
- **régime valeur** [FL:regime] Ø→
- **contrainte** [FL:contrainte] Ø→
- **exemple** [FL:exemple] Ø→

↓ **LOCUTIONS CONSTRUITES AVEC LE MOT-CLÉ** Ø↓ + phraseme(2)...

- **locution** [phraseme:phraseme] Ø→

↓ **EXEMPLES D'EMPLOI DU MOT-CLÉ** Ø↓ + exemple(2)...

- **phrase d'exemple** [exemple:exemple] Ø→

Copyright © 2005 OLST—Université de Montréal. Tous droits réservés.
http://idefix.ling.umontreal.ca/dicouebe/LICENCE.txt

48 résultats

Exporter les résultats

lexie vocable	lexie num	lexie cgs	lexie formuleEtiquette	FL formuleFL	FL glose	FL estFusionnee	FL marqueDUsage	FL lexie	FL regime	FL contrainte	FL exemple	phraseme phraseme	exemple exemple
AMOUR-PROPRE		nom, masc	trait de caractère	QSyn		0		orgueil#1					Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE		nom, masc	trait de caractère	QSyn		0		fierté					Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE		nom, masc	trait de caractère	QSyn		0		égo					Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE		nom, masc	trait de caractère	QAnti		0		abnégation					Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE		nom, masc	trait de caractère	QAnti		0		humilité					Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE		nom, masc	trait de caractère	A1Non	[X] qui n'a pas d'A.	0		dénué	[de (ART) ~]				Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
													Je croyais de moins en moins à

AMOUR-PROPRE		nom, masc	trait de caractère	Magn	Sensible	0		beaucoup	[de ~]				l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE		nom, masc	trait de caractère	AntiBon.Magn	Trop sensible	0		démesuré		postpos			Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE		nom, masc	trait de caractère	Oper1	[X] avoir de l'A.	0		avoir	[ART <de l'> ~]				Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE		nom, masc	trait de caractère	Real2	[Y] affecter positivement ~	0		titiller					Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE		nom, masc	trait de caractère	Real2	[Y] affecter positivement ~	0		caresser					Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE		nom, masc	trait de caractère	Real2	[Y] affecter positivement ~	0		chatouiller					Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE		nom, masc	trait de caractère	Real2	[Y] affecter positivement ~	0		flatter					Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
													Je croyais de

AMOUR-PROPRE	nom, masc	trait de caractère	Real2	[Y] affecter positivement 0 ~		satisfaire	[ART ~]			moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE	nom, masc	trait de caractère	AntiReal2	[Y] affecter négativement 0 ~		titiller				Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE	nom, masc	trait de caractère	AntiReal2	[Y] affecter négativement 0 ~		blesser	[ART ~]			Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE	nom, masc	trait de caractère	AntiReal2	[Y] affecter négativement 0 ~		heurter	[ART ~]			Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE	nom, masc	trait de caractère	AntiLabreal21	[Y] affecter négativement 0 ~		atteindre	[N=X dans A-poss ~]			Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE	nom, masc	trait de caractère	AntiLabreal21	[Y] affecter négativement 0 ~		blesser	[N=X dans A-poss ~]			Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.
AMOUR-PROPRE	nom, masc	trait de caractère	AntiLabreal21	[Y] affecter négativement 0 ~		toucher	[N=X dans A-poss ~]			Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.

Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.	[N=X]	blesser	[Y] affecter négativement ~	1	AntiLabreal21	trait de caractère	nom, masc	AMOUR-PROPRE	
Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.	[N=X]	humilier	[Y] affecter négativement ~	1	AntiLabreal21	trait de caractère	nom, masc	AMOUR-PROPRE	
Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.	[N=X]	mortifier	[Y] affecter négativement ~	1	AntiLabreal21	trait de caractère	nom, masc	AMOUR-PROPRE	
Je croyais de moins en moins à l'amour, mais de plus en plus à l'amour-propre.	[d'~]	blessure	Ce qui résulte du fait que Y affecte négativement ~	0	SresAntiLabreal21	trait de caractère	nom, masc	AMOUR-PROPRE	
Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.		orgueil#1		0	QSyn	trait de caractère	nom, masc	AMOUR-PROPRE	
Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration		fierté		0	QSyn	trait de caractère	nom, masc	AMOUR-PROPRE	

	AMOUR-PROPRE									dans la classe.
		nom, masc	trait de caractère	QSyn	0	égo				Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.
	AMOUR-PROPRE									
		nom, masc	trait de caractère	QAnti	0	abnégation				Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.
	AMOUR-PROPRE									
		nom, masc	trait de caractère	QAnti	0	humilité				Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.
	AMOUR-PROPRE									
		nom, masc	trait de caractère	A1Non	[X] qui n'a pas d'A.	dénué	[de (ART) ~]			Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.

AMOUR-PROPRE									culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.
	nom, masc								
		trait de caractère	Magn						
				Sensible	0				
AMOUR-PROPRE									
	nom, masc								
		trait de caractère	AntiBon.Magn						
				Trop sensible	0				
AMOUR-PROPRE									
	nom, masc								
		trait de caractère	Oper1						
				[X] avoir de l'A.	0				
AMOUR-PROPRE									
	nom, masc								
		trait de caractère	Real2						
				[Y] affecter positivement ~	0				
									[Y] affecter

									culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.

AMOUR-PROPRE									
	nom, masc	trait de caractère	Real2	~ positivement	0				son amour-propre et éprouve des difficultés d'intégration dans la classe.
AMOUR-PROPRE									
	nom, masc	trait de caractère	Real2	[Y] affecter positivement ~	0		chatouiller		Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.
AMOUR-PROPRE									
	nom, masc	trait de caractère	Real2	[Y] affecter positivement ~	0		flatter		Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.
AMOUR-PROPRE									
	nom, masc	trait de caractère	Real2	[Y] affecter positivement ~	0		satisfaire	[ART ~]	Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.
AMOUR-PROPRE									
	nom, masc	trait de caractère	AntiReal2	[Y] affecter négativement ~	0		titiller		Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés

										d'intégration dans la classe.
										Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.
		AMOUR-PROPRE								
		nom, masc								
		trait de caractère	AntiReal2							
		[Y] affecter négativement 0 ~								
					blesser	[ART ~]				
										Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.
		AMOUR-PROPRE								
		nom, masc								
		trait de caractère	AntiReal2							
		[Y] affecter négativement 0 ~								
					heurter	[ART ~]				
										Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.
		AMOUR-PROPRE								
		nom, masc								
		trait de caractère	AntiLabreal21							
		[Y] affecter négativement 0 ~								
					atteindre	[N=X dans A-poss ~]				
										Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.
		AMOUR-PROPRE								
		nom, masc								
		trait de caractère	AntiLabreal21							
		[Y] affecter négativement 0 ~								
					blesser	[N=X dans A-poss ~]				
										Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.
										Un élève

dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.	AMOUR-PROPRE								
Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.	AMOUR-PROPRE	nom, masc	trait de caractère	AntiLabreal21	[Y] affecter négativement 0 ~	toucher	[N=X dans A-poss ~]		
Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.	AMOUR-PROPRE	nom, masc	trait de caractère	AntiLabreal21	[Y] affecter négativement 1 ~	blesser	[N=X]		
Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.	AMOUR-PROPRE	nom, masc	trait de caractère	AntiLabreal21	[Y] affecter négativement 1 ~	humilier	[N=X]		
Un élève dont la culture est dévalorisée se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.	AMOUR-PROPRE	nom, masc	trait de caractère	AntiLabreal21	[Y] affecter négativement 1 ~	mortifier	[N=X]		Ce qui
Un élève dont la culture est dévalorisée									

AMOUR-PROPRE									
	nom, masc	trait de caractère	SresAntiLabreal21	résulte du fait que Y affecte négativement ~	0		blessure	[d'~]				se sent touché dans son amour-propre et éprouve des difficultés d'intégration dans la classe.

afficher requête SQL

A.2 Exemples de règles de PDG

79.
```
arity     2
syntax    np-sy ==> possdet -v-yg
meaning   annoy(love[s=ger](I,Mary),John)
text      My loving Mary annoys John
owners    love, like, see
```

80.
```
arity     1
syntax    s ==> vp
meaning   snore(John)
text      John snores
owners    snore, sleep
```

81.
```
arity     2
syntax    s ==> vp np-ca
meaning   like(John,Mary)
text      John likes Mary
owners    like, kiss
```

82.
```
arity     2
syntax    s ==> -vp s
meaning   believe(John,love(Jack,Mary))
text      John believes Jack loves Mary
owners    believe, think
```

83.
```
arity     2
syntax    s ==> -vp that s
meaning   believe(John,love(Jack,Mary))
text      John believes that Jack loves Mary
owners    believe, think
```

84.
```
arity     1
syntax    s ==> -vp adv--gravely np-ca
meaning   magn[l=ba0c,s=adv0](doubt,he(love[s=n0](she,he)))
text      He doubts gravely her love for him
owners    doubt
```

85.
```
arity     1
syntax    s ==> -vp adv--gravely s
meaning   magn[l=ba0c,s=adv0](doubt,he(love(she,he)))
text      He doubts gravely she loves him
owners    doubt
```

86.
```
arity     1
syntax    s ==> -vp adv--gravely that s
meaning   magn[l=ba0c,s=adv0](doubt,he(love(se,he)))
text      He doubts gravely that she loves him
owners    doubt
```

87.
```
arity     1
syntax    s ==> -vp adv--seriously np-ca
meaning   magn[l=ba0c,s=adv0](doubt,he(love[s=n0](she,he)))
text      He doubts seriously her love for him
owners    doubt
```

88.
```
arity     1
syntax    s ==> -vp adv--seriously s
meaning   magn[l=ba0c,s=adv0](doubt,he(love(se,he)))
text      He doubts seriously she loves him
owners    doubt
```

89.
```
arity     1
syntax    s ==> -vp adv--seriously that s
meaning   magn[l=ba0c,s=adv0](doubt,he(love(se,he)))
text      He doubts seriously that she loves him
owners    doubt
```

90.
```
arity     1
syntax    s ==> -vp adv--severely np-ca
meaning   magn[l=ba0c,s=adv0](doubt,he(love[s=n0](she,he)))
text      He doubts severely her love for him
owners    doubt
```

91.
```
arity     1
syntax    s ==> -vp adv--severely s
meaning   magn[l=ba0c,s=adv0](doubt,he(love(se,he)))
text      He doubts severely she loves him
owners    doubt
```

92.
```
arity     1
syntax    s ==> -vp adv--severely that s
meaning   magn[l=ba0c,s=adv0](doubt,he(love(se,he)))
text      He doubts severely that she loves him
owners    doubt
```

93.
```
arity     2
syntax    s ==> -vp prep--in np-ca
meaning   believe(he,love[s=n0](she,he))
text      He believes in her love for him
owners    believe
```

94.
```
arity     3
syntax    s ==> -vp np-ca prep--to np-ca
meaning   give(John,book[u=the],Mary)
text      John gives the book to Mary
owners    give
```

Annexe B

Codes du programme en Prolog

B.1 92 phrases tagguées sentences.pl

```
sentence([pron('je'), v('suis'), adj('déçue'), prep('par'), det('une'),
n('tablette'), prep('de'), n('chocolat'), n('Nom_du_Produit')]).
sentence([pron('cela'), pron('me'), v('déçoit')]).
sentence([pron('celle-ci'),  pron('me'),  aux('a'),  adv('beaucoup'),
v('déçu')]).
sentence([n('La_Marque'), pron('me'), v('déçoit')]).
sentence([pron('je'),     v('trouve'),     pron('ça'),     adv('très'),
adj('décevant')]).
sentence([prep('à'), det('ma'), adj('grande'), n('déception')]).
sentence([pron('je'), v('suis'), adj('déçue'), prep('pour'), det('la'),
adj('première'), n('fois'), prep('depuis'), adj('25'), n('ans')]).
sentence([pron('je'),     v('suis'),     det('une'),     adj('grande'),
n('consommatrice'),     det('de'),     pron('vos'),     n('yaourts'),
adj('fruités'), det('0'), n('%')]).
sentence([pron('je'),   v('suis'),   det('une'),       adj('fidèle'),
n('consommatrice')]).
sentence([det('ce'),   n('courrier'),   prep('pour'),   pron('vous'),
v('faire'), n('part'), prep('de'), det('ma'), n('déception')]).
sentence([adj('quelle'),  n('déception'),  adj('même'),  adv('pas'),
det('une'), n('décoration'), prep('sur'), det('les'), n('bûches')]).
sentence([pron('je'), pron('vous'), v('fais'), n('part'), prep('de'),
det('ma'), n('déception')]).
sentence([pron('je'), pron('vous'), v('fais'), n('part'), prep('de'),
det('ma'),  n('déception'),  prep('vis-à-vis'),  prep('de'),  det('le'),
n('Nom_du_Produit')]).
sentence([det('la'),     n('déception'),     prep('de'),     det('un'),
n('enfant')]).
sentence([adj('quel'),  adv('ne'),  v('est'),  adv('pas'),  det('ma'),
n('déception')]).
sentence([pron('je'),  aux('ai'),  v('été'),  adj('déçu'),  adv('lors'),
prep('de'), det('mon'), adj('dernier'), n('achat')]).
sentence([pron('je'),  aux('ai'),  v('été'),  adj('déçue'),  prep('par'),
det('le'),    n('côté'),    adj('âpre'),    prep('de'),    det('cet'),
n('article')]).
sentence([pron('je'),    pron('me'),    aux('suis'),    v('trouvée'),
adv('très'), adj('déçue'), prep('de'), det('ce'), n('produit')]).
sentence([pron('je'), v('suis'), adv('amèrement'), adj('déçue')]).
sentence([pron('je'),   v('suis'),   det('une'),   n('consommatrice'),
adv('très'),  adj('déçue'),  prep('de'),  det('votre'),  adj('grande'),
n('marque')]).
sentence([pron('je'),    v('suis'),    adv('vraiment'),    adj('déçu'),
prep('de'),      det('vos'),      n('pizzas'),      adj('surgelées'),
n('Nom_du_Produit')]).
sentence([pron('je'), v('suis'), adj('déçue')]).
sentence([pron('je'), v('suis'), adv('très'), adj('déçue')]).
```

```
sentence([pron('je'), aux('ai'), v('été'), adv('très'), adj('déçu'),
det('ce'), n('jour')]).
sentence([pron('je'), pron(en), aux('ai'), v('été'), adj('déçue')]).
sentence([pron('je'), v('suis'), adv('particulièrement'), adj('déçue'),
prep('de'), det('mon'), adj('dernier'), n('achat')]).
sentence([pron('nous'), aux('avons'), v('été'), adj('déçus')]).
sentence([pron('je'), v('bois'), adv('régulièrement'), det('ce'),
n('chocolat'), prep('à'), det('le'), n('bureau'), pron('que'),
pron('je'), v('trouvais'), adj('délicieux')]).
sentence([pron('je'), pron('le'), aux('ai'), v('entamé'), adv('hier'),
conj('ou'), adv('avant-hier')]).
sentence([pron('je'), v('utilise'), det('ce'), n('lait'),
prep('depuis'), det('sa'), n('naissance')]).
sentence([pron('il'), adv('ne'), v('mange'), adv('que'),
pron('cela')]).
sentence([pron('je'), v('suis'), adv('donc'), adv('très'),
adj('déçue'), prep('quant'), prep('à'), det('la'), n('éthique'),
prep('de'), det('votre'), n('société')]).
sentence([adj('fidèle'), n('consommateur'), det('de'), pron('vos'),
n('produits')]).
sentence([pron('je'), adv('ne'), v('mange'), adv('plus'),
prep('depuis'), det('deux'), n('jours')]).
sentence([pron('je'), v('recherche'), adv('donc'), det('cette'),
n('crème'), conj('car'), pron('il'), pron('la'), aux('a'),
v('apprécie'), adv('beaucoup'), prep('depuis'), pron('que'),
pron('il'), v('mange'), prep('à'), det('la'), n('cuillère')]).
sentence([n('consommatrice'), adj('inconditionnelle'), prep('avec'),
det('mes'), adj('petits'), n('enfants'), det('de'), det('votre'),
n('Nom_du_Produit')]).
sentence([pron('nous'), v('utilisons'), det('le'), n('lait'),
n('Nom_du_Produit'), prep('depuis'), det('la'), n('naissance'),
det('de'), det('notre'), n('fils')]).
sentence([pron('je'), v('utilise'), det('ce'), n('lait'),
prep('depuis'), adv('longtemps')]).
sentence([pron('je'), v('ai'), det('la'), n('habitude'), det('de'),
v('consommer'), pron('vos'), n('produits')]).
sentence([det(mon), n('enfant'), aux('a'), v('bu'), det('plusieurs'),
n('biberons'), prep('depuis'), n('dimanche')]).
sentence([pron('on'), aux('a'), v('mangé'), pron('tous'),
adv('pareil'), conj('sauf_que'), pron('eux'), aux('ont'), v('mangé'),
det('le'), n('jambon')]).
sentence([pron('nous'), pron('nous'), adv('ne'), pron('en'),
aix('avons'), adv('pas'), v('mangé')]).
sentence([pron('je'), v('suis'), adj('déçue'), prep('de'),
det('votre'), n('marque'), n('La_Marque')]).
```

```
sentence([pron('je'),        v('consomme'),        n('Nom_du_Produit'),
prep('depuis'), det('15'), n('ans')]).
sentence([pron('je'), v('suis'), adv('très'), adj('déçu'), prep('de'),
det('les'), n('truffes'), adj('noir'), adj('70'), n('%')]).
sentence([pron('je'), v('suis'), adv('très'), adj('déçue'), prep('de'),
det('ce'), n('produit')]).
sentence([pron('je'),  v('utilise'), adv('habituellement'), det('ce'),
n('lait')]).
sentence([det('mes'), n('chiens'), adv('ne'), v('mangent'), adv('que'),
pron('ça')]).
sentence([pron('je'),      v('suis'),      det('une'),      adv('très'),
adj('grande'),     n('consommatrice'),     det('de'),     pron('vos'),
n('barres')]).
sentence([prep('après'),   det('les'),   aux('avoir'),   v('consommés'),
det('à'), adj('fameux'), n('repas'), pron('ils'), aux('ont'), v('eu'),
det('des'), n('rillettes')]).
sentence([det('mes'),     det('trois'),     n('chats'),     v('manger'),
det('nos'), n('restes'), det('de'), n('table')]).
sentence([pron('je'),     v('suis'),     det('une'),     adj('grande'),
n('consommateur'), det('de'), n('Nom_du_Produit')]).
sentence([pron('ce'),  v('est'),  det('une'),  n('lait'),  pron('que'),
pron('je'), v('utilise'), prep('depuis'), det('1'), n('mois')]).
sentence([pron('je'),    v('suis'),    det('une'),    adj('fervente'),
n('consommatrice')]).
sentence([det('les'),    n('Français'),    v('mangent'),    adv('trop'),
det('de'), n('sel')]).
sentence([pron('je'), v('suis'), adj('déçue'), prep('de'), det('leur'),
n('remarque')]).
sentence([pron('je'),  v('suis'),  adj('déçu'),  prep('de'),  det('mes'),
adj('deux'), adj('derniers'), n('achats'), adj('identiques')]).
sentence([pron('je'), v('suis'), adv('très'), adj('déçu'), prep('par'),
det('le'), n('menu')]).
sentence([adv('Très'),     n('amateur'),     prep('de'),     det('les'),
adv('habituellement'),      adj('excellents'),      n('Nom_du_Produit'),
pron('que'),  pron('je'),  v('consomme'),  adv('sans'),  n('modération'),
prep('en'),    n('période'),    det('de'),    n('fin'),    prep('de'),
n('année')]).
sentence([det('mon'),      n('fils'),      v('mange'),      det('des'),
n('biscuits')]).
sentence([n('consommateurs'),       adj('réguliers'),       det('mes'),
n('enfants'), conj('et'), pron('moi')]).
sentence([n('consommatrice'),   det('de'),   det('vos'),   n('cafés'),
adj('solubles'),    pron('dont'),    det('le'),    n('Nom_du_Produit'),
prep('depuis'), det('de'), adj('nombreuses'), n('années')]).
```

```
sentence([pron('ce'), pron('qui'), v('devient'), adj('difficile'),
prep('pour'), n('Tom'), det('de'), det('les'), v('déguster')]).
sentence([pron('je'), v('suis'), n('utilisateur'), prep('de'),
det('le'), n('lait'), adj('infantile'), n('La_Marque'), prep('en'),
n('poudre')]).
sentence([pron('je'), pron('en'), v('mange'), adv('beaucoup')]).
sentence([pron('je'), pron('le'), aux('ai'), v('goûté'), conj('et'),
pron('je'), aux('ai'), adv('tout'), v('recraché')]).
sentence([pron('ce'), v('est'), det('la'), adj('troisième'),
n('boite'), pron('que'), pron('je'), v('utilise')]).
sentence([pron('ce'), n('paquet'), det('de'), n('céréales'),
v('contenait'), det('une'), n('fil'), det('de'), n('nylon'), adj('ci-
joint')]).
sentence([adj('habituée'), pron('je'), v('consomme'),
n('Nom_du_Produit'), prep('depuis'), det('15'), n('ans')]).
sentence([adj('habituée'), det('les'), n('lardons'),
v('ressemblaient'), prep('à'), det('des'), adj('petites'),
n('chiquettes')]).
sentence([adj('habituée'), pron('il'), adv('ne'), v('mange'),
pron('que'), pron('cela')]).
sentence([adv('habituellement'), det('ce'), v('est'), adv('bien'),
adj('meilleur')]).
sentence([pron('je'), v('achète'), adv('habituellement'), det('votre'),
n('produit'), adj('Nom_du_Produit'), adj('adulte'), n('poulet')]).
sentence([pron('je'), v('achète'), adv('régulièrement'), det('du'),
n('Nom_du_Produit'), prep('en'), n('terrine'), conj('car'),
pron('ils'), adv('ne'), v('aiment'), adv('que'), det('la'),
n('terrine')]).
sentence([pron('je'), v('achète'), adv('souvent'), det('ces'),
n('lasagnes')]).
sentence([n('consommatrice'), adj('inconditionnelle'), prep('avec'),
det('mes'), adj('petits'), n('enfants'), det('de'), det('votre'),
n('Nom_du_Produit')]).
sentence([pron('elle'), v('mange'), det('les'), n('Nom_du_Produit'),
prep('de'), n('habitude')]).
sentence([pron('elle'), pron('y'), v('est'), adj('habituée'),
det('ce'), v('est'), adv('pourquoi'), pron('je'), v('insiste'),
prep('auprès'), det('de'), pron('vous')]).
sentence([adv('en_tant_que'), n('cliente'), adj('fidèle')]).
sentence([adv('est-ce_que'), n('Nom_du_Produit'), v('fait'),
adv('comme'), n('La_Marque'), prep('à'), v('savoir'), det('des'),
n('collecteurs'), prep('pour'), v('fidéliser'), det('les'),
n('mamans')]).
sentence([adj('fidèle'), n('consommateur'), det('de'), pron('vos'),
n('pizzas')]).
```

```
sentence([pron('je'), v('utilise'), pron('vos'), n('pâtes')]).
sentence([pron('je'),    v('peux'),    v('consommer'),    det('des'),
n('saucisses'),  pron('que'),  pron('je'),  aux('ai'),  v('laissé'),
n('dehors'), prep('de'), det('le'), n('réfrigérateur'), prep('depuis'),
n('mercredi')]).
sentence([pron('je'),     v('consomme'),     det('du'),     n('café'),
adj('décaféiné'), adj('soluble'), n('La_Marque')]).
sentence([pron('ils'),     v('prennent'),     det('du'),     n('lait'),
n('Nom_du_Produit')]).
sentence([pron('je'),    v('voudrais'),    v('savoir'),    conj('si'),
pron('je'), v('peux'), pron('leur'), v('donner'), det('le'), n('lait'),
n('transit'), prep('à'), det('la'), n('place')]).
sentence([prep('par'),    n('mesure'),    det('de'),    n('hygiène'),
pron('je'),    adv('ne'),    pron('le'),    aux('ai'),    adv('pas'),
v('consommer')]).
sentence([pron('elle'),    v('attrape'),    det('des'),    n('hauts'),
det('le'),  n('cœur'),  adj('terribles'),  conj('et'),  v('rende'),
adv('aussitôt'),  det('le'),  n('peu'),  pron('que'),  pron('elle'),
aux('a'), v('avalé')]).
sentence([adj('fidèle'),  n('consommateur'),  det('de'),  pron('vos'),
n('produits'), conj('soit'), det('les'), n('Nom_du_Produit')]).
sentence([adj('fidèle'),  prep('depuis'),  det('des'),  n('années'),
det('de'), det('votre'), n('Nom_du_Produit')]).
sentence([pron('elle'),   aux('a'),   adv('énormément'),   det('de'),
n('coliques')]).
sentence([pron('je'),      v('ai'),      det('des'),      n('chats'),
adj('diabétiques')]).
```

B.2 Liste des dérivés derives.txt

déçoit
déçois
décevons
décevez
déçoient
déçu
déçue
déçus
déçues
décevant
décevante
décevants
décevants
déception
ennui
désappointement
désappointer
déconvenue
désenchantement
désillusion
chagrin
insatisfaisant
insatisfaisante
frustrer
frustré
frustrés
frustrée
frustrées
frustrant
frustrantes

B.3 Codes du programme noyau module2.pl

```
/* module2.pl : extract candidate sentences */
/* Wannachai Kampeera */

date('20130112').

sentences_file('sentences.pl').
derived_file('derives.txt').
candidates_file_name('candidates.pl').
candidates_log_file_name('candidates.log').
rejects_log_file_name('rejects.log').
rejects_file_name('rejects.pl').

extract_sentences :-
      sentences_file(SentencesFile),
      read_term_file(SentencesFile, SentencesList),
      derived_file(DerivedFile),
      read_text_file(DerivedFile, DerivedList),
      compare_word(SentencesList, DerivedList, CandidateSentences,
      RejectedSentences),
      write_output_to_file(CandidateSentences, RejectedSentences),
      write_candidates_log(CandidateSentences),
      write_rejects_log(RejectedSentences),
      !.

read_term_file(FileName, SentencesList):-
      open(FileName, 'read', Stream),
      read(Stream, Terms),
      read_terms(Stream, Terms, SentencesList),
      close(Stream).

read_terms(_Stream, Terms, Term):-
      Terms == end_of_file,
      Term = [],
      !.
read_terms(Stream, TaggedSentence, [[TaggedSentence,
            WordList]|NextTerms]):-
      TaggedSentence = sentence(TaggedWordList),
      construct_word_list(TaggedWordList, WordList),
      read(Stream, NextTerm),
      read_terms(Stream, NextTerm, NextTerms).

construct_word_list([], []).
construct_word_list([TaggedWord|TaggedWordList], [Word|RestWords]):-
      TaggedWord =.. [_POS, Word],
      construct_word_list(TaggedWordList, RestWords).
```

```
read_text_file(FileName, SentencesList):-
      open(FileName, 'read', Stream),
      read_line_to_codes(Stream, Codes),
      read_lines(Stream, Codes, SentencesList),
      close(Stream).

read_lines(_Stream, Codes, NextCodes):-
      Codes == end_of_file,
      NextCodes = [],
      !.
read_lines(Stream, Codes, [Word|NextWord]):-
      name(Word, Codes),
      read_line_to_codes(Stream, NextCodes),
      read_lines(Stream, NextCodes, NextWord).

compare_word([], _DerivedList, [], []).
compare_word([SentenceWords|RestSentences], DerivedList, [[Word,
           TaggedSentence, WordList]|RestCandidates], Rejects):-
      SentenceWords = [TaggedSentence|[WordList]],
      compare_word(WordList, DerivedList, Word),
      compare_word(RestSentences, DerivedList, RestCandidates,
      Rejects).
compare_word([SentenceWords|RestSentences], DerivedList,
RestCandidates, [[[], TaggedSentence, WordList]|RestRejects]):-
      [TaggedSentence, WordList] = SentenceWords,
      \+ compare_word(WordList, DerivedList, _Word),
      compare_word(RestSentences, DerivedList, RestCandidates,
RestRejects).

compare_word(_Sentence, [], _):-
      fail.
compare_word(Sentence, [Word|_DerivedList], Word):-
      member(Word, Sentence),
      !.
compare_word(Sentence, [Word|DerivedList], NewWord):-
      \+ member(Word, Sentence),
      compare_word(Sentence, DerivedList, NewWord).

write_output_to_file(Candidates, Rejects):-
      candidates_file_name(CandidatesFileName),
      open(CandidatesFileName, 'write', Stream),
      writeq(Stream, candidates(Candidates)),
      write(Stream, '.'),
      close(Stream),
```

```
      rejects_file_name(RejectsFileName),
      open(RejectsFileName, 'write', Stream1),
      writeq(Stream1, rejects(Rejects)),
      write(Stream1, '.'),
      close(Stream1).

write_candidates_log(Candidates):-
      candidates_log_file_name(FileName),
      open(FileName, 'write', Stream),
      write_candidates_log(Candidates, Stream),
      close(Stream).

write_candidates_log([], _Stream).
write_candidates_log([Candidate|RestCandidates], Stream):-
      Candidate = [KeyWord, TaggedSentence, WordList],
      write(Stream, 'keyword: '),
      write(Stream, KeyWord),
      write(Stream, '\n'),
      write(Stream, 'tagged sentence: '),
      write(Stream, TaggedSentence),
      write(Stream, '\n'),
      write(Stream, 'words list: '),
      write(Stream, WordList),
      write(Stream, '\n'),
      write(Stream, '\n'),
      write_candidates_log(RestCandidates, Stream).

write_rejects_log(Rejects):-
      rejects_log_file_name(FileName),
      open(FileName, 'write', Stream),
      write_rejects_log(Rejects, Stream),
      close(Stream).

write_rejects_log([], _Stream).
write_rejects_log([Rejects|RestRejects], Stream):-
      Rejects = [KeyWord, TaggedSentence, WordList],
      write(Stream, 'keyword: '),
      write(Stream, KeyWord),
      write(Stream, '\n'),
      write(Stream, 'tagged sentence: '),
      write(Stream, TaggedSentence),
      write(Stream, '\n'),
      write(Stream, 'words list: '),
      write(Stream, WordList),
      write(Stream, '\n'),
```

```
      write(Stream, '\n'),
      write_candidates_log(RestRejects, Stream).

/* End of Programme */
```

B.4 Données de sortie du module 2 candidates.pl

candidates([['déçue',sentence([pron(je),v(suis),adj('déçue'),prep(par),
det(une),n(tablette),prep(de),n(chocolat),n('Nom_du_Produit')]),[je,sui
s,'déçue',par,une,tablette,de,chocolat,'Nom_du_Produit']],['déçoit',sen
tence([pron(cela),pron(me),v('déçoit')]),[cela,me,'déçoit']],['déçu',se
ntence([pron('celle-
ci'),pron(me),aux(a),adv(beaucoup),v('déçu')]),['celle-
ci',me,a,beaucoup,'déçu']],['déçoit',sentence([n('La_Marque'),pron(me),
v('déçoit')]),['La_Marque',me,'déçoit']],['décevant',sentence([pron(je)
,v(trouve),pron(ça),adv('très'),adj('décevant')]),[je,trouve,ça,'très',
'décevant']],['déception',sentence([prep(à),det(ma),adj(grande),n('déce
ption')]),[à,ma,grande,'déception']],['déçue',sentence([pron(je),v(suis
),adj('déçue'),prep(pour),det(la),adj('première'),n(fois),prep(depuis),
adj('25'),n(ans)]),[je,suis,'déçue',pour,la,'première',fois,depuis,'25'
,ans]],['déception',sentence([det(ce),n(courrier),prep(pour),pron(vous)
,v(faire),n(part),prep(de),det(ma),n('déception')]),[ce,courrier,pour,v
ous,faire,part,de,ma,'déception']],['déception',sentence([adj(quelle),n
('déception'),adj('même'),adv(pas),det(une),n('décoration'),prep(sur),d
et(les),n('bûches')]),[quelle,'déception','même',pas,une,'décoration',s
ur,les,'bûches']],['déception',sentence([pron(je),pron(vous),v(fais),n(
part),prep(de),det(ma),n('déception')]),[je,vous,fais,part,de,ma,'décep
tion']],['déception',sentence([pron(je),pron(vous),v(fais),n(part),prep
(de),det(ma),n('déception'),prep('vis-à-
vis'),prep(de),det(le),n('Nom_du_Produit')]),[je,vous,fais,part,de,ma,'
déception','vis-à-
vis',de,le,'Nom_du_Produit']],['déception',sentence([det(la),n('décepti
on'),prep(de),det(un),n(enfant)]),[la,'déception',de,un,enfant]],['déce
ption',sentence([adj(quel),adv(ne),v(est),adv(pas),det(ma),n('déception
')]),[quel,ne,est,pas,ma,'déception']],['déçu',sentence([pron(je),aux(a
i),v('été'),adj('déçu'),adv(lors),prep(de),det(mon),adj(dernier),n(acha
t)]),[je,ai,'été','déçu',lors,de,mon,dernier,achat]],['déçue',sentence(
[pron(je),aux(ai),v('été'),adj('déçue'),prep(par),det(le),n('côté'),adj
(âpre),prep(de),det(cet),n(article)]),[je,ai,'été','déçue',par,le,'côté
',âpre,de,cet,article]],['déçue',sentence([pron(je),pron(me),aux(suis),
v('trouvée'),adv('très'),adj('déçue'),prep(de),det(ce),n(produit)]),[je
,me,suis,'trouvée','très','déçue',de,ce,produit]],['déçue',sentence([pr
on(je),v(suis),adv('amèrement'),adj('déçue')]),[je,suis,'amèrement','dé
çue']],['déçue',sentence([pron(je),v(suis),det(une),n(consommatrice),ad
v('très'),adj('déçue'),prep(de),det(votre),adj(grande),n(marque)]),[je,
suis,une,consommatrice,'très','déçue',de,votre,grande,marque]],['déçu',
sentence([pron(je),v(suis),adv(vraiment),adj('déçu'),prep(de),det(vos),
n(pizzas),adj('surgelées'),n('Nom_du_Produit')]),[je,suis,vraiment,'déç
u',de,vos,pizzas,'surgelées','Nom_du_Produit']],['déçue',sentence([pron
(je),v(suis),adj('déçue')]),[je,suis,'déçue']],['déçue',sentence([pron(
je),v(suis),adv('très'),adj('déçue')]),[je,suis,'très','déçue']],['déçu
',sentence([pron(je),aux(ai),v('été'),adv('très'),adj('déçu'),det(ce),n

(jour)]),[je,ai,'été','très','déçu',ce,jour]],['déçue',sentence([pron(j
e),pron(en),aux(ai),v('été'),adj('déçue')]),[je,en,ai,'été','déçue']],[
'déçue',sentence([pron(je),v(suis),adv('particulièrement'),adj('déçue')
,prep(de),det(mon),adj(dernier),n(achat)]),[je,suis,'particulièrement',
'déçue',de,mon,dernier,achat]],['déçus',sentence([pron(nous),aux(avons)
,v('été'),adj('déçus')]),[nous,avons,'été','déçus']],['déçue',sentence(
[pron(je),v(suis),adv(donc),adv('très'),adj('déçue'),prep(quant),prep(à
),det(la),n(éthique),prep(de),det(votre),n('société')]),[je,suis,donc,'
très','déçue',quant,à,la,éthique,de,votre,'société']],['déçue',sentence
([pron(je),v(suis),adj('déçue'),prep(de),det(votre),n(marque),n('La_Mar
que')]),[je,suis,'déçue',de,votre,marque,'La_Marque']],['déçu',sentence
([pron(je),v(suis),adv('très'),adj('déçu'),prep(de),det(les),n(truffes)
,adj(noir),adj('70'),n('%')]),[je,suis,'très','déçu',de,les,truffes,noi
r,'70','%']],['déçue',sentence([pron(je),v(suis),adv('très'),adj('déçue
'),prep(de),det(ce),n(produit)]),[je,suis,'très','déçue',de,ce,produit]
],['déçue',sentence([pron(je),v(suis),adj('déçue'),prep(de),det(leur),n
(remarque)]),[je,suis,'déçue',de,leur,remarque]],['déçu',sentence([pron
(je),v(suis),adj('déçu'),prep(de),det(mes),adj(deux),adj(derniers),n(ac
hats),adj(identiques)]),[je,suis,'déçu',de,mes,deux,derniers,achats,ide
ntiques]],['déçu',sentence([pron(je),v(suis),adv('très'),adj('déçu'),pr
ep(par),det(le),n(menu)]),[je,suis,'très','déçu',par,le,menu]]]).

B.5 Données de sortie du module 2 candidates.log

```
keyword: déçue
tagged sentence:
sentence([pron(je),v(suis),adj(déçue),prep(par),det(une),n(tablette),pr
ep(de),n(chocolat),n(Nom_du_Produit)])
words list: [je,suis,déçue,par,une,tablette,de,chocolat,Nom_du_Produit]

keyword: déçoit
tagged sentence: sentence([pron(cela),pron(me),v(déçoit)])
words list: [cela,me,déçoit]

keyword: déçu
tagged sentence: sentence([pron(celle-
ci),pron(me),aux(a),adv(beaucoup),v(déçu)])
words list: [celle-ci,me,a,beaucoup,déçu]

keyword: déçoit
tagged sentence: sentence([n(La_Marque),pron(me),v(déçoit)])
words list: [La_Marque,me,déçoit]

keyword: décevant
tagged sentence:
sentence([pron(je),v(trouve),pron(ça),adv(très),adj(décevant)])
words list: [je,trouve,ça,très,décevant]

keyword: déception
tagged sentence: sentence([prep(à),det(ma),adj(grande),n(déception)])
words list: [à,ma,grande,déception]

keyword: déçue
tagged sentence:
sentence([pron(je),v(suis),adj(déçue),prep(pour),det(la),adj(première),
n(fois),prep(depuis),adj(25),n(ans)])
words list: [je,suis,déçue,pour,la,première,fois,depuis,25,ans]

keyword: déception
tagged sentence:
sentence([det(ce),n(courrier),prep(pour),pron(vous),v(faire),n(part),pr
ep(de),det(ma),n(déception)])
words list: [ce,courrier,pour,vous,faire,part,de,ma,déception]

keyword: déception
tagged sentence:
sentence([adj(quelle),n(déception),adj(même),adv(pas),det(une),n(décora
tion),prep(sur),det(les),n(bûches)])
words list: [quelle,déception,même,pas,une,décoration,sur,les,bûches]
```

```
keyword: déception
tagged sentence:
sentence([pron(je),pron(vous),v(fais),n(part),prep(de),det(ma),n(décept
ion)])
words list: [je,vous,fais,part,de,ma,déception]

keyword: déception
tagged sentence:
sentence([pron(je),pron(vous),v(fais),n(part),prep(de),det(ma),n(décept
ion),prep(vis-à-vis),prep(de),det(le),n(Nom_du_Produit)])
words list: [je,vous,fais,part,de,ma,déception,vis-à-
vis,de,le,Nom_du_Produit]

keyword: déception
tagged sentence:
sentence([det(la),n(déception),prep(de),det(un),n(enfant)])
words list: [la,déception,de,un,enfant]

keyword: déception
tagged sentence:
sentence([adj(quel),adv(ne),v(est),adv(pas),det(ma),n(déception)])
words list: [quel,ne,est,pas,ma,déception]

keyword: déçu
tagged sentence:
sentence([pron(je),aux(ai),v(été),adj(déçu),adv(lors),prep(de),det(mon)
,adj(dernier),n(achat)])
words list: [je,ai,été,déçu,lors,de,mon,dernier,achat]

keyword: déçue
tagged sentence:
sentence([pron(je),aux(ai),v(été),adj(déçue),prep(par),det(le),n(côté),
adj(âpre),prep(de),det(cet),n(article)])
words list: [je,ai,été,déçue,par,le,côté,âpre,de,cet,article]

keyword: déçue
tagged sentence:
sentence([pron(je),pron(me),aux(suis),v(trouvée),adv(très),adj(déçue),p
rep(de),det(ce),n(produit)])
words list: [je,me,suis,trouvée,très,déçue,de,ce,produit]

keyword: déçue
tagged sentence: sentence([pron(je),v(suis),adv(amèrement),adj(déçue)])
words list: [je,suis,amèrement,déçue]
```

```
keyword: déçue
tagged sentence:
sentence([pron(je),v(suis),det(une),n(consommatrice),adv(très),adj(déçu
e),prep(de),det(votre),adj(grande),n(marque)])
words list:
[je,suis,une,consommatrice,très,déçue,de,votre,grande,marque]

keyword: déçu
tagged sentence:
sentence([pron(je),v(suis),adv(vraiment),adj(déçu),prep(de),det(vos),n(
pizzas),adj(surgelées),n(Nom_du_Produit)])
words list:
[je,suis,vraiment,déçu,de,vos,pizzas,surgelées,Nom_du_Produit]

keyword: déçue
tagged sentence: sentence([pron(je),v(suis),adj(déçue)])
words list: [je,suis,déçue]

keyword: déçue
tagged sentence: sentence([pron(je),v(suis),adv(très),adj(déçue)])
words list: [je,suis,très,déçue]

keyword: déçu
tagged sentence:
sentence([pron(je),aux(ai),v(été),adv(très),adj(déçu),det(ce),n(jour)])
words list: [je,ai,été,très,déçu,ce,jour]

keyword: déçue
tagged sentence:
sentence([pron(je),pron(en),aux(ai),v(été),adj(déçue)])
words list: [je,en,ai,été,déçue]

keyword: déçue
tagged sentence:
sentence([pron(je),v(suis),adv(particulièrement),adj(déçue),prep(de),de
t(mon),adj(dernier),n(achat)])
words list: [je,suis,particulièrement,déçue,de,mon,dernier,achat]

keyword: déçus
tagged sentence: sentence([pron(nous),aux(avons),v(été),adj(déçus)])
words list: [nous,avons,été,déçus]

keyword: déçue
```

```
tagged sentence:
sentence([pron(je),v(suis),adv(donc),adv(très),adj(déçue),prep(quant),p
rep(à),det(la),n(éthique),prep(de),det(votre),n(société)])
words list:
[je,suis,donc,très,déçue,quant,à,la,éthique,de,votre,société]

keyword: déçue
tagged sentence:
sentence([pron(je),v(suis),adj(déçue),prep(de),det(votre),n(marque),n(L
a_Marque)])
words list: [je,suis,déçue,de,votre,marque,La_Marque]

keyword: déçu
tagged sentence:
sentence([pron(je),v(suis),adv(très),adj(déçu),prep(de),det(les),n(truf
fes),adj(noir),adj(70),n(%)])
words list: [je,suis,très,déçu,de,les,truffes,noir,70,%]

keyword: déçue
tagged sentence:
sentence([pron(je),v(suis),adv(très),adj(déçue),prep(de),det(ce),n(prod
uit)])
words list: [je,suis,très,déçue,de,ce,produit]

keyword: déçue
tagged sentence:
sentence([pron(je),v(suis),adj(déçue),prep(de),det(leur),n(remarque)])
words list: [je,suis,déçue,de,leur,remarque]

keyword: déçu
tagged sentence:
sentence([pron(je),v(suis),adj(déçu),prep(de),det(mes),adj(deux),adj(de
rniers),n(achats),adj(identiques)])
words list: [je,suis,déçu,de,mes,deux,derniers,achats,identiques]

keyword: déçu
tagged sentence:
sentence([pron(je),v(suis),adv(très),adj(déçu),prep(par),det(le),n(menu
)])
words list: [je,suis,très,déçu,par,le,menu]
```

B.6 Données de sortie du module 2 rejects.pl

rejects([[[],sentence([pron(je),v(suis),det(une),adj(grande),n(consomma
trice),det(de),pron(vos),n(yaourts),adj('fruités'),det('0'),n('%')]),[j
e,suis,une,grande,consommatrice,de,vos,yaourts,'fruités','0','%']],[[],
sentence([pron(je),v(suis),det(une),adj('fidèle'),n(consommatrice)]),[j
e,suis,une,'fidèle',consommatrice]],[[],sentence([pron(je),v(bois),adv(
'régulièrement'),det(ce),n(chocolat),prep(à),det(le),n(bureau),pron(que
),pron(je),v(trouvais),adj('délicieux')]),[je,bois,'régulièrement',ce,c
hocolat,à,le,bureau,que,je,trouvais,'délicieux']],[[],sentence([pron(je
),pron(le),aux(ai),v('entamé'),adv(hier),conj(ou),adv('avant-
hier')]),[je,le,ai,'entamé',hier,ou,'avant-
hier']],[[],sentence([pron(je),v(utilise),det(ce),n(lait),prep(depuis),
det(sa),n(naissance)]),[je,utilise,ce,lait,depuis,sa,naissance]],[[],se
ntence([pron(il),adv(ne),v(mange),adv(que),pron(cela)]),[il,ne,mange,qu
e,cela]],[[],sentence([adj('fidèle'),n(consommateur),det(de),pron(vos),
n(produits)]),['fidèle',consommateur,de,vos,produits]],[[],sentence([pr
on(je),adv(ne),v(mange),adv(plus),prep(depuis),det(deux),n(jours)]),[je
,ne,mange,plus,depuis,deux,jours]],[[],sentence([pron(je),v(recherche),
adv(donc),det(cette),n('crème'),conj(car),pron(il),pron(la),aux(a),v('a
pprécie'),adv(beaucoup),prep(depuis),pron(que),pron(il),v(mange),prep(à
),det(la),n('cuillère')]),[je,recherche,donc,cette,'crème',car,il,la,a,
'apprécie',beaucoup,depuis,que,il,mange,à,la,'cuillère']],[[],sentence(
[n(consommatrice),adj(inconditionnelle),prep(avec),det(mes),adj(petits)
,n(enfants),det(de),det(votre),n('Nom_du_Produit')]),[consommatrice,inc
onditionnelle,avec,mes,petits,enfants,de,votre,'Nom_du_Produit']],[[],s
entence([pron(nous),v(utilisons),det(le),n(lait),n('Nom_du_Produit'),pr
ep(depuis),det(la),n(naissance),det(de),det(notre),n(fils)]),[nous,util
isons,le,lait,'Nom_du_Produit',depuis,la,naissance,de,notre,fils]],[[],
sentence([pron(je),v(utilise),det(ce),n(lait),prep(depuis),adv(longtemp
s)]),[je,utilise,ce,lait,depuis,longtemps]],[[],sentence([pron(je),v(ai
),det(la),n(habitude),det(de),v(consommer),pron(vos),n(produits)]),[je,
ai,la,habitude,de,consommer,vos,produits]],[[],sentence([det(mon),n(enf
ant),aux(a),v(bu),det(plusieurs),n(biberons),prep(depuis),n(dimanche)])
,[mon,enfant,a,bu,plusieurs,biberons,depuis,dimanche]],[[],sentence([pr
on(on),aux(a),v('mangé'),pron(tous),adv(pareil),conj(sauf_que),pron(eux
),aux(ont),v('mangé'),det(le),n(jambon)]),[on,a,'mangé',tous,pareil,sau
f_que,eux,ont,'mangé',le,jambon]],[[],sentence([pron(nous),pron(nous),a
dv(ne),pron(en),aix(avons),adv(pas),v('mangé')]),[nous,nous,ne,en,avons
,pas,'mangé']],[[],sentence([pron(je),v(consomme),n('Nom_du_Produit'),p
rep(depuis),det('15'),n(ans)]),[je,consomme,'Nom_du_Produit',depuis,'15
',ans]],[[],sentence([pron(je),v(utilise),adv(habituellement),det(ce),n
(lait)]),[je,utilise,habituellement,ce,lait]],[[],sentence([det(mes),n(
chiens),adv(ne),v(mangent),adv(que),pron(ça)]),[mes,chiens,ne,mangent,q
ue,ça]],[[],sentence([pron(je),v(suis),det(une),adv('très'),adj(grande)
,n(consommatrice),det(de),pron(vos),n(barres)]),[je,suis,une,'très',gra
nde,consommatrice,de,vos,barres]],[[],sentence([prep('après'),det(les),

aux(avoir),v('consommés'),det(à),adj(fameux),n(repas),pron(ils),aux(ont
),v(eu),det(des),n(rillettes)]),['après',les,avoir,'consommés',à,fameux
,repas,ils,ont,eu,des,rillettes]],[[],sentence([det(mes),det(trois),n(c
hats),v(manger),det(nos),n(restes),det(de),n(table)]),[mes,trois,chats,
manger,nos,restes,de,table]],[[],sentence([pron(je),v(suis),det(une),ad
j(grande),n(consommateur),det(de),n('Nom_du_Produit')]),[je,suis,une,gr
ande,consommateur,de,'Nom_du_Produit']],[[],sentence([pron(ce),v(est),d
et(une),n(lait),pron(que),pron(je),v(utilise),prep(depuis),det('1'),n(m
ois)]),[ce,est,une,lait,que,je,utilise,depuis,'1',mois]],[[],sentence([
pron(je),v(suis),det(une),adj(fervente),n(consommatrice)]),[je,suis,une
,fervente,consommatrice]],[[],sentence([det(les),n('Français'),v(mangen
t),adv(trop),det(de),n(sel)]),[les,'Français',mangent,trop,de,sel]],[[]
,sentence([adv('Très'),n(amateur),prep(de),det(les),adv(habituellement)
,adj(excellents),n('Nom_du_Produit'),pron(que),pron(je),v(consomme),adv
(sans),n('modération'),prep(en),n('période'),det(de),n(fin),prep(de),n(
'année')]),['Très',amateur,de,les,habituellement,excellents,'Nom_du_Pro
duit',que,je,consomme,sans,'modération',en,'période',de,fin,de,'année']
],[[],sentence([det(mon),n(fils),v(mange),det(des),n(biscuits)]),[mon,f
ils,mange,des,biscuits]],[[],sentence([n(consommateurs),adj('réguliers'
),det(mes),n(enfants),conj(et),pron(moi)]),[consommateurs,'réguliers',m
es,enfants,et,moi]],[[],sentence([n(consommatrice),det(de),det(vos),n('
cafés'),adj(solubles),pron(dont),det(le),n('Nom_du_Produit'),prep(depui
s),det(de),adj(nombreuses),n('années')]),[consommatrice,de,vos,'cafés',
solubles,dont,le,'Nom_du_Produit',depuis,de,nombreuses,'années']],[[],s
entence([pron(ce),pron(qui),v(devient),adj(difficile),prep(pour),n('Tom
'),det(de),det(les),v('déguster')]),[ce,qui,devient,difficile,pour,'Tom
',de,les,'déguster']],[[],sentence([pron(je),v(suis),n(utilisateur),pre
p(de),det(le),n(lait),adj(infantile),n('La_Marque'),prep(en),n(poudre)]
),[je,suis,utilisateur,de,le,lait,infantile,'La_Marque',en,poudre]],[[]
,sentence([pron(je),pron(en),v(mange),adv(beaucoup)]),[je,en,mange,beau
coup]],[[],sentence([pron(je),pron(le),aux(ai),v('goûté'),conj(et),pron
(je),aux(ai),adv(tout),v('recraché')]),[je,le,ai,'goûté',et,je,ai,tout,
'recraché']],[[],sentence([pron(ce),v(est),det(la),adj('troisième'),n(b
oite),pron(que),pron(je),v(utilise)]),[ce,est,la,'troisième',boite,que,
je,utilise]],[[],sentence([pron(ce),n(paquet),det(de),n('céréales'),v(c
ontenait),det(une),n(fil),det(de),n(nylon),adj('ci-
joint')]),[ce,paquet,de,'céréales',contenait,une,fil,de,nylon,'ci-
joint']],[[],sentence([adj('habituée'),pron(je),v(consomme),n('Nom_du_P
roduit'),prep(depuis),det('15'),n(ans)]),['habituée',je,consomme,'Nom_d
u_Produit',depuis,'15',ans]],[[],sentence([adj('habituée'),det(les),n(l
ardons),v(ressemblaient),prep(à),det(des),adj(petites),n(chiquettes)]),
['habituée',les,lardons,ressemblaient,à,des,petites,chiquettes]],[[],se
ntence([adj('habituée'),pron(il),adv(ne),v(mange),pron(que),pron(cela)]
),['habituée',il,ne,mange,que,cela]],[[],sentence([adv(habituellement),
det(ce),v(est),adv(bien),adj(meilleur)]),[habituellement,ce,est,bien,me

illeur]],[[],sentence([pron(je),v('achète'),adv(habituellement),det(votre),n(produit),adj('Nom_du_Produit'),adj(adulte),n(poulet)]),[je,'achète',habituellement,votre,produit,'Nom_du_Produit',adulte,poulet]],[[],sentence([pron(je),v('achète'),adv('régulièrement'),det(du),n('Nom_du_Produit'),prep(en),n(terrine),conj(car),pron(ils),adv(ne),v(aiment),adv(que),det(la),n(terrine)]),[je,'achète','régulièrement',du,'Nom_du_Produit',en,terrine,car,ils,ne,aiment,que,la,terrine]],[[],sentence([pron(je),v('achète'),adv(souvent),det(ces),n(lasagnes)]),[je,'achète',souvent,ces,lasagnes]],[[],sentence([n(consommatrice),adj(inconditionnelle),prep(avec),det(mes),adj(petits),n(enfants),det(de),det(votre),n('Nom_du_Produit')]),[consommatrice,inconditionnelle,avec,mes,petits,enfants,de,votre,'Nom_du_Produit']],[[],sentence([pron(elle),v(mange),det(les),n('Nom_du_Produit'),prep(de),n(habitude)]),[elle,mange,les,'Nom_du_Produit',de,habitude]],[[],sentence([pron(elle),pron(y),v(est),adj('habituée'),det(ce),v(est),adv(pourquoi),pron(je),v(insiste),prep('auprès'),det(de),pron(vous)]),[elle,y,est,'habituée',ce,est,pourquoi,je,insiste,'auprès',de,vous]],[[],sentence([adv(en_tant_que),n(cliente),adj('fidèle')]),[en_tant_que,cliente,'fidèle']],[[],sentence([adv('est-ce_que'),n('Nom_du_Produit'),v(fait),adv(comme),n('La_Marque'),prep(à),v(savoir),det(des),n(collecteurs),prep(pour),v('fidéliser'),det(les),n(mamans)]),['est-ce_que','Nom_du_Produit',fait,comme,'La_Marque',à,savoir,des,collecteurs,pour,'fidéliser',les,mamans]],[[],sentence([adj('fidèle'),n(consommateur),det(de),pron(vos),n(pizzas)]),['fidèle',consommateur,de,vos,pizzas]],[[],sentence([pron(je),v(utilise),pron(vos),n('pâtes')]),[je,utilise,vos,'pâtes']],[[],sentence([pron(je),v(peux),v(consommer),det(des),n(saucisses),pron(que),pron(je),aux(ai),v('laissé'),n(dehors),prep(de),det(le),n('réfrigérateur'),prep(depuis),n(mercredi)]),[je,peux,consommer,des,saucisses,que,je,ai,'laissé',dehors,de,le,'réfrigérateur',depuis,mercredi]],[[],sentence([pron(je),v(consomme),det(du),n('café'),adj('décaféiné'),adj(soluble),n('La_Marque')]),[je,consomme,du,'café','décaféiné',soluble,'La_Marque']],[[],sentence([pron(ils),v(prennent),det(du),n(lait),n('Nom_du_Produit')]),[ils,prennent,du,lait,'Nom_du_Produit']],[[],sentence([pron(je),v(voudrais),v(savoir),conj(si),pron(je),v(peux),pron(leur),v(donner),det(le),n(lait),n(transit),prep(à),det(la),n(place)]),[je,voudrais,savoir,si,je,peux,leur,donner,le,lait,transit,à,la,place]],[[],sentence([prep(par),n(mesure),det(de),n('hygiène'),pron(je),adv(ne),pron(le),aux(ai),adv(pas),v(consommer)]),[par,mesure,de,'hygiène',je,ne,le,ai,pas,consommer]],[[],sentence([pron(elle),v(attrape),det(des),n(hauts),det(le),n(cœur),adj(terribles),conj(et),v(rende),adv('aussitôt'),det(le),n(peu),pron(que),pron(elle),aux(a),v('avalé')]),[elle,attrape,des,hauts,le,cœur,terribles,et,rende,'aussitôt',le,peu,que,elle,a,'avalé']],[[],sentence([adj('fidèle'),n(consommateur),det(de),pron(vos),n(produits),conj(soit),det(les),n('Nom_du_Produit')]),['fidèle',consommateur,de,vos,produits,soit,les,'Nom_du_Produit']],[[],sentence([adj('fidèl

e'),prep(depuis),det(des),n('années'),det(de),det(votre),n('Nom_du_Produit')]),['fidèle',depuis,des,'années',de,votre,'Nom_du_Produit']],[[],sentence([pron(elle),aux(a),adv('énormément'),det(de),n(coliques)]),[elle,a,'énormément',de,coliques]],[[],sentence([pron(je),v(ai),det(des),n(chats),adj('diabétiques')]),[je,ai,des,chats,'diabétiques']]]).

B.7 Données de sortie du module 2 rejects.log

```
keyword: []
tagged sentence:
sentence([pron(je),v(suis),det(une),adj(grande),n(consommatrice),det(de
),pron(vos),n(yaourts),adj(fruités),det(0),n(%)])
words list:
[je,suis,une,grande,consommatrice,de,vos,yaourts,fruités,0,%]

keyword: []
tagged sentence:
sentence([pron(je),v(suis),det(une),adj(fidèle),n(consommatrice)])
words list: [je,suis,une,fidèle,consommatrice]

keyword: []
tagged sentence:
sentence([pron(je),v(bois),adv(régulièrement),det(ce),n(chocolat),prep(
à),det(le),n(bureau),pron(que),pron(je),v(trouvais),adj(délicieux)])
words list:
[je,bois,régulièrement,ce,chocolat,à,le,bureau,que,je,trouvais,délicieu
x]

keyword: []
tagged sentence:
sentence([pron(je),pron(le),aux(ai),v(entamé),adv(hier),conj(ou),adv(av
ant-hier)])
words list: [je,le,ai,entamé,hier,ou,avant-hier]

keyword: []
tagged sentence:
sentence([pron(je),v(utilise),det(ce),n(lait),prep(depuis),det(sa),n(na
issance)])
words list: [je,utilise,ce,lait,depuis,sa,naissance]

keyword: []
tagged sentence:
sentence([pron(il),adv(ne),v(mange),adv(que),pron(cela)])
words list: [il,ne,mange,que,cela]

keyword: []
tagged sentence:
sentence([adj(fidèle),n(consommateur),det(de),pron(vos),n(produits)])
words list: [fidèle,consommateur,de,vos,produits]

keyword: []
```

```
tagged sentence:
sentence([pron(je),adv(ne),v(mange),adv(plus),prep(depuis),det(deux),n(
jours)])
words list: [je,ne,mange,plus,depuis,deux,jours]

keyword: []
tagged sentence:
sentence([pron(je),v(recherche),adv(donc),det(cette),n(crème),conj(car)
,pron(il),pron(la),aux(a),v(apprécie),adv(beaucoup),prep(depuis),pron(q
ue),pron(il),v(mange),prep(à),det(la),n(cuillère)])
words list:
[je,recherche,donc,cette,crème,car,il,la,a,apprécie,beaucoup,depuis,que
,il,mange,à,la,cuillère]

keyword: []
tagged sentence:
sentence([n(consommatrice),adj(inconditionnelle),prep(avec),det(mes),ad
j(petits),n(enfants),det(de),det(votre),n(Nom_du_Produit)])
words list:
[consommatrice,inconditionnelle,avec,mes,petits,enfants,de,votre,Nom_du
_Produit]

keyword: []
tagged sentence:
sentence([pron(nous),v(utilisons),det(le),n(lait),n(Nom_du_Produit),pre
p(depuis),det(la),n(naissance),det(de),det(notre),n(fils)])
words list:
[nous,utilisons,le,lait,Nom_du_Produit,depuis,la,naissance,de,notre,fil
s]

keyword: []
tagged sentence:
sentence([pron(je),v(utilise),det(ce),n(lait),prep(depuis),adv(longtemp
s)])
words list: [je,utilise,ce,lait,depuis,longtemps]

keyword: []
tagged sentence:
sentence([pron(je),v(ai),det(la),n(habitude),det(de),v(consommer),pron(
vos),n(produits)])
words list: [je,ai,la,habitude,de,consommer,vos,produits]

keyword: []
```

```
tagged sentence:
sentence([det(mon),n(enfant),aux(a),v(bu),det(plusieurs),n(biberons),pr
ep(depuis),n(dimanche)])
words list: [mon,enfant,a,bu,plusieurs,biberons,depuis,dimanche]

keyword: []
tagged sentence:
sentence([pron(on),aux(a),v(mangé),pron(tous),adv(pareil),conj(sauf_que
),pron(eux),aux(ont),v(mangé),det(le),n(jambon)])
words list: [on,a,mangé,tous,pareil,sauf_que,eux,ont,mangé,le,jambon]

keyword: []
tagged sentence:
sentence([pron(nous),pron(nous),adv(ne),pron(en),aix(avons),adv(pas),v(
mangé)])
words list: [nous,nous,ne,en,avons,pas,mangé]

keyword: []
tagged sentence:
sentence([pron(je),v(consomme),n(Nom_du_Produit),prep(depuis),det(15),n
(ans)])
words list: [je,consomme,Nom_du_Produit,depuis,15,ans]

keyword: []
tagged sentence:
sentence([pron(je),v(utilise),adv(habituellement),det(ce),n(lait)])
words list: [je,utilise,habituellement,ce,lait]

keyword: []
tagged sentence:
sentence([det(mes),n(chiens),adv(ne),v(mangent),adv(que),pron(ça)])
words list: [mes,chiens,ne,mangent,que,ça]

keyword: []
tagged sentence:
sentence([pron(je),v(suis),det(une),adv(très),adj(grande),n(consommatri
ce),det(de),pron(vos),n(barres)])
words list: [je,suis,une,très,grande,consommatrice,de,vos,barres]

keyword: []
tagged sentence:
sentence([prep(après),det(les),aux(avoir),v(consommés),det(à),adj(fameu
x),n(repas),pron(ils),aux(ont),v(eu),det(des),n(rillettes)])
words list:
[après,les,avoir,consommés,à,fameux,repas,ils,ont,eu,des,rillettes]
```

```
keyword: []
tagged sentence:
sentence([det(mes),det(trois),n(chats),v(manger),det(nos),n(restes),det
(de),n(table)])
words list: [mes,trois,chats,manger,nos,restes,de,table]

keyword: []
tagged sentence:
sentence([pron(je),v(suis),det(une),adj(grande),n(consommateur),det(de)
,n(Nom_du_Produit)])
words list: [je,suis,une,grande,consommateur,de,Nom_du_Produit]

keyword: []
tagged sentence:
sentence([pron(ce),v(est),det(une),n(lait),pron(que),pron(je),v(utilise
),prep(depuis),det(1),n(mois)])
words list: [ce,est,une,lait,que,je,utilise,depuis,1,mois]

keyword: []
tagged sentence:
sentence([pron(je),v(suis),det(une),adj(fervente),n(consommatrice)])
words list: [je,suis,une,fervente,consommatrice]

keyword: []
tagged sentence:
sentence([det(les),n(Français),v(mangent),adv(trop),det(de),n(sel)])
words list: [les,Français,mangent,trop,de,sel]

keyword: []
tagged sentence:
sentence([adv(Très),n(amateur),prep(de),det(les),adv(habituellement),ad
j(excellents),n(Nom_du_Produit),pron(que),pron(je),v(consomme),adv(sans
),n(modération),prep(en),n(période),det(de),n(fin),prep(de),n(année)])
words list:
[Très,amateur,de,les,habituellement,excellents,Nom_du_Produit,que,je,co
nsomme,sans,modération,en,période,de,fin,de,année]

keyword: []
tagged sentence:
sentence([det(mon),n(fils),v(mange),det(des),n(biscuits)])
words list: [mon,fils,mange,des,biscuits]

keyword: []
```

```
tagged sentence:
sentence([n(consommateurs),adj(réguliers),det(mes),n(enfants),conj(et),
pron(moi)])
words list: [consommateurs,réguliers,mes,enfants,et,moi]

keyword: []
tagged sentence:
sentence([n(consommatrice),det(de),det(vos),n(cafés),adj(solubles),pron
(dont),det(le),n(Nom_du_Produit),prep(depuis),det(de),adj(nombreuses),n
(années)])
words list:
[consommatrice,de,vos,cafés,solubles,dont,le,Nom_du_Produit,depuis,de,n
ombreuses,années]

keyword: []
tagged sentence:
sentence([pron(ce),pron(qui),v(devient),adj(difficile),prep(pour),n(Tom
),det(de),det(les),v(déguster)])
words list: [ce,qui,devient,difficile,pour,Tom,de,les,déguster]

keyword: []
tagged sentence:
sentence([pron(je),v(suis),n(utilisateur),prep(de),det(le),n(lait),adj(
infantile),n(La_Marque),prep(en),n(poudre)])
words list:
[je,suis,utilisateur,de,le,lait,infantile,La_Marque,en,poudre]

keyword: []
tagged sentence: sentence([pron(je),pron(en),v(mange),adv(beaucoup)])
words list: [je,en,mange,beaucoup]

keyword: []
tagged sentence:
sentence([pron(je),pron(le),aux(ai),v(goûté),conj(et),pron(je),aux(ai),
adv(tout),v(recraché)])
words list: [je,le,ai,goûté,et,je,ai,tout,recraché]

keyword: []
tagged sentence:
sentence([pron(ce),v(est),det(la),adj(troisième),n(boite),pron(que),pro
n(je),v(utilise)])
words list: [ce,est,la,troisième,boite,que,je,utilise]

keyword: []
```

```
tagged sentence:
sentence([pron(ce),n(paquet),det(de),n(céréales),v(contenait),det(une),
n(fil),det(de),n(nylon),adj(ci-joint)])
words list: [ce,paquet,de,céréales,contenait,une,fil,de,nylon,ci-joint]

keyword: []
tagged sentence:
sentence([adj(habituée),pron(je),v(consomme),n(Nom_du_Produit),prep(dep
uis),det(15),n(ans)])
words list: [habituée,je,consomme,Nom_du_Produit,depuis,15,ans]

keyword: []
tagged sentence:
sentence([adj(habituée),det(les),n(lardons),v(ressemblaient),prep(à),de
t(des),adj(petites),n(chiquettes)])
words list:
[habituée,les,lardons,ressemblaient,à,des,petites,chiquettes]

keyword: []
tagged sentence:
sentence([adj(habituée),pron(il),adv(ne),v(mange),pron(que),pron(cela)]
)
words list: [habituée,il,ne,mange,que,cela]

keyword: []
tagged sentence:
sentence([adv(habituellement),det(ce),v(est),adv(bien),adj(meilleur)])
words list: [habituellement,ce,est,bien,meilleur]

keyword: []
tagged sentence:
sentence([pron(je),v(achète),adv(habituellement),det(votre),n(produit),
adj(Nom_du_Produit),adj(adulte),n(poulet)])
words list:
[je,achète,habituellement,votre,produit,Nom_du_Produit,adulte,poulet]

keyword: []
tagged sentence:
sentence([pron(je),v(achète),adv(régulièrement),det(du),n(Nom_du_Produi
t),prep(en),n(terrine),conj(car),pron(ils),adv(ne),v(aiment),adv(que),d
et(la),n(terrine)])
words list:
[je,achète,régulièrement,du,Nom_du_Produit,en,terrine,car,ils,ne,aiment
,que,la,terrine]
```

```
keyword: []
tagged sentence:
sentence([pron(je),v(achète),adv(souvent),det(ces),n(lasagnes)])
words list: [je,achète,souvent,ces,lasagnes]

keyword: []
tagged sentence:
sentence([n(consommatrice),adj(inconditionnelle),prep(avec),det(mes),ad
j(petits),n(enfants),det(de),det(votre),n(Nom_du_Produit)])
words list:
[consommatrice,inconditionnelle,avec,mes,petits,enfants,de,votre,Nom_du
_Produit]

keyword: []
tagged sentence:
sentence([pron(elle),v(mange),det(les),n(Nom_du_Produit),prep(de),n(hab
itude)])
words list: [elle,mange,les,Nom_du_Produit,de,habitude]

keyword: []
tagged sentence:
sentence([pron(elle),pron(y),v(est),adj(habituée),det(ce),v(est),adv(po
urquoi),pron(je),v(insiste),prep(auprès),det(de),pron(vous)])
words list:
[elle,y,est,habituée,ce,est,pourquoi,je,insiste,auprès,de,vous]

keyword: []
tagged sentence: sentence([adv(en_tant_que),n(cliente),adj(fidèle)])
words list: [en_tant_que,cliente,fidèle]

keyword: []
tagged sentence: sentence([adv(est-
ce_que),n(Nom_du_Produit),v(fait),adv(comme),n(La_Marque),prep(à),v(sav
oir),det(des),n(collecteurs),prep(pour),v(fidéliser),det(les),n(mamans)
])
words list: [est-
ce_que,Nom_du_Produit,fait,comme,La_Marque,à,savoir,des,collecteurs,pou
r,fidéliser,les,mamans]

keyword: []
tagged sentence:
sentence([adj(fidèle),n(consommateur),det(de),pron(vos),n(pizzas)])
words list: [fidèle,consommateur,de,vos,pizzas]

keyword: []
```

```
tagged sentence: sentence([pron(je),v(utilise),pron(vos),n(pâtes)])
words list: [je,utilise,vos,pâtes]

keyword: []
tagged sentence:
sentence([pron(je),v(peux),v(consommer),det(des),n(saucisses),pron(que)
,pron(je),aux(ai),v(laissé),n(dehors),prep(de),det(le),n(réfrigérateur)
,prep(depuis),n(mercredi)])
words list:
[je,peux,consommer,des,saucisses,que,je,ai,laissé,dehors,de,le,réfrigér
ateur,depuis,mercredi]

keyword: []
tagged sentence:
sentence([pron(je),v(consomme),det(du),n(café),adj(décaféiné),adj(solub
le),n(La_Marque)])
words list: [je,consomme,du,café,décaféiné,soluble,La_Marque]

keyword: []
tagged sentence:
sentence([pron(ils),v(prennent),det(du),n(lait),n(Nom_du_Produit)])
words list: [ils,prennent,du,lait,Nom_du_Produit]

keyword: []
tagged sentence:
sentence([pron(je),v(voudrais),v(savoir),conj(si),pron(je),v(peux),pron
(leur),v(donner),det(le),n(lait),n(transit),prep(à),det(la),n(place)])
words list:
[je,voudrais,savoir,si,je,peux,leur,donner,le,lait,transit,à,la,place]

keyword: []
tagged sentence:
sentence([prep(par),n(mesure),det(de),n(hygiène),pron(je),adv(ne),pron(
le),aux(ai),adv(pas),v(consommer)])
words list: [par,mesure,de,hygiène,je,ne,le,ai,pas,consommer]

keyword: []
tagged sentence:
sentence([pron(elle),v(attrape),det(des),n(hauts),det(le),n(cœur),adj(t
erribles),conj(et),v(rende),adv(aussitôt),det(le),n(peu),pron(que),pron
(elle),aux(a),v(avalé)])
words list:
[elle,attrape,des,hauts,le,cœur,terribles,et,rende,aussitôt,le,peu,que,
elle,a,avalé]
```

```
keyword: []
tagged sentence:
sentence([adj(fidèle),n(consommateur),det(de),pron(vos),n(produits),con
j(soit),det(les),n(Nom_du_Produit)])
words list:
[fidèle,consommateur,de,vos,produits,soit,les,Nom_du_Produit]

keyword: []
tagged sentence:
sentence([adj(fidèle),prep(depuis),det(des),n(années),det(de),det(votre
),n(Nom_du_Produit)])
words list: [fidèle,depuis,des,années,de,votre,Nom_du_Produit]

keyword: []
tagged sentence:
sentence([pron(elle),aux(a),adv(énormément),det(de),n(coliques)])
words list: [elle,a,énormément,de,coliques]

keyword: []
tagged sentence:
sentence([pron(je),v(ai),det(des),n(chats),adj(diabétiques)])
words list: [je,ai,des,chats,diabétiques]
```

B.8 Codes du programme noyau module3.pl

```
/* Module 3: Chunking */
/* Wannachai Kampeera */

date('20130104').

chunks_file_name('chunked_sentences.pl').
chunks_log_file('chunked_sentences.log').
module3_input_file('candidates.pl').

chunking :-
      module3_input_file(FileName),
      load_module3_input(FileName, Candidates),
      candidates(ListOfSentences)= Candidates,
      chunking(ListOfSentences, ChunkedSentences),
      write_chunks_to_file(ChunkedSentences),
      write_chunks_log(ChunkedSentences),
      !.

load_module3_input(FileName, SentencesList):-
      open(FileName, 'read', Stream),
      read(Stream, SentencesList),
      close(Stream).

chunking([], []).
chunking([[KeyWord, sentence(Sentence), WordsList]|RestSentences],
            [KeWoSeCh|RestChunkedSentences]):-
      grouping(Sentence, ChunkedSentence),
      KeWoSeCh = [KeyWord, WordsList, sentence(Sentence),
chunked_sentence(ChunkedSentence)],
      chunking(RestSentences, RestChunkedSentences).

grouping([], []).
grouping(Sentence, [Chunk|RestChunks]):-
      chunk(POSHead, Rule),
      append(Rule, RestWords, Sentence),
      Chunk =.. [POSHead, Rule],
      grouping(RestWords, RestChunks).

write_chunks_to_file(ChunkedSentences):-
      chunks_file_name(ChunksFileName),
      open(ChunksFileName, 'write', Stream),
      writeq(Stream, chunked_sentences(ChunkedSentences)),
      write(Stream, '.'),
      close(Stream).
```

```
write_chunks_log(ChunkedSentences):-
      chunks_log_file(FileName),
      open(FileName, 'write', Stream),
      write_chunks_log(ChunkedSentences, Stream),
      close(Stream).

write_chunks_log([], _Stream).
write_chunks_log([Candidate|RestCandidates], Stream):-
      Candidate = [KeyWord, WordList, TaggedSentence,
ChunkedSentences],
      write(Stream, 'keyword: '),
      write(Stream, KeyWord),
      write(Stream, '\n'),
      write(Stream, 'words list: '),
      write(Stream, WordList),
      write(Stream, '\n'),
      write(Stream, 'tagged sentence: '),
      write(Stream, TaggedSentence),
      write(Stream, '\n'),
      write(Stream, 'chunked sentence: '),
      write(Stream, ChunkedSentences),
      write(Stream, '\n'),
      chunked_sentence(ListOfChunks) = ChunkedSentences,
      write_chunk_per_line(Stream, ListOfChunks),
      write(Stream, '\n'),
      write(Stream, '\n'),
      write_chunks_log(RestCandidates, Stream).

write_chunk_per_line(_Stream, []).
write_chunk_per_line(Stream, [Chunk|RestChunks]):-
      write(Stream, '\t'),
      write(Stream, Chunk),
      write(Stream, '\n'),
      write_chunk_per_line(Stream, RestChunks).


/* expressions */
chunk(expression, [prep('à'),det('le'),n('moment'),prep('de'),v(_V)]).
chunk(expression, [v('faire'),n('part'),prep('de')]).
chunk(expression, [v('fais'),n('part'),prep('de')]).
chunk(expression, [adj('quel'),adv('ne'),v('est'),adv('pas')]).
chunk(gprep, [prep('vis-à-vis'),prep('de')]).
chunk(gprep, [adv('lors'),prep('de')]).
chunk(gprep, [prep('quant'),prep('à')]).
```

```
/* verb chunk */
chunk(gv, [aux(_), adv(_), v(_)]).
chunk(gv, [aux(_), v(_)]).
chunk(gv, [pron('me'), aux(_), v(_)]).
chunk(gv, [pron('te'), aux(_), v(_)]).
chunk(gv, [pron('se'), aux(_), v(_)]).
chunk(gv, [v(_)]).

/* noun chunk */
chunk(gn,[det(_), n(_), adj(_), prep(_), det(_), n(_)]).
chunk(gn,[det(_), adj(_), adj(_), n(_), adj(_)]).
chunk(gn,[det(_), n(_), prep(_), det(_), n(_)]).
chunk(gn,[det(_), n(_), prep(_), n(_), n(_)]).
chunk(gn,[det(_), n(_), adj(_), adj, n(_)]).
chunk(gn,[det(_), n(_), adv(_), adj(_)]).
chunk(gn,[det(_), n(_), adj(_), n(_)]).
chunk(gn,[det(_), adj(_), n(_)]).
chunk(gn,[det(_), n(_), n(_)]).
chunk(gn,[det(_), n(_), adj(_)]).
chunk(gn,[det(_), n(_)]).
chunk(gn,[adj(_), n(_)]).
chunk(gn,[pron(_)]).
chunk(gn,[n(_)]).

/* adjective chunk */
chunk(gadj, [adv(_), adj(_)]).
chunk(gadj, [adj(_)]).

/* adverb */
chunk(gadv, [adv(_)]).

/* preposition */
chunk(gprep, [prep(_)]).

/* End of Programme */
```

B.9 Règles de chunking (grammaires de groupes de mots)

```
/* expressions */
chunk(expression, [prep('à'),det('le'),n('moment'),prep('de'),v(_V)]).
chunk(expression, [v('faire'),n('part'),prep('de')]).
chunk(expression, [v('fais'),n('part'),prep('de')]).
chunk(expression, [adj('quel'),adv('ne'),v('est'),adv('pas')]).
chunk(gprep, [prep('vis-à-vis'),prep('de')]).
chunk(gprep, [adv('lors'),prep('de')]).
chunk(gprep, [prep('quant'),prep('à')]).

/* verb chunk */
chunk(gv, [aux(_), adv(_), v(_)]).
chunk(gv, [aux(_), v(_)]).
chunk(gv, [pron('me'), aux(_), v(_)]).
chunk(gv, [pron('te'), aux(_), v(_)]).
chunk(gv, [pron('se'), aux(_), v(_)]).
chunk(gv, [v(_)]).

/* noun chunk */
chunk(gn,[det(_), n(_), adj(_), prep(_), det(_), n(_)]).
chunk(gn,[det(_), adj(_), adj(_), n(_), adj(_)]).
chunk(gn,[det(_), n(_), prep(_), det(_), n(_)]).
chunk(gn,[det(_), n(_), prep(_), n(_), n(_)]).
chunk(gn,[det(_), n(_), adj(_), adj, n(_)]).
chunk(gn,[det(_), n(_), adv(_), adj(_)]).
chunk(gn,[det(_), n(_), adj(_), n(_)]).
chunk(gn,[det(_), adj(_), n(_)]).
chunk(gn,[det(_), n(_), n(_)]).
chunk(gn,[det(_), n(_), adj(_)]).
chunk(gn,[det(_), n(_)]).
chunk(gn,[adj(_), n(_)]).
chunk(gn,[pron(_)]).
chunk(gn,[n(_)]).

/* adjective chunk */
chunk(gadj, [adv(_), adj(_)]).
chunk(gadj, [adj(_)]).

/* adverb */
chunk(gadv, [adv(_)]).

/* preposition */
chunk(gprep, [prep(_)]).

/* End of Programme */
```

B.10 Données de sortie du module 3 chunked_sentences.pl

chunked_sentences([['déçoit',[cela,me,'déçoit'],sentence([pron(cela),pr
on(me),v('déçoit')]),chunked_sentence([gn([pron(cela)]),gn([pron(me)]),
gv([v('déçoit')])])],['déçu',['celle-
ci',me,a,beaucoup,'déçu'],sentence([pron('celle-
ci'),pron(me),aux(a),adv(beaucoup),v('déçu')]),chunked_sentence([gn([pr
on('celle-
ci')]),gn([pron(me)]),gv([aux(a),adv(beaucoup),v('déçu')])])],['déçoit'
,['La_Marque',me,'déçoit'],sentence([n('La_Marque'),pron(me),v('déçoit'
)]),chunked_sentence([gn([n('La_Marque')]),gn([pron(me)]),gv([v('déçoit
')])])],['décevant',[je,trouve,ça,'très','décevant'],sentence([pron(je)
,v(trouve),pron(ça),adv('très'),adj('décevant')]),chunked_sentence([gn(
[pron(je)]),gv([v(trouve)]),gn([pron(ça)]),gadj([adv('très'),adj('décev
ant')])])],['déception',[à,ma,grande,'déception'],sentence([prep(à),det
(ma),adj(grande),n('déception')]),chunked_sentence([gprep([prep(à)]),gn
([det(ma),adj(grande),n('déception')])])],['déception',[ce,courrier,pou
r,vous,faire,part,de,ma,'déception'],sentence([det(ce),n(courrier),prep
(pour),pron(vous),v(faire),n(part),prep(de),det(ma),n('déception')]),ch
unked_sentence([gn([det(ce),n(courrier)]),gprep([prep(pour)]),gn([pron(
vous)]),expression([v(faire),n(part),prep(de)]),gn([det(ma),n('déceptio
n')])])],['déception',[quelle,'déception','même',pas,une,'décoration',s
ur,les,'bûches'],sentence([adj(quelle),n('déception'),adj('même'),adv(p
as),det(une),n('décoration'),prep(sur),det(les),n('bûches')]),chunked_s
entence([gn([adj(quelle),n('déception')]),gadj([adj('même')]),gadv([adv
(pas)]),gn([det(une),n('décoration'),prep(sur),det(les),n('bûches')])])
],['déception',[je,vous,fais,part,de,ma,'déception'],sentence([pron(je)
,pron(vous),v(fais),n(part),prep(de),det(ma),n('déception')]),chunked_s
entence([gn([pron(je)]),gn([pron(vous)]),expression([v(fais),n(part),pr
ep(de)]),gn([det(ma),n('déception')])])],['déception',[je,vous,fais,par
t,de,ma,'déception','vis-à-
vis',de,le,'Nom_du_Produit'],sentence([pron(je),pron(vous),v(fais),n(pa
rt),prep(de),det(ma),n('déception'),prep('vis-à-
vis'),prep(de),det(le),n('Nom_du_Produit')]),chunked_sentence([gn([pron
(je)]),gn([pron(vous)]),expression([v(fais),n(part),prep(de)]),gn([det(
ma),n('déception')]),gprep([prep('vis-à-
vis'),prep(de)]),gn([det(le),n('Nom_du_Produit')])])],['déception',[la,
'déception',de,un,enfant],sentence([det(la),n('déception'),prep(de),det
(un),n(enfant)]),chunked_sentence([gn([det(la),n('déception'),prep(de),
det(un),n(enfant)])])],['déception',[quel,ne,est,pas,ma,'déception'],se
ntence([adj(quel),adv(ne),v(est),adv(pas),det(ma),n('déception')]),chun
ked_sentence([expression([adj(quel),adv(ne),v(est),adv(pas)]),gn([det(m
a),n('déception')])])],['déçu',[je,ai,'été','déçu',lors,de,mon,dernier,
achat],sentence([pron(je),aux(ai),v('été'),adj('déçu'),adv(lors),prep(d
e),det(mon),adj(dernier),n(achat)]),chunked_sentence([gn([pron(je)]),gv
([aux(ai),v('été')]),gadj([adj('déçu')]),gprep([adv(lors),prep(de)]),gn
([det(mon),adj(dernier),n(achat)])])],['déçue',[je,ai,'été','déçue',par

,le,'côté',âpre,de,cet,article],sentence([pron(je),aux(ai),v('été'),adj
('déçue'),prep(par),det(le),n('côté'),adj(âpre),prep(de),det(cet),n(art
icle)]),chunked_sentence([gn([pron(je)]),gv([aux(ai),v('été')]),gadj([a
dj('déçue')]),gprep([prep(par)]),gn([det(le),n('côté'),adj(âpre),prep(d
e),det(cet),n(article)])])],['déçue',[je,me,suis,'trouvée','très','déçu
e',de,ce,produit],sentence([pron(je),pron(me),aux(suis),v('trouvée'),ad
v('très'),adj('déçue'),prep(de),det(ce),n(produit)]),chunked_sentence([
gn([pron(je)]),gv([pron(me),aux(suis),v('trouvée')]),gadj([adv('très'),
adj('déçue')]),gprep([prep(de)]),gn([det(ce),n(produit)])])],['déçue',[
je,suis,'amèrement','déçue'],sentence([pron(je),v(suis),adv('amèrement'
),adj('déçue')]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([a
dv('amèrement'),adj('déçue')])])],['déçue',[je,suis,'déçue',de,leur,rem
arque],sentence([pron(je),v(suis),adj('déçue'),prep(de),det(leur),n(rem
arque)]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçu
e')]),gprep([prep(de)]),gn([det(leur),n(remarque)])])],['déçu',[je,suis
,'déçu',de,mes,deux,derniers,achats,identiques],sentence([pron(je),v(su
is),adj('déçu'),prep(de),det(mes),adj(deux),adj(derniers),n(achats),adj
(identiques)]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj
('déçu')]),gprep([prep(de)]),gn([det(mes),adj(deux),adj(derniers),n(ach
ats),adj(identiques)])])],['déçue',[je,suis,'déçue',de,votre,marque,'La
_Marque'],sentence([pron(je),v(suis),adj('déçue'),prep(de),det(votre),n
(marque),n('La_Marque')]),chunked_sentence([gn([pron(je)]),gv([v(suis)]
),gadj([adj('déçue')]),gprep([prep(de)]),gn([det(votre),n(marque),n('La
_Marque')])])],['déçue',[je,suis,'déçue',par,une,tablette,de,chocolat,'
Nom_du_Produit'],sentence([pron(je),v(suis),adj('déçue'),prep(par),det(
une),n(tablette),prep(de),n(chocolat),n('Nom_du_Produit')]),chunked_sen
tence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gprep([prep(pa
r)]),gn([det(une),n(tablette),prep(de),n(chocolat),n('Nom_du_Produit')]
)])],['déçue',[je,suis,'déçue',pour,la,'première',fois,depuis,'25',ans]
,sentence([pron(je),v(suis),adj('déçue'),prep(pour),det(la),adj('premiè
re'),n(fois),prep(depuis),adj('25'),n(ans)]),chunked_sentence([gn([pron
(je)]),gv([v(suis)]),gadj([adj('déçue')]),gprep([prep(pour)]),gn([det(l
a),adj('première'),n(fois)]),gprep([prep(depuis)]),gn([adj('25'),n(ans)
])])],['déçue',[je,suis,donc,'très','déçue',quant,à,la,éthique,de,votre
,'société'],sentence([pron(je),v(suis),adv(donc),adv('très'),adj('déçue
'),prep(quant),prep(à),det(la),n(éthique),prep(de),det(votre),n('société
')]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadv([adv(donc)]),
gadj([adv('très'),adj('déçue')]),gprep([prep(quant),prep(à)]),gn([det(l
a),n(éthique),prep(de),det(votre),n('société')])])],['déçu',[je,suis,'t
rès','déçu',de,les,truffes,noir,'70','%'],sentence([pron(je),v(suis),ad
v('très'),adj('déçu'),prep(de),det(les),n(truffes),adj(noir),adj('70'),
n('%')]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('très
'),adj('déçu')]),gprep([prep(de)]),gn([det(les),n(truffes),adj(noir)]),
gn([adj('70'),n('%')])])],['déçue',[je,suis,'très','déçue',de,ce,produi
t],sentence([pron(je),v(suis),adv('très'),adj('déçue'),prep(de),det(ce)

```
,n(produit)]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv(
'très'),adj('déçue')]),gprep([prep(de)]),gn([det(ce),n(produit)])])],['
déçu',[je,suis,'très','déçu',par,le,menu],sentence([pron(je),v(suis),ad
v('très'),adj('déçu'),prep(par),det(le),n(menu)]),chunked_sentence([gn(
[pron(je)]),gv([v(suis)]),gadj([adv('très'),adj('déçu')]),gprep([prep(p
ar)]),gn([det(le),n(menu)])])],['déçue',[je,suis,une,consommatrice,'très
','déçue',de,votre,grande,marque],sentence([pron(je),v(suis),det(une),
n(consommatrice),adv('très'),adj('déçue'),prep(de),det(votre),adj(grand
e),n(marque)]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gn([det(u
ne),n(consommatrice),adv('très'),adj('déçue')]),gprep([prep(de)]),gn([d
et(votre),adj(grande),n(marque)])])],['déçu',[je,suis,vraiment,'déçu',d
e,vos,pizzas,'surgelées','Nom_du_Produit'],sentence([pron(je),v(suis),a
dv(vraiment),adj('déçu'),prep(de),det(vos),n(pizzas),adj('surgelées'),n
('Nom_du_Produit')]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gad
j([adv(vraiment),adj('déçu')]),gprep([prep(de)]),gn([det(vos),n(pizzas)
,adj('surgelées'),n('Nom_du_Produit')])])],['déçue',[je,suis,'déçue'],s
entence([pron(je),v(suis),adj('déçue')]),chunked_sentence([gn([pron(je)
]),gv([v(suis)]),gadj([adj('déçue')])])],['déçue',[je,suis,'très','déçu
e'],sentence([pron(je),v(suis),adv('très'),adj('déçue')]),chunked_sente
nce([gn([pron(je)]),gv([v(suis)]),gadj([adv('très'),adj('déçue')])])],[
'déçu',[je,ai,'été','très','déçu',ce,jour],sentence([pron(je),aux(ai),v
('été'),adv('très'),adj('déçu'),det(ce),n(jour)]),chunked_sentence([gn(
[pron(je)]),gv([aux(ai),v('été')]),gadj([adv('très'),adj('déçu')]),gn([
det(ce),n(jour)])])],['déçue',[je,en,ai,'été','déçue'],sentence([pron(j
e),pron(en),aux(ai),v('été'),adj('déçue')]),chunked_sentence([gn([pron(
je)]),gn([pron(en)]),gv([aux(ai),v('été')]),gadj([adj('déçue')])])],['d
éçue',[je,suis,'particulièrement','déçue',de,mon,dernier,achat],sentenc
e([pron(je),v(suis),adv('particulièrement'),adj('déçue'),prep(de),det(m
on),adj(dernier),n(achat)]),chunked_sentence([gn([pron(je)]),gv([v(suis
)]),gadj([adv('particulièrement'),adj('déçue')]),gprep([prep(de)]),gn([
det(mon),adj(dernier),n(achat)])])],['déçus',[nous,avons,'été','déçus']
,sentence([pron(nous),aux(avons),v('été'),adj('déçus')]),chunked_senten
ce([gn([pron(nous)]),gv([aux(avons),v('été')]),gadj([adj('déçus')])])]]
).
```

B.11 Données de sortie du module 3 chunked_sentences.log

```
keyword: déçoit
words list: [cela,me,déçoit]
tagged sentence: sentence([pron(cela),pron(me),v(déçoit)])
chunked sentence:
chunked_sentence([gn([pron(cela)]),gn([pron(me)]),gv([v(déçoit)])])
      gn([pron(cela)])
      gn([pron(me)])
      gv([v(déçoit)])

keyword: déçu
words list: [celle-ci,me,a,beaucoup,déçu]
tagged sentence: sentence([pron(celle-
ci),pron(me),aux(a),adv(beaucoup),v(déçu)])
chunked sentence: chunked_sentence([gn([pron(celle-
ci)]),gn([pron(me)]),gv([aux(a),adv(beaucoup),v(déçu)])])
      gn([pron(celle-ci)])
      gn([pron(me)])
      gv([aux(a),adv(beaucoup),v(déçu)])

keyword: déçoit
words list: [La_Marque,me,déçoit]
tagged sentence: sentence([n(La_Marque),pron(me),v(déçoit)])
chunked sentence:
chunked_sentence([gn([n(La_Marque)]),gn([pron(me)]),gv([v(déçoit)])])
      gn([n(La_Marque)])
      gn([pron(me)])
      gv([v(déçoit)])

keyword: décevant
words list: [je,trouve,ça,très,décevant]
tagged sentence:
sentence([pron(je),v(trouve),pron(ça),adv(très),adj(décevant)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(trouve)]),gn([pron(ça)]),gadj([a
dv(très),adj(décevant)])])
      gn([pron(je)])
      gv([v(trouve)])
      gn([pron(ça)])
      gadj([adv(très),adj(décevant)])

keyword: déception
words list: [à,ma,grande,déception]
tagged sentence: sentence([prep(à),det(ma),adj(grande),n(déception)])
```

```
chunked sentence:
chunked_sentence([gprep([prep(à)]),gn([det(ma),adj(grande),n(déception)
])])
      gprep([prep(à)])
      gn([det(ma),adj(grande),n(déception)])

keyword: déception
words list: [ce,courrier,pour,vous,faire,part,de,ma,déception]
tagged sentence:
sentence([det(ce),n(courrier),prep(pour),pron(vous),v(faire),n(part),pr
ep(de),det(ma),n(déception)])
chunked sentence:
chunked_sentence([gn([det(ce),n(courrier)]),gprep([prep(pour)]),gn([pro
n(vous)]),expression([v(faire),n(part),prep(de)]),gn([det(ma),n(décepti
on)])])
      gn([det(ce),n(courrier)])
      gprep([prep(pour)])
      gn([pron(vous)])
      expression([v(faire),n(part),prep(de)])
      gn([det(ma),n(déception)])

keyword: déception
words list: [quelle,déception,même,pas,une,décoration,sur,les,bûches]
tagged sentence:
sentence([adj(quelle),n(déception),adj(même),adv(pas),det(une),n(décora
tion),prep(sur),det(les),n(bûches)])
chunked sentence:
chunked_sentence([gn([adj(quelle),n(déception)]),gadj([adj(même)]),gadv
([adv(pas)]),gn([det(une),n(décoration),prep(sur),det(les),n(bûches)])]
)
      gn([adj(quelle),n(déception)])
      gadj([adj(même)])
      gadv([adv(pas)])
      gn([det(une),n(décoration),prep(sur),det(les),n(bûches)])

keyword: déception
words list: [je,vous,fais,part,de,ma,déception]
tagged sentence:
sentence([pron(je),pron(vous),v(fais),n(part),prep(de),det(ma),n(décept
ion)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gn([pron(vous)]),expression([v(fais),n
(part),prep(de)]),gn([det(ma),n(déception)])])
      gn([pron(je)])
      gn([pron(vous)])
```

```
      expression([v(fais),n(part),prep(de)])
      gn([det(ma),n(déception)])

keyword: déception
words list: [je,vous,fais,part,de,ma,déception,vis-à-
vis,de,le,Nom_du_Produit]
tagged sentence:
sentence([pron(je),pron(vous),v(fais),n(part),prep(de),det(ma),n(décept
ion),prep(vis-à-vis),prep(de),det(le),n(Nom_du_Produit)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gn([pron(vous)]),expression([v(fais),n
(part),prep(de)]),gn([det(ma),n(déception)]),gprep([prep(vis-à-
vis),prep(de)]),gn([det(le),n(Nom_du_Produit)])])
      gn([pron(je)])
      gn([pron(vous)])
      expression([v(fais),n(part),prep(de)])
      gn([det(ma),n(déception)])
      gprep([prep(vis-à-vis),prep(de)])
      gn([det(le),n(Nom_du_Produit)])

keyword: déception
words list: [la,déception,de,un,enfant]
tagged sentence:
sentence([det(la),n(déception),prep(de),det(un),n(enfant)])
chunked sentence:
chunked_sentence([gn([det(la),n(déception),prep(de),det(un),n(enfant)])
])
      gn([det(la),n(déception),prep(de),det(un),n(enfant)])

keyword: déception
words list: [quel,ne,est,pas,ma,déception]
tagged sentence:
sentence([adj(quel),adv(ne),v(est),adv(pas),det(ma),n(déception)])
chunked sentence:
chunked_sentence([expression([adj(quel),adv(ne),v(est),adv(pas)]),gn([d
et(ma),n(déception)])])
      expression([adj(quel),adv(ne),v(est),adv(pas)])
      gn([det(ma),n(déception)])

keyword: déçu
words list: [je,ai,été,déçu,lors,de,mon,dernier,achat]
tagged sentence:
sentence([pron(je),aux(ai),v(été),adj(déçu),adv(lors),prep(de),det(mon)
,adj(dernier),n(achat)])
```

```
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([aux(ai),v(été)]),gadj([adj(déçu)])
,gprep([adv(lors),prep(de)]),gn([det(mon),adj(dernier),n(achat)])])
      gn([pron(je)])
      gv([aux(ai),v(été)])
      gadj([adj(déçu)])
      gprep([adv(lors),prep(de)])
      gn([det(mon),adj(dernier),n(achat)])

keyword: déçue
words list: [je,ai,été,déçue,par,le,côté,âpre,de,cet,article]
tagged sentence:
sentence([pron(je),aux(ai),v(été),adj(déçue),prep(par),det(le),n(côté),
adj(âpre),prep(de),det(cet),n(article)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([aux(ai),v(été)]),gadj([adj(déçue)]
),gprep([prep(par)]),gn([det(le),n(côté),adj(âpre),prep(de),det(cet),n(
article)])])
      gn([pron(je)])
      gv([aux(ai),v(été)])
      gadj([adj(déçue)])
      gprep([prep(par)])
      gn([det(le),n(côté),adj(âpre),prep(de),det(cet),n(article)])

keyword: déçue
words list: [je,me,suis,trouvée,très,déçue,de,ce,produit]
tagged sentence:
sentence([pron(je),pron(me),aux(suis),v(trouvée),adv(très),adj(déçue),p
rep(de),det(ce),n(produit)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([pron(me),aux(suis),v(trouvée)]),ga
dj([adv(très),adj(déçue)]),gprep([prep(de)]),gn([det(ce),n(produit)])])
      gn([pron(je)])
      gv([pron(me),aux(suis),v(trouvée)])
      gadj([adv(très),adj(déçue)])
      gprep([prep(de)])
      gn([det(ce),n(produit)])

keyword: déçue
words list: [je,suis,amèrement,déçue]
tagged sentence: sentence([pron(je),v(suis),adv(amèrement),adj(déçue)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv(amèrement),adj
(déçue)])])
      gn([pron(je)])
```

```
      gv([v(suis)])
      gadj([adv(amèrement),adj(déçue)])

keyword: déçue
words list: [je,suis,déçue,de,leur,remarque]
tagged sentence:
sentence([pron(je),v(suis),adj(déçue),prep(de),det(leur),n(remarque)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj(déçue)]),gprep
([prep(de)]),gn([det(leur),n(remarque)])])
      gn([pron(je)])
      gv([v(suis)])
      gadj([adj(déçue)])
      gprep([prep(de)])
      gn([det(leur),n(remarque)])

keyword: déçu
words list: [je,suis,déçu,de,mes,deux,derniers,achats,identiques]
tagged sentence:
sentence([pron(je),v(suis),adj(déçu),prep(de),det(mes),adj(deux),adj(de
rniers),n(achats),adj(identiques)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj(déçu)]),gprep(
[prep(de)]),gn([det(mes),adj(deux),adj(derniers),n(achats),adj(identiqu
es)])])
      gn([pron(je)])
      gv([v(suis)])
      gadj([adj(déçu)])
      gprep([prep(de)])
      gn([det(mes),adj(deux),adj(derniers),n(achats),adj(identiques)])

keyword: déçue
words list: [je,suis,déçue,de,votre,marque,La_Marque]
tagged sentence:
sentence([pron(je),v(suis),adj(déçue),prep(de),det(votre),n(marque),n(L
a_Marque)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj(déçue)]),gprep
([prep(de)]),gn([det(votre),n(marque),n(La_Marque)])])
      gn([pron(je)])
      gv([v(suis)])
      gadj([adj(déçue)])
      gprep([prep(de)])
      gn([det(votre),n(marque),n(La_Marque)])
```

```
keyword: déçue
words list: [je,suis,déçue,par,une,tablette,de,chocolat,Nom_du_Produit]
tagged sentence:
sentence([pron(je),v(suis),adj(déçue),prep(par),det(une),n(tablette),pr
ep(de),n(chocolat),n(Nom_du_Produit)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj(déçue)]),gprep
([prep(par)]),gn([det(une),n(tablette),prep(de),n(chocolat),n(Nom_du_Pr
oduit)])])
      gn([pron(je)])
      gv([v(suis)])
      gadj([adj(déçue)])
      gprep([prep(par)])
      gn([det(une),n(tablette),prep(de),n(chocolat),n(Nom_du_Produit)])

keyword: déçue
words list: [je,suis,déçue,pour,la,première,fois,depuis,25,ans]
tagged sentence:
sentence([pron(je),v(suis),adj(déçue),prep(pour),det(la),adj(première),
n(fois),prep(depuis),adj(25),n(ans)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj(déçue)]),gprep
([prep(pour)]),gn([det(la),adj(première),n(fois)]),gprep([prep(depuis)]
),gn([adj(25),n(ans)])])
      gn([pron(je)])
      gv([v(suis)])
      gadj([adj(déçue)])
      gprep([prep(pour)])
      gn([det(la),adj(première),n(fois)])
      gprep([prep(depuis)])
      gn([adj(25),n(ans)])

keyword: déçue
words list:
[je,suis,donc,très,déçue,quant,à,la,éthique,de,votre,société]
tagged sentence:
sentence([pron(je),v(suis),adv(donc),adv(très),adj(déçue),prep(quant),p
rep(à),det(la),n(éthique),prep(de),det(votre),n(société)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadv([adv(donc)]),gadj([
adv(très),adj(déçue)]),gprep([prep(quant),prep(à)]),gn([det(la),n(éthiq
ue),prep(de),det(votre),n(société)])])
      gn([pron(je)])
      gv([v(suis)])
```

```
      gadv([adv(donc)])
      gadj([adv(très),adj(déçue)])
      gprep([prep(quant),prep(à)])
      gn([det(la),n(éthique),prep(de),det(votre),n(société)])

keyword: déçu
words list: [je,suis,très,déçu,de,les,truffes,noir,70,%]
tagged sentence:
sentence([pron(je),v(suis),adv(très),adj(déçu),prep(de),det(les),n(truf
fes),adj(noir),adj(70),n(%)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv(très),adj(déçu
)]),gprep([prep(de)]),gn([det(les),n(truffes),adj(noir)]),gn([adj(70),n
(%)])])
      gn([pron(je)])
      gv([v(suis)])
      gadj([adv(très),adj(déçu)])
      gprep([prep(de)])
      gn([det(les),n(truffes),adj(noir)])
      gn([adj(70),n(%)])

keyword: déçue
words list: [je,suis,très,déçue,de,ce,produit]
tagged sentence:
sentence([pron(je),v(suis),adv(très),adj(déçue),prep(de),det(ce),n(prod
uit)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv(très),adj(déçu
e)]),gprep([prep(de)]),gn([det(ce),n(produit)])])
      gn([pron(je)])
      gv([v(suis)])
      gadj([adv(très),adj(déçue)])
      gprep([prep(de)])
      gn([det(ce),n(produit)])

keyword: déçu
words list: [je,suis,très,déçu,par,le,menu]
tagged sentence:
sentence([pron(je),v(suis),adv(très),adj(déçu),prep(par),det(le),n(menu
)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv(très),adj(déçu
)]),gprep([prep(par)]),gn([det(le),n(menu)])])
      gn([pron(je)])
      gv([v(suis)])
```

```
      gadj([adv(très),adj(déçu)])
      gprep([prep(par)])
      gn([det(le),n(menu)])

keyword: déçue
words list:
[je,suis,une,consommatrice,très,déçue,de,votre,grande,marque]
tagged sentence:
sentence([pron(je),v(suis),det(une),n(consommatrice),adv(très),adj(déçu
e),prep(de),det(votre),adj(grande),n(marque)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gn([det(une),n(consommat
rice),adv(très),adj(déçue)]),gprep([prep(de)]),gn([det(votre),adj(grand
e),n(marque)])])
      gn([pron(je)])
      gv([v(suis)])
      gn([det(une),n(consommatrice),adv(très),adj(déçue)])
      gprep([prep(de)])
      gn([det(votre),adj(grande),n(marque)])

keyword: déçu
words list:
[je,suis,vraiment,déçu,de,vos,pizzas,surgelées,Nom_du_Produit]
tagged sentence:
sentence([pron(je),v(suis),adv(vraiment),adj(déçu),prep(de),det(vos),n(
pizzas),adj(surgelées),n(Nom_du_Produit)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv(vraiment),adj(
déçu)]),gprep([prep(de)]),gn([det(vos),n(pizzas),adj(surgelées),n(Nom_d
u_Produit)])])
      gn([pron(je)])
      gv([v(suis)])
      gadj([adv(vraiment),adj(déçu)])
      gprep([prep(de)])
      gn([det(vos),n(pizzas),adj(surgelées),n(Nom_du_Produit)])

keyword: déçue
words list: [je,suis,déçue]
tagged sentence: sentence([pron(je),v(suis),adj(déçue)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj(déçue)])])
      gn([pron(je)])
      gv([v(suis)])
      gadj([adj(déçue)])
```

```
keyword: déçue
words list: [je,suis,très,déçue]
tagged sentence: sentence([pron(je),v(suis),adv(très),adj(déçue)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv(très),adj(déçu
e)])])
      gn([pron(je)])
      gv([v(suis)])
      gadj([adv(très),adj(déçue)])

keyword: déçu
words list: [je,ai,été,très,déçu,ce,jour]
tagged sentence:
sentence([pron(je),aux(ai),v(été),adv(très),adj(déçu),det(ce),n(jour)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([aux(ai),v(été)]),gadj([adv(très),a
dj(déçu)]),gn([det(ce),n(jour)])])
      gn([pron(je)])
      gv([aux(ai),v(été)])
      gadj([adv(très),adj(déçu)])
      gn([det(ce),n(jour)])

keyword: déçue
words list: [je,en,ai,été,déçue]
tagged sentence:
sentence([pron(je),pron(en),aux(ai),v(été),adj(déçue)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gn([pron(en)]),gv([aux(ai),v(été)]),ga
dj([adj(déçue)])])
      gn([pron(je)])
      gn([pron(en)])
      gv([aux(ai),v(été)])
      gadj([adj(déçue)])

keyword: déçue
words list: [je,suis,particulièrement,déçue,de,mon,dernier,achat]
tagged sentence:
sentence([pron(je),v(suis),adv(particulièrement),adj(déçue),prep(de),de
t(mon),adj(dernier),n(achat)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv(particulièreme
nt),adj(déçue)]),gprep([prep(de)]),gn([det(mon),adj(dernier),n(achat)])
])
      gn([pron(je)])
```

```
      gv([v(suis)])
      gadj([adv(particulièrement),adj(déçue)])
      gprep([prep(de)])
      gn([det(mon),adj(dernier),n(achat)])

keyword: déçus
words list: [nous,avons,été,déçus]
tagged sentence: sentence([pron(nous),aux(avons),v(été),adj(déçus)])
chunked sentence:
chunked_sentence([gn([pron(nous)]),gv([aux(avons),v(été)]),gadj([adj(dé
çus)])])
      gn([pron(nous)])
      gv([aux(avons),v(été)])
      gadj([adj(déçus)])
```

B.12 Codes du programme noyau module4.pl

```
/* module 4: chunks clean-up */
/* Wannachai Kampeera */

date('20130105').

:- ensure_loaded('flechie_canonical.pl').

chunks_cleaned_file_name('chunked_sentences_cleaned.pl').
chunks_cleaned_log_file('chunked_sentences_cleaned.log').
module4_input_file('chunked_sentences.pl').

clean_up :-
      module4_input_file(FileName),
      load_module4_input(FileName, ChunkedSentences),
      chunked_sentences(ListOfChunkedSentences) = ChunkedSentences,
      clean_up(ListOfChunkedSentences, CleanChunks),
      write_clean_chunks_to_file(CleanChunks),
      write_clean_chunks_log(CleanChunks),
      !.

load_module4_input(FileName, SentencesList):-
      open(FileName, 'read', Stream),
      read(Stream, SentencesList),
      close(Stream).

clean_up([], []).
clean_up([ChunkedSentence|RestChunkedSentences],
[KeWoSeChCl|Rest_KeWoSeChCl]):-
      ChunkedSentence = [KeyWord, WordsList, sentence(Sentence),
chunked_sentence(ListOfChunks)],
      transforms(ListOfChunks, CleanChunks),
      KeWoSeChCl = [KeyWord, WordsList,
                  sentence(Sentence),
                  chunked_sentence(ListOfChunks),
                  clean_chunked_sentence(CleanChunks)],
      clean_up(RestChunkedSentences, Rest_KeWoSeChCl).

transforms([], []).
transforms([Chunk|RestChunks], [NewChunk|RestNewChunks]):-
      transform_to(Chunk, TransformedChunk),
      (
          TransformedChunk =.. [POS, ListOfWords],
          inflected_to_canonical(ListOfWords, CanonicalForm),
          NewChunk =.. [POS, CanonicalForm],
          !
```

```
      ;
          \+ TransformedChunk =.. [POS, ListOfWords], % in case of list
of decomposition
          TransformedChunk = [FirstWord, SecondWord],
          FirstWord =.. [POS, ListOfWords],
          SecondWord =.. [POS2, ListOfWords2],
          inflected_to_canonical(ListOfWords, CanonicalForm),
          inflected_to_canonical(ListOfWords2, CanonicalForm2),
          Word1 =.. [POS, CanonicalForm],
          Word2 =.. [POS2, CanonicalForm2],
          NewChunk = [Word1, Word2]
      ),
      transforms(RestChunks, RestNewChunks).
transforms([Chunk|RestChunks], [NewChunk|RestNewChunks]):-
      \+ transform_to(Chunk, _NewChunk),
      Chunk =.. [POS, ListOfWords],
      inflected_to_canonical(ListOfWords, CanonicalForm),
      NewChunk =.. [POS, CanonicalForm],
      transforms(RestChunks, RestNewChunks).

inflected_to_canonical([], []).
inflected_to_canonical([InflectedWord|RestInflectedWords],
[CanonicalForm|RestCanonicalForms]):-
      flechie_canonique(InflectedWord, CanonicalForm),
      inflected_to_canonical(RestInflectedWords, RestCanonicalForms).
inflected_to_canonical([InflectedWord|RestInflectedWords],
[InflectedWord|RestCanonicalForms]):-
      \+ flechie_canonique(InflectedWord, _CanonicalForm),
      inflected_to_canonical(RestInflectedWords, RestCanonicalForms).

write_clean_chunks_to_file(CleanChunkedSentences):-
      chunks_cleaned_file_name(FileName),
      open(FileName, 'write', Stream),
      writeq(Stream, clean_chunked_sentences(CleanChunkedSentences)),
      write(Stream, '.'),
      close(Stream).

write_clean_chunks_log(ChunkedSentences):-
      chunks_cleaned_log_file(FileName),
      open(FileName, 'write', Stream),
      write_clean_chunks_log(ChunkedSentences, Stream),
      close(Stream).

write_clean_chunks_log([], _Stream).
write_clean_chunks_log([Sentence|RestSentences], Stream):-
```

```
      Sentence = [KeyWord, WordList, TaggedSentence, ChunkedSentences,
CleanChunks],
      write(Stream, 'keyword: '),
      writeq(Stream, KeyWord),
      write(Stream, '\n'),
      write(Stream, 'words list: '),
      writeq(Stream, WordList),
      write(Stream, '\n'),
      write(Stream, 'tagged sentence: '),
      writeq(Stream, TaggedSentence),
      write(Stream, '\n'),
      write(Stream, 'chunked sentence: '),
      writeq(Stream, ChunkedSentences),
      write(Stream, '\n'),
      write(Stream, 'chunked sentence cleaned: '),
      writeq(Stream, CleanChunks),
      write(Stream, '\n'),
      chunked_sentence(ListOfChunks) = ChunkedSentences,
      clean_chunked_sentence(ListOfCleanChunks) = CleanChunks,
      write_chunk_per_line(Stream, ListOfChunks, ListOfCleanChunks),
      write(Stream, '\n'),
      write(Stream, '\n'),
      write_clean_chunks_log(RestSentences, Stream).

write_chunk_per_line(_Stream, [], []).
write_chunk_per_line(Stream, [Chunk|RestChunks],
[CleanChunk|RestCleanChunks]):-
      write(Stream, '\t'),
      writeq(Stream, Chunk),
      write(Stream, ' ==> '),
      writeq(Stream, CleanChunk),
      write(Stream, '\n'),
      write_chunk_per_line(Stream, RestChunks, RestCleanChunks).

/* keep 1st and 2nd person pronouns */
transform_to(gn([det('mes'),adj(_Adj),adj(_Adj2),n(N),adj(_Adj3)]),
gn([det('mes'), n(N)])).
transform_to(gn([det('vos'),n(N1),adj(_Adj),n(N2)]), gn([det('vos'),
n(N1), n(N2)])).
transform_to(gn([det('votre'),n(N),n(N2)]), gn([det('votre'), n(N),
n(N2)])).
transform_to(gn([det('votre'),adj(_Adj),n(N)]), gn([det('votre'),
n(N)])).
transform_to(gn([det('ma'),adj(_),n(N)]), gn([det('ma'), n(N)])).
transform_to(gn([det('ma'),n(N)]), gn([det('ma'), n(N)])).
```

```
transform_to(gn([det('mon'),adj(_),n(N)]), gn([det('mon'), n(N)])).

/* decomposition, as a keyword is found in a chunk but is not the head
of this chunk */
transform_to(gn([det(_Det),n(N),adv(_Adv),adj('déçue')]), [gn([n(N)]),
gadj([adj('déçue')])]).

/* gn */
transform_to(gn([det(_Det1),n(N1),adj(_Adj),prep(Prep),det(_Det2),n(N2)
]), gn([n(N1),prep(Prep),n(N2)])).
transform_to(gn([det(_Det),n(N1),prep(Prep),n(N2),n(N3)]),
gn([n(N1),prep(Prep),n(N2),n(N3)])).
transform_to(gn([det(_),n(N1),prep(Prep),det(_Det),n(N2)]), gn([n(N1),
prep(Prep), n(N2)])).
transform_to(gn([det(_Det),n(N1),prep(Prep),det(_Det2),n(N2)]),
gn([n(N1),prep(Prep),n(N2)])).
transform_to(gn([det(_Det),n(N),adj(_Adj)]), gn([n(N)])).
transform_to(gn([det(_Det),adj(_Adj),n(N)]), gn([n(N)])).
transform_to(gn([adj(_Adj),n(N)]), gn([n(N)])).
transform_to(gn([det(_),n(N)]), gn([n(N)])).
transform_to(gn([det(_Det),n(N)]), gn([n(N)])).

/* gv */
transform_to(gv([pron(Pron),aux(_Aux),v(V)]), gv([pron_v(Pron),
v_pron(V)])).
transform_to(gv([aux(_Aux),adv(_Adv),v(V)]), gv([v(V)])).
transform_to(gv([aux(_Aux),v(V)]), gv([v(V)])).

/* gadj */
transform_to(gadj([adv(_Adv),adj(Adj)]), gadj([adj(Adj)])).

/* End of Programme */
```

B.13 Dictionnaire forme fléchie - forme canonique flechie_canonique.pl

```
flechie_canonique(adj('déçu'), adj('déçu')).
flechie_canonique(adj('déçue'), adj('déçu')).
flechie_canonique(adj('déçus'), adj('déçu')).
flechie_canonique(adj('décevant'), adj('décevant')).
flechie_canonique(adj('même'), adj('même')).
flechie_canonique(adj(quel), adj(quel)).
flechie_canonique(det(ma), det(ma)).
flechie_canonique(det(mes), det(mon)).
flechie_canonique(det(mon), det(mon)).
flechie_canonique(det(vos), det(votre)).
flechie_canonique(det(votre), det(votre)).
flechie_canonique(n(éthique), n(éthique)).
flechie_canonique(n('%'), n('pourcent')).
flechie_canonique(n('La_Marque'), n('La_Marque')).
flechie_canonique(n('Nom_du_Produit'), n('Nom_du_Produit')).
flechie_canonique(n('bûches'), n('bûche')).
flechie_canonique(n('côté'), n('côté')).
flechie_canonique(n('déception'), n('déception')).
flechie_canonique(n('décoration'), n('décoration')).
flechie_canonique(n('société'), n('société')).
flechie_canonique(n(achat), n(achat)).
flechie_canonique(n(achat), n(achat)).
flechie_canonique(n(achats), n(achat)).
flechie_canonique(n(ans), n(ans)).
flechie_canonique(n(article), n(article)).
flechie_canonique(n(chocolat), n(chocolat)).
flechie_canonique(n(consommatrice),n(consommatrice)).
flechie_canonique(n(courrier), n(courrier)).
flechie_canonique(n(enfant), n(enfant)).
flechie_canonique(n(fois), n(fois)).
flechie_canonique(n(jour), n(jour)).
flechie_canonique(n(marque), n(marque)).
flechie_canonique(n(marque), n(marque)).
flechie_canonique(n(menu), n(menu)).
flechie_canonique(n(part), n(part)).
flechie_canonique(n(pizzas), n(pizza)).
flechie_canonique(n(produit), n(produit)).
flechie_canonique(n(produit), n(produit)).
flechie_canonique(n(remarque), n(remarque)).
flechie_canonique(n(tablette), n(tablette)).
flechie_canonique(n(truffes), n(truffe)).
flechie_canonique(pron(ça), pron(ça)).
flechie_canonique(pron('celle-ci'), pron('celle-ci')).
flechie_canonique(pron(cela), pron(cela)).
flechie_canonique(pron(en), pron(en)).
```

```
flechie_canonique(pron(je), pron(je)).
flechie_canonique(pron(me), pron(me)).
flechie_canonique(pron(nous), pron(nous)).
flechie_canonique(pron(vous), pron(vous)).
flechie_canonique(v('été'), v('être')).
flechie_canonique(v(est), v('être')).
flechie_canonique(v('déçoit'), v('décevoir')).
flechie_canonique(v('déçu'), v('décevoir')).
flechie_canonique(v_pron('trouvée'), v_pron('trouver')).
flechie_canonique(v(est), v(être)).
flechie_canonique(v(faire), v(faire)).
flechie_canonique(v(fais), v(faire)).
flechie_canonique(v(suis), v(être)).
flechie_canonique(v(trouve), v(trouver)).
```

B.14 Règles de réduction/décomposition

```
/* keep 1st and 2nd person pronouns */
transform_to(gn([det('mes'),adj(_Adj),adj(_Adj2),n(N),adj(_Adj3)]),
gn([det('mes'), n(N)])).
transform_to(gn([det('vos'),n(N1),adj(_Adj),n(N2)]), gn([det('vos'),
n(N1), n(N2)])).
transform_to(gn([det('votre'),n(N),n(N2)]), gn([det('votre'), n(N),
n(N2)])).
transform_to(gn([det('votre'),adj(_Adj),n(N)]), gn([det('votre'),
n(N)])).
transform_to(gn([det('ma'),adj(_),n(N)]), gn([det('ma'), n(N)])).
transform_to(gn([det('ma'),n(N)]), gn([det('ma'), n(N)])).
transform_to(gn([det('mon'),adj(_),n(N)]), gn([det('mon'), n(N)])).

/* decomposition, as a keyword is found in a chunk but is not the head
of this chunk */
transform_to(gn([det(_Det),n(N),adv(_Adv),adj('déçue')]), [gn([n(N)]),
gadj([adj('déçue')])]).

/* gn */
transform_to(gn([det(_Det1),n(N1),adj(_Adj),prep(Prep),det(_Det2),n(N2)
]), gn([n(N1),prep(Prep),n(N2)])).
transform_to(gn([det(_Det),n(N1),prep(Prep),n(N2),n(N3)]),
gn([n(N1),prep(Prep),n(N2),n(N3)])).
transform_to(gn([det(_),n(N1),prep(Prep),det(_Det),n(N2)]), gn([n(N1),
prep(Prep), n(N2)])).
transform_to(gn([det(_Det),n(N1),prep(Prep),det(_Det2),n(N2)]),
gn([n(N1),prep(Prep),n(N2)])).
transform_to(gn([det(_Det),n(N),adj(_Adj)]), gn([n(N)])).
transform_to(gn([det(_Det),adj(_Adj),n(N)]), gn([n(N)])).
transform_to(gn([adj(_Adj),n(N)]), gn([n(N)])).
transform_to(gn([det(_),n(N)]), gn([n(N)])).
transform_to(gn([det(_Det),n(N)]), gn([n(N)])).

/* gv */
transform_to(gv([pron(Pron),aux(_Aux),v(V)]), gv([pron_v(Pron),
v_pron(V)])).
transform_to(gv([aux(_Aux),adv(_Adv),v(V)]), gv([v(V)])).
transform_to(gv([aux(_Aux),v(V)]), gv([v(V)])).

/* gadj */
transform_to(gadj([adv(_Adv),adj(Adj)]), gadj([adj(Adj)])).
```

B.15 Données de sortie du module 4 chunked_sentences_cleaned.pl

```
clean_chunked_sentences([['déçoit',[cela,me,'déçoit'],sentence([pron(ce
la),pron(me),v('déçoit')]),chunked_sentence([gn([pron(cela)]),gn([pron(
me)]),gv([v('déçoit')])]),clean_chunked_sentence([gn([pron(cela)]),gn([
pron(me)]),gv([v('décevoir')])])],['déçu',['celle-
ci',me,a,beaucoup,'déçu'],sentence([pron('celle-
ci'),pron(me),aux(a),adv(beaucoup),v('déçu')]),chunked_sentence([gn([pr
on('celle-
ci')]),gn([pron(me)]),gv([aux(a),adv(beaucoup),v('déçu')])]),clean_chun
ked_sentence([gn([pron('celle-
ci')]),gn([pron(me)]),gv([v('décevoir')])])],['déçoit',['La_Marque',me,
'déçoit'],sentence([n('La_Marque'),pron(me),v('déçoit')]),chunked_sente
nce([gn([n('La_Marque')]),gn([pron(me)]),gv([v('déçoit')])]),clean_chun
ked_sentence([gn([n('La_Marque')]),gn([pron(me)]),gv([v('décevoir')])])
],['décevant',[je,trouve,ça,'très','décevant'],sentence([pron(je),v(tro
uve),pron(ça),adv('très'),adj('décevant')]),chunked_sentence([gn([pron(
je)]),gv([v(trouve)]),gn([pron(ça)]),gadj([adv('très'),adj('décevant')]
)]),clean_chunked_sentence([gn([pron(je)]),gv([v(trouver)]),gn([pron(ça
)]),gadj([adj('décevant')])])],['déception',[à,ma,grande,'déception'],s
entence([prep(à),det(ma),adj(grande),n('déception')]),chunked_sentence(
[gprep([prep(à)]),gn([det(ma),adj(grande),n('déception')])]),clean_chun
ked_sentence([gprep([prep(à)]),gn([det(ma),n('déception')])])],['décept
ion',[ce,courrier,pour,vous,faire,part,de,ma,'déception'],sentence([det
(ce),n(courrier),prep(pour),pron(vous),v(faire),n(part),prep(de),det(ma
),n('déception')]),chunked_sentence([gn([det(ce),n(courrier)]),gprep([p
rep(pour)]),gn([pron(vous)]),expression([v(faire),n(part),prep(de)]),gn
([det(ma),n('déception')])]),clean_chunked_sentence([gn([n(courrier)]),
gprep([prep(pour)]),gn([pron(vous)]),expression([v(faire),n(part),prep(
de)]),gn([det(ma),n('déception')])])],['déception',[quelle,'déception',
'même',pas,une,'décoration',sur,les,'bûches'],sentence([adj(quelle),n('
déception'),adj('même'),adv(pas),det(une),n('décoration'),prep(sur),det
(les),n('bûches')]),chunked_sentence([gn([adj(quelle),n('déception')]),
gadj([adj('même')]),gadv([adv(pas)]),gn([det(une),n('décoration'),prep(
sur),det(les),n('bûches')])]),clean_chunked_sentence([gn([n('déception'
)]),gadj([adj('même')]),gadv([adv(pas)]),gn([n('décoration'),prep(sur),
n('bûche')])])],['déception',[je,vous,fais,part,de,ma,'déception'],sent
ence([pron(je),pron(vous),v(fais),n(part),prep(de),det(ma),n('déception
')]),chunked_sentence([gn([pron(je)]),gn([pron(vous)]),expression([v(fa
is),n(part),prep(de)]),gn([det(ma),n('déception')])]),clean_chunked_sen
tence([gn([pron(je)]),gn([pron(vous)]),expression([v(faire),n(part),pre
p(de)]),gn([det(ma),n('déception')])])],['déception',[je,vous,fais,part
,de,ma,'déception','vis-à-
vis',de,le,'Nom_du_Produit'],sentence([pron(je),pron(vous),v(fais),n(pa
rt),prep(de),det(ma),n('déception'),prep('vis-à-
vis'),prep(de),det(le),n('Nom_du_Produit')]),chunked_sentence([gn([pron
(je)]),gn([pron(vous)]),expression([v(fais),n(part),prep(de)]),gn([det(
```

ma),n('déception')]),gprep([prep('vis-à-
vis'),prep(de)]),gn([det(le),n('Nom_du_Produit')])]),clean_chunked_sent
ence([gn([pron(je)]),gn([pron(vous)]),expression([v(faire),n(part),prep
(de)]),gn([det(ma),n('déception')]),gprep([prep('vis-à-
vis'),prep(de)]),gn([n('Nom_du_Produit')])])],['déception',[la,'décepti
on',de,un,enfant],sentence([det(la),n('déception'),prep(de),det(un),n(e
nfant)]),chunked_sentence([gn([det(la),n('déception'),prep(de),det(un),
n(enfant)])]),clean_chunked_sentence([gn([n('déception'),prep(de),n(enf
ant)])])],['déception',[quel,ne,est,pas,ma,'déception'],sentence([adj(q
uel),adv(ne),v(est),adv(pas),det(ma),n('déception')]),chunked_sentence(
[expression([adj(quel),adv(ne),v(est),adv(pas)]),gn([det(ma),n('décepti
on')])]),clean_chunked_sentence([expression([adj(quel),adv(ne),v(être),
adv(pas)]),gn([det(ma),n('déception')])])],['déçu',[je,ai,'été','déçu',
lors,de,mon,dernier,achat],sentence([pron(je),aux(ai),v('été'),adj('déç
u'),adv(lors),prep(de),det(mon),adj(dernier),n(achat)]),chunked_sentenc
e([gn([pron(je)]),gv([aux(ai),v('été')]),gadj([adj('déçu')]),gprep([adv
(lors),prep(de)]),gn([det(mon),adj(dernier),n(achat)])]),clean_chunked_
sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([adv(l
ors),prep(de)]),gn([det(mon),n(achat)])])],['déçue',[je,ai,'été','déçue
',par,le,'côté',âpre,de,cet,article],sentence([pron(je),aux(ai),v('été'
),adj('déçue'),prep(par),det(le),n('côté'),adj(âpre),prep(de),det(cet),
n(article)]),chunked_sentence([gn([pron(je)]),gv([aux(ai),v('été')]),ga
dj([adj('déçue')]),gprep([prep(par)]),gn([det(le),n('côté'),adj(âpre),p
rep(de),det(cet),n(article)])]),clean_chunked_sentence([gn([pron(je)]),
gv([v(être)]),gadj([adj('déçu')]),gprep([prep(par)]),gn([n('côté'),prep
(de),n(article)])])],['déçue',[je,me,suis,'trouvée','très','déçue',de,c
e,produit],sentence([pron(je),pron(me),aux(suis),v('trouvée'),adv('très
'),adj('déçue'),prep(de),det(ce),n(produit)]),chunked_sentence([gn([pro
n(je)]),gv([pron(me),aux(suis),v('trouvée')]),gadj([adv('très'),adj('dé
çue')]),gprep([prep(de)]),gn([det(ce),n(produit)])]),clean_chunked_sent
ence([gn([pron(je)]),gv([pron_v(me),v_pron(trouver)]),gadj([adj('déçu')
]),gprep([prep(de)]),gn([n(produit)])])],['déçue',[je,suis,'amèrement',
'déçue'],sentence([pron(je),v(suis),adv('amèrement'),adj('déçue')]),chu
nked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('amèrement'),adj(
'déçue')])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj(
[adj('déçu')])])],['déçue',[je,suis,'déçue',de,leur,remarque],sentence(
[pron(je),v(suis),adj('déçue'),prep(de),det(leur),n(remarque)]),chunked
_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gprep([pre
p(de)]),gn([det(leur),n(remarque)])]),clean_chunked_sentence([gn([pron(
je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([prep(de)]),gn([n(remarqu
e)])])],['déçu',[je,suis,'déçu',de,mes,deux,derniers,achats,identiques]
,sentence([pron(je),v(suis),adj('déçu'),prep(de),det(mes),adj(deux),adj
(derniers),n(achats),adj(identiques)]),chunked_sentence([gn([pron(je)])
,gv([v(suis)]),gadj([adj('déçu')]),gprep([prep(de)]),gn([det(mes),adj(d
eux),adj(derniers),n(achats),adj(identiques)])]),clean_chunked_sentence

([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([prep(de)]),gn
([det(mon),n(achat)])])],['déçue',[je,suis,'déçue',de,votre,marque,'La_
Marque'],sentence([pron(je),v(suis),adj('déçue'),prep(de),det(votre),n(
marque),n('La_Marque')]),chunked_sentence([gn([pron(je)]),gv([v(suis)])
,gadj([adj('déçue')]),gprep([prep(de)]),gn([det(votre),n(marque),n('La_
Marque')])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj(
[adj('déçu')]),gprep([prep(de)]),gn([det(votre),n(marque),n('La_Marque'
)])])],['déçue',[je,suis,'déçue',par,une,tablette,de,chocolat,'Nom_du_P
roduit'],sentence([pron(je),v(suis),adj('déçue'),prep(par),det(une),n(t
ablette),prep(de),n(chocolat),n('Nom_du_Produit')]),chunked_sentence([g
n([pron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gprep([prep(par)]),gn(
[det(une),n(tablette),prep(de),n(chocolat),n('Nom_du_Produit')])]),clea
n_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gp
rep([prep(par)]),gn([n(tablette),prep(de),n(chocolat),n('Nom_du_Produit
')])])],['déçue',[je,suis,'déçue',pour,la,'première',fois,depuis,'25',a
ns],sentence([pron(je),v(suis),adj('déçue'),prep(pour),det(la),adj('pre
mière'),n(fois),prep(depuis),adj('25'),n(ans)]),chunked_sentence([gn([p
ron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gprep([prep(pour)]),gn([de
t(la),adj('première'),n(fois)]),gprep([prep(depuis)]),gn([adj('25'),n(a
ns)])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj(
'déçu')]),gprep([prep(pour)]),gn([n(fois)]),gprep([prep(depuis)]),gn([n
(ans)])])],['déçue',[je,suis,donc,'très','déçue',quant,à,la,éthique,de,
votre,'société'],sentence([pron(je),v(suis),adv(donc),adv('très'),adj('
déçue'),prep(quant),prep(à),det(la),n(éthique),prep(de),det(votre),n('s
ociété')]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadv([adv(don
c)]),gadj([adv('très'),adj('déçue')]),gprep([prep(quant),prep(à)]),gn([
det(la),n(éthique),prep(de),det(votre),n('société')])]),clean_chunked_s
entence([gn([pron(je)]),gv([v(être)]),gadv([adv(donc)]),gadj([adj('déçu
')]),gprep([prep(quant),prep(à)]),gn([n(éthique),prep(de),n('société')]
)])],['déçu',[je,suis,'très','déçu',de,les,truffes,noir,'70','%'],sente
nce([pron(je),v(suis),adv('très'),adj('déçu'),prep(de),det(les),n(truff
es),adj(noir),adj('70'),n('%')]),chunked_sentence([gn([pron(je)]),gv([v
(suis)]),gadj([adv('très'),adj('déçu')]),gprep([prep(de)]),gn([det(les)
,n(truffes),adj(noir)]),gn([adj('70'),n('%')])]),clean_chunked_sentence
([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([prep(de)]),gn
([n(truffe)]),gn([n(pourcent)])])],['déçue',[je,suis,'très','déçue',de,
ce,produit],sentence([pron(je),v(suis),adv('très'),adj('déçue'),prep(de
),det(ce),n(produit)]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),g
adj([adv('très'),adj('déçue')]),gprep([prep(de)]),gn([det(ce),n(produit
)])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('d
éçu')]),gprep([prep(de)]),gn([n(produit)])])],['déçu',[je,suis,'très','
déçu',par,le,menu],sentence([pron(je),v(suis),adv('très'),adj('déçu'),p
rep(par),det(le),n(menu)]),chunked_sentence([gn([pron(je)]),gv([v(suis)
]),gadj([adv('très'),adj('déçu')]),gprep([prep(par)]),gn([det(le),n(men
u)])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('

déçu')]),gprep([prep(par)]),gn([n(menu)])])],['déçue',[je,suis,une,cons
ommatrice,'très','déçue',de,votre,grande,marque],sentence([pron(je),v(s
uis),det(une),n(consommatrice),adv('très'),adj('déçue'),prep(de),det(vo
tre),adj(grande),n(marque)]),chunked_sentence([gn([pron(je)]),gv([v(sui
s)]),gn([det(une),n(consommatrice),adv('très'),adj('déçue')]),gprep([pr
ep(de)]),gn([det(votre),adj(grande),n(marque)])]),clean_chunked_sentenc
e([gn([pron(je)]),gv([v(être)]),[gn([n(consommatrice)]),gadj([adj('déçu
')])],gprep([prep(de)]),gn([det(votre),n(marque)])])],['déçu',[je,suis,
vraiment,'déçu',de,vos,pizzas,'surgelées','Nom_du_Produit'],sentence([p
ron(je),v(suis),adv(vraiment),adj('déçu'),prep(de),det(vos),n(pizzas),a
dj('surgelées'),n('Nom_du_Produit')]),chunked_sentence([gn([pron(je)]),
gv([v(suis)]),gadj([adv(vraiment),adj('déçu')]),gprep([prep(de)]),gn([d
et(vos),n(pizzas),adj('surgelées'),n('Nom_du_Produit')])]),clean_chunke
d_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([pre
p(de)]),gn([det(votre),n(pizza),n('Nom_du_Produit')])])],['déçue',[je,s
uis,'déçue'],sentence([pron(je),v(suis),adj('déçue')]),chunked_sentence
([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')])]),clean_chunked_sen
tence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')])])],['déçue',[je
,suis,'très','déçue'],sentence([pron(je),v(suis),adv('très'),adj('déçue
')]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('très'),a
dj('déçue')])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),ga
dj([adj('déçu')])])],['déçu',[je,ai,'été','très','déçu',ce,jour],senten
ce([pron(je),aux(ai),v('été'),adv('très'),adj('déçu'),det(ce),n(jour)])
,chunked_sentence([gn([pron(je)]),gv([aux(ai),v('été')]),gadj([adv('très
'),adj('déçu')]),gn([det(ce),n(jour)])]),clean_chunked_sentence([gn([p
ron(je)]),gv([v(être)]),gadj([adj('déçu')]),gn([n(jour)])])],['déçue',[
je,en,ai,'été','déçue'],sentence([pron(je),pron(en),aux(ai),v('été'),ad
j('déçue')]),chunked_sentence([gn([pron(je)]),gn([pron(en)]),gv([aux(ai
),v('été')]),gadj([adj('déçue')])]),clean_chunked_sentence([gn([pron(je
)]),gn([pron(en)]),gv([v(être)]),gadj([adj('déçu')])])],['déçue',[je,su
is,'particulièrement','déçue',de,mon,dernier,achat],sentence([pron(je),
v(suis),adv('particulièrement'),adj('déçue'),prep(de),det(mon),adj(dern
ier),n(achat)]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([ad
v('particulièrement'),adj('déçue')]),gprep([prep(de)]),gn([det(mon),adj
(dernier),n(achat)])]),clean_chunked_sentence([gn([pron(je)]),gv([v(êtr
e)]),gadj([adj('déçu')]),gprep([prep(de)]),gn([det(mon),n(achat)])])],[
'déçus',[nous,avons,'été','déçus'],sentence([pron(nous),aux(avons),v('é
té'),adj('déçus')]),chunked_sentence([gn([pron(nous)]),gv([aux(avons),v
('été')]),gadj([adj('déçus')])]),clean_chunked_sentence([gn([pron(nous)
]),gv([v(être)]),gadj([adj('déçu')])])]]).

B.16 Données de sortie du module 4 chunked_sentences_cleaned.log

```
keyword: 'déçoit'
words list: [cela,me,'déçoit']
tagged sentence: sentence([pron(cela),pron(me),v('déçoit')])
chunked sentence:
chunked_sentence([gn([pron(cela)]),gn([pron(me)]),gv([v('déçoit')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(cela)]),gn([pron(me)]),gv([v('décevoir
')])])
      gn([pron(cela)]) ==> gn([pron(cela)])
      gn([pron(me)]) ==> gn([pron(me)])
      gv([v('déçoit')]) ==> gv([v('décevoir')])


keyword: 'déçu'
words list: ['celle-ci',me,a,beaucoup,'déçu']
tagged sentence: sentence([pron('celle-
ci'),pron(me),aux(a),adv(beaucoup),v('déçu')])
chunked sentence: chunked_sentence([gn([pron('celle-
ci')]),gn([pron(me)]),gv([aux(a),adv(beaucoup),v('déçu')])])
chunked sentence cleaned: clean_chunked_sentence([gn([pron('celle-
ci')]),gn([pron(me)]),gv([v('décevoir')])])
      gn([pron('celle-ci')]) ==> gn([pron('celle-ci')])
      gn([pron(me)]) ==> gn([pron(me)])
      gv([aux(a),adv(beaucoup),v('déçu')]) ==> gv([v('décevoir')])


keyword: 'déçoit'
words list: ['La_Marque',me,'déçoit']
tagged sentence: sentence([n('La_Marque'),pron(me),v('déçoit')])
chunked sentence:
chunked_sentence([gn([n('La_Marque')]),gn([pron(me)]),gv([v('déçoit')])
])
chunked sentence cleaned:
clean_chunked_sentence([gn([n('La_Marque')]),gn([pron(me)]),gv([v('déce
voir')])])
      gn([n('La_Marque')]) ==> gn([n('La_Marque')])
      gn([pron(me)]) ==> gn([pron(me)])
      gv([v('déçoit')]) ==> gv([v('décevoir')])


keyword: 'décevant'
words list: [je,trouve,ça,'très','décevant']
tagged sentence:
sentence([pron(je),v(trouve),pron(ça),adv('très'),adj('décevant')])
```

```
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(trouve)]),gn([pron(ça)]),gadj([a
dv('très'),adj('décevant')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(trouver)]),gn([pron(ça)]),
gadj([adj('décevant')])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(trouve)]) ==> gv([v(trouver)])
      gn([pron(ça)]) ==> gn([pron(ça)])
      gadj([adv('très'),adj('décevant')]) ==> gadj([adj('décevant')])


keyword: 'déception'
words list: [à,ma,grande,'déception']
tagged sentence: sentence([prep(à),det(ma),adj(grande),n('déception')])
chunked sentence:
chunked_sentence([gprep([prep(à)]),gn([det(ma),adj(grande),n('déception
')])])
chunked sentence cleaned:
clean_chunked_sentence([gprep([prep(à)]),gn([det(ma),n('déception')])])
      gprep([prep(à)]) ==> gprep([prep(à)])
      gn([det(ma),adj(grande),n('déception')]) ==>
gn([det(ma),n('déception')])


keyword: 'déception'
words list: [ce,courrier,pour,vous,faire,part,de,ma,'déception']
tagged sentence:
sentence([det(ce),n(courrier),prep(pour),pron(vous),v(faire),n(part),pr
ep(de),det(ma),n('déception')])
chunked sentence:
chunked_sentence([gn([det(ce),n(courrier)]),gprep([prep(pour)]),gn([pro
n(vous)]),expression([v(faire),n(part),prep(de)]),gn([det(ma),n('décept
ion')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([n(courrier)]),gprep([prep(pour)]),gn([pron(
vous)]),expression([v(faire),n(part),prep(de)]),gn([det(ma),n('déceptio
n')])])
      gn([det(ce),n(courrier)]) ==> gn([n(courrier)])
      gprep([prep(pour)]) ==> gprep([prep(pour)])
      gn([pron(vous)]) ==> gn([pron(vous)])
      expression([v(faire),n(part),prep(de)]) ==>
expression([v(faire),n(part),prep(de)])
      gn([det(ma),n('déception')]) ==> gn([det(ma),n('déception')])
```

```
keyword: 'déception'
words list:
[quelle,'déception','même',pas,une,'décoration',sur,les,'bûches']
tagged sentence:
sentence([adj(quelle),n('déception'),adj('même'),adv(pas),det(une),n('d
écoration'),prep(sur),det(les),n('bûches')])
chunked sentence:
chunked_sentence([gn([adj(quelle),n('déception')]),gadj([adj('même')]),
gadv([adv(pas)]),gn([det(une),n('décoration'),prep(sur),det(les),n('bûc
hes')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([n('déception')]),gadj([adj('même')]),gadv([
adv(pas)]),gn([n('décoration'),prep(sur),n('bûche')])])
      gn([adj(quelle),n('déception')]) ==> gn([n('déception')])
      gadj([adj('même')]) ==> gadj([adj('même')])
      gadv([adv(pas)]) ==> gadv([adv(pas)])
      gn([det(une),n('décoration'),prep(sur),det(les),n('bûches')]) ==>
gn([n('décoration'),prep(sur),n('bûche')])


keyword: 'déception'
words list: [je,vous,fais,part,de,ma,'déception']
tagged sentence:
sentence([pron(je),pron(vous),v(fais),n(part),prep(de),det(ma),n('décep
tion')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gn([pron(vous)]),expression([v(fais),n
(part),prep(de)]),gn([det(ma),n('déception')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gn([pron(vous)]),expression([v(f
aire),n(part),prep(de)]),gn([det(ma),n('déception')])])
      gn([pron(je)]) ==> gn([pron(je)])
      gn([pron(vous)]) ==> gn([pron(vous)])
      expression([v(fais),n(part),prep(de)]) ==>
expression([v(faire),n(part),prep(de)])
      gn([det(ma),n('déception')]) ==> gn([det(ma),n('déception')])


keyword: 'déception'
words list: [je,vous,fais,part,de,ma,'déception','vis-à-
vis',de,le,'Nom_du_Produit']
tagged sentence:
sentence([pron(je),pron(vous),v(fais),n(part),prep(de),det(ma),n('décep
tion'),prep('vis-à-vis'),prep(de),det(le),n('Nom_du_Produit')])
```

```
chunked sentence:
chunked_sentence([gn([pron(je)]),gn([pron(vous)]),expression([v(fais),n
(part),prep(de)]),gn([det(ma),n('déception')]),gprep([prep('vis-à-
vis'),prep(de)]),gn([det(le),n('Nom_du_Produit')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gn([pron(vous)]),expression([v(f
aire),n(part),prep(de)]),gn([det(ma),n('déception')]),gprep([prep('vis-
à-vis'),prep(de)]),gn([n('Nom_du_Produit')])])
     gn([pron(je)]) ==> gn([pron(je)])
     gn([pron(vous)]) ==> gn([pron(vous)])
     expression([v(fais),n(part),prep(de)]) ==>
expression([v(faire),n(part),prep(de)])
     gn([det(ma),n('déception')]) ==> gn([det(ma),n('déception')])
     gprep([prep('vis-à-vis'),prep(de)]) ==> gprep([prep('vis-à-
vis'),prep(de)])
     gn([det(le),n('Nom_du_Produit')]) ==> gn([n('Nom_du_Produit')])


keyword: 'déception'
words list: [la,'déception',de,un,enfant]
tagged sentence:
sentence([det(la),n('déception'),prep(de),det(un),n(enfant)])
chunked sentence:
chunked_sentence([gn([det(la),n('déception'),prep(de),det(un),n(enfant)
])])
chunked sentence cleaned:
clean_chunked_sentence([gn([n('déception'),prep(de),n(enfant)])])
     gn([det(la),n('déception'),prep(de),det(un),n(enfant)]) ==>
gn([n('déception'),prep(de),n(enfant)])


keyword: 'déception'
words list: [quel,ne,est,pas,ma,'déception']
tagged sentence:
sentence([adj(quel),adv(ne),v(est),adv(pas),det(ma),n('déception')])
chunked sentence:
chunked_sentence([expression([adj(quel),adv(ne),v(est),adv(pas)]),gn([d
et(ma),n('déception')])])
chunked sentence cleaned:
clean_chunked_sentence([expression([adj(quel),adv(ne),v(être),adv(pas)]
),gn([det(ma),n('déception')])])
     expression([adj(quel),adv(ne),v(est),adv(pas)]) ==>
expression([adj(quel),adv(ne),v(être),adv(pas)])
     gn([det(ma),n('déception')]) ==> gn([det(ma),n('déception')])
```

```
keyword: 'déçu'
words list: [je,ai,'été','déçu',lors,de,mon,dernier,achat]
tagged sentence:
sentence([pron(je),aux(ai),v('été'),adj('déçu'),adv(lors),prep(de),det(
mon),adj(dernier),n(achat)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([aux(ai),v('été')]),gadj([adj('déçu
')]),gprep([adv(lors),prep(de)]),gn([det(mon),adj(dernier),n(achat)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([adv(lors),prep(de)]),gn([det(mon),n(achat)])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([aux(ai),v('été')]) ==> gv([v(être)])
      gadj([adj('déçu')]) ==> gadj([adj('déçu')])
      gprep([adv(lors),prep(de)]) ==> gprep([adv(lors),prep(de)])
      gn([det(mon),adj(dernier),n(achat)]) ==> gn([det(mon),n(achat)])


keyword: 'déçue'
words list: [je,ai,'été','déçue',par,le,'côté',âpre,de,cet,article]
tagged sentence:
sentence([pron(je),aux(ai),v('été'),adj('déçue'),prep(par),det(le),n('c
ôté'),adj(âpre),prep(de),det(cet),n(article)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([aux(ai),v('été')]),gadj([adj('déçu
e')]),gprep([prep(par)]),gn([det(le),n('côté'),adj(âpre),prep(de),det(c
et),n(article)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(par)]),gn([n('côté'),prep(de),n(article)])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([aux(ai),v('été')]) ==> gv([v(être)])
      gadj([adj('déçue')]) ==> gadj([adj('déçu')])
      gprep([prep(par)]) ==> gprep([prep(par)])
      gn([det(le),n('côté'),adj(âpre),prep(de),det(cet),n(article)])
==> gn([n('côté'),prep(de),n(article)])


keyword: 'déçue'
words list: [je,me,suis,'trouvée','très','déçue',de,ce,produit]
tagged sentence:
sentence([pron(je),pron(me),aux(suis),v('trouvée'),adv('très'),adj('déç
ue'),prep(de),det(ce),n(produit)])
```

```
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([pron(me),aux(suis),v('trouvée')]),
gadj([adv('très'),adj('déçue')]),gprep([prep(de)]),gn([det(ce),n(produi
t)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([pron_v(me),v_pron(trouver)])
,gadj([adj('déçu')]),gprep([prep(de)]),gn([n(produit)])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([pron(me),aux(suis),v('trouvée')]) ==>
gv([pron_v(me),v_pron(trouver)])
      gadj([adv('très'),adj('déçue')]) ==> gadj([adj('déçu')])
      gprep([prep(de)]) ==> gprep([prep(de)])
      gn([det(ce),n(produit)]) ==> gn([n(produit)])


keyword: 'déçue'
words list: [je,suis,'amèrement','déçue']
tagged sentence:
sentence([pron(je),v(suis),adv('amèrement'),adj('déçue')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('amèrement'),a
dj('déçue')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
)])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gadj([adv('amèrement'),adj('déçue')]) ==> gadj([adj('déçu')])


keyword: 'déçue'
words list: [je,suis,'déçue',de,leur,remarque]
tagged sentence:
sentence([pron(je),v(suis),adj('déçue'),prep(de),det(leur),n(remarque)]
)
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gpr
ep([prep(de)]),gn([det(leur),n(remarque)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(de)]),gn([n(remarque)])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gadj([adj('déçue')]) ==> gadj([adj('déçu')])
      gprep([prep(de)]) ==> gprep([prep(de)])
```

```
      gn([det(leur),n(remarque)]) ==> gn([n(remarque)])


keyword: 'déçu'
words list: [je,suis,'déçu',de,mes,deux,derniers,achats,identiques]
tagged sentence:
sentence([pron(je),v(suis),adj('déçu'),prep(de),det(mes),adj(deux),adj(
derniers),n(achats),adj(identiques)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçu')]),gpre
p([prep(de)]),gn([det(mes),adj(deux),adj(derniers),n(achats),adj(identi
ques)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(de)]),gn([det(mon),n(achat)])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gadj([adj('déçu')]) ==> gadj([adj('déçu')])
      gprep([prep(de)]) ==> gprep([prep(de)])
      gn([det(mes),adj(deux),adj(derniers),n(achats),adj(identiques)])
==> gn([det(mon),n(achat)])


keyword: 'déçue'
words list: [je,suis,'déçue',de,votre,marque,'La_Marque']
tagged sentence:
sentence([pron(je),v(suis),adj('déçue'),prep(de),det(votre),n(marque),n
('La_Marque')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gpr
ep([prep(de)]),gn([det(votre),n(marque),n('La_Marque')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(de)]),gn([det(votre),n(marque),n('La_Marque')])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gadj([adj('déçue')]) ==> gadj([adj('déçu')])
      gprep([prep(de)]) ==> gprep([prep(de)])
      gn([det(votre),n(marque),n('La_Marque')]) ==>
gn([det(votre),n(marque),n('La_Marque')])


keyword: 'déçue'
words list:
[je,suis,'déçue',par,une,tablette,de,chocolat,'Nom_du_Produit']
```

```
tagged sentence:
sentence([pron(je),v(suis),adj('déçue'),prep(par),det(une),n(tablette),
prep(de),n(chocolat),n('Nom_du_Produit')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gpr
ep([prep(par)]),gn([det(une),n(tablette),prep(de),n(chocolat),n('Nom_du
_Produit')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(par)]),gn([n(tablette),prep(de),n(chocolat),n('Nom_du_Pro
duit')])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gadj([adj('déçue')]) ==> gadj([adj('déçu')])
      gprep([prep(par)]) ==> gprep([prep(par)])
      gn([det(une),n(tablette),prep(de),n(chocolat),n('Nom_du_Produit')
]) ==> gn([n(tablette),prep(de),n(chocolat),n('Nom_du_Produit')])


keyword: 'déçue'
words list: [je,suis,'déçue',pour,la,'première',fois,depuis,'25',ans]
tagged sentence:
sentence([pron(je),v(suis),adj('déçue'),prep(pour),det(la),adj('premièr
e'),n(fois),prep(depuis),adj('25'),n(ans)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gpr
ep([prep(pour)]),gn([det(la),adj('première'),n(fois)]),gprep([prep(depu
is)]),gn([adj('25'),n(ans)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(pour)]),gn([n(fois)]),gprep([prep(depuis)]),gn([n(ans)])]
)
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gadj([adj('déçue')]) ==> gadj([adj('déçu')])
      gprep([prep(pour)]) ==> gprep([prep(pour)])
      gn([det(la),adj('première'),n(fois)]) ==> gn([n(fois)])
      gprep([prep(depuis)]) ==> gprep([prep(depuis)])
      gn([adj('25'),n(ans)]) ==> gn([n(ans)])


keyword: 'déçue'
words list:
[je,suis,donc,'très','déçue',quant,à,la,éthique,de,votre,'société']
```

```
tagged sentence:
sentence([pron(je),v(suis),adv(donc),adv('très'),adj('déçue'),prep(quan
t),prep(à),det(la),n(éthique),prep(de),det(votre),n('société')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadv([adv(donc)]),gadj([
adv('très'),adj('déçue')]),gprep([prep(quant),prep(à)]),gn([det(la),n(é
thique),prep(de),det(votre),n('société')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadv([adv(donc)]),
gadj([adj('déçu')]),gprep([prep(quant),prep(à)]),gn([n(éthique),prep(de
),n('société')])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gadv([adv(donc)]) ==> gadv([adv(donc)])
      gadj([adv('très'),adj('déçue')]) ==> gadj([adj('déçu')])
      gprep([prep(quant),prep(à)]) ==> gprep([prep(quant),prep(à)])
      gn([det(la),n(éthique),prep(de),det(votre),n('société')]) ==>
gn([n(éthique),prep(de),n('société')])


keyword: 'déçu'
words list: [je,suis,'très','déçu',de,les,truffes,noir,'70','%']
tagged sentence:
sentence([pron(je),v(suis),adv('très'),adj('déçu'),prep(de),det(les),n(
truffes),adj(noir),adj('70'),n('%')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('très'),adj('d
éçu')]),gprep([prep(de)]),gn([det(les),n(truffes),adj(noir)]),gn([adj('
70'),n('%')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(de)]),gn([n(truffe)]),gn([n(pourcent)])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gadj([adv('très'),adj('déçu')]) ==> gadj([adj('déçu')])
      gprep([prep(de)]) ==> gprep([prep(de)])
      gn([det(les),n(truffes),adj(noir)]) ==> gn([n(truffe)])
      gn([adj('70'),n('%')]) ==> gn([n(pourcent)])


keyword: 'déçue'
words list: [je,suis,'très','déçue',de,ce,produit]
tagged sentence:
sentence([pron(je),v(suis),adv('très'),adj('déçue'),prep(de),det(ce),n(
produit)])
```

```
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('très'),adj('d
éçue')]),gprep([prep(de)]),gn([det(ce),n(produit)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(de)]),gn([n(produit)])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gadj([adv('très'),adj('déçue')]) ==> gadj([adj('déçu')])
      gprep([prep(de)]) ==> gprep([prep(de)])
      gn([det(ce),n(produit)]) ==> gn([n(produit)])


keyword: 'déçu'
words list: [je,suis,'très','déçu',par,le,menu]
tagged sentence:
sentence([pron(je),v(suis),adv('très'),adj('déçu'),prep(par),det(le),n(
menu)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('très'),adj('d
éçu')]),gprep([prep(par)]),gn([det(le),n(menu)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(par)]),gn([n(menu)])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gadj([adv('très'),adj('déçu')]) ==> gadj([adj('déçu')])
      gprep([prep(par)]) ==> gprep([prep(par)])
      gn([det(le),n(menu)]) ==> gn([n(menu)])


keyword: 'déçue'
words list:
[je,suis,une,consommatrice,'très','déçue',de,votre,grande,marque]
tagged sentence:
sentence([pron(je),v(suis),det(une),n(consommatrice),adv('très'),adj('d
éçue'),prep(de),det(votre),adj(grande),n(marque)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gn([det(une),n(consommat
rice),adv('très'),adj('déçue')]),gprep([prep(de)]),gn([det(votre),adj(g
rande),n(marque)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),[gn([n(consommatri
ce)]),gadj([adj('déçu')])],gprep([prep(de)]),gn([det(votre),n(marque)])
])
```

```
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gn([det(une),n(consommatrice),adv('très'),adj('déçue')]) ==>
[gn([n(consommatrice)]),gadj([adj('déçu')])]
      gprep([prep(de)]) ==> gprep([prep(de)])
      gn([det(votre),adj(grande),n(marque)]) ==>
gn([det(votre),n(marque)])


keyword: 'déçu'
words list:
[je,suis,vraiment,'déçu',de,vos,pizzas,'surgelées','Nom_du_Produit']
tagged sentence:
sentence([pron(je),v(suis),adv(vraiment),adj('déçu'),prep(de),det(vos),
n(pizzas),adj('surgelées'),n('Nom_du_Produit')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv(vraiment),adj(
'déçu')]),gprep([prep(de)]),gn([det(vos),n(pizzas),adj('surgelées'),n('
Nom_du_Produit')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(de)]),gn([det(votre),n(pizza),n('Nom_du_Produit')])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gadj([adv(vraiment),adj('déçu')]) ==> gadj([adj('déçu')])
      gprep([prep(de)]) ==> gprep([prep(de)])
      gn([det(vos),n(pizzas),adj('surgelées'),n('Nom_du_Produit')]) ==>
gn([det(votre),n(pizza),n('Nom_du_Produit')])


keyword: 'déçue'
words list: [je,suis,'déçue']
tagged sentence: sentence([pron(je),v(suis),adj('déçue')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
)])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gadj([adj('déçue')]) ==> gadj([adj('déçu')])


keyword: 'déçue'
words list: [je,suis,'très','déçue']
```

```
tagged sentence: sentence([pron(je),v(suis),adv('très'),adj('déçue')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('très'),adj('d
éçue')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
)])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gadj([adv('très'),adj('déçue')]) ==> gadj([adj('déçu')])


keyword: 'déçu'
words list: [je,ai,'été','très','déçu',ce,jour]
tagged sentence:
sentence([pron(je),aux(ai),v('été'),adv('très'),adj('déçu'),det(ce),n(j
our)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([aux(ai),v('été')]),gadj([adv('très
'),adj('déçu')]),gn([det(ce),n(jour)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gn([n(jour)])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([aux(ai),v('été')]) ==> gv([v(être)])
      gadj([adv('très'),adj('déçu')]) ==> gadj([adj('déçu')])
      gn([det(ce),n(jour)]) ==> gn([n(jour)])


keyword: 'déçue'
words list: [je,en,ai,'été','déçue']
tagged sentence:
sentence([pron(je),pron(en),aux(ai),v('été'),adj('déçue')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gn([pron(en)]),gv([aux(ai),v('été')]),
gadj([adj('déçue')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gn([pron(en)]),gv([v(être)]),gad
j([adj('déçu')])])
      gn([pron(je)]) ==> gn([pron(je)])
      gn([pron(en)]) ==> gn([pron(en)])
      gv([aux(ai),v('été')]) ==> gv([v(être)])
      gadj([adj('déçue')]) ==> gadj([adj('déçu')])
```

```
keyword: 'déçue'
words list: [je,suis,'particulièrement','déçue',de,mon,dernier,achat]
tagged sentence:
sentence([pron(je),v(suis),adv('particulièrement'),adj('déçue'),prep(de
),det(mon),adj(dernier),n(achat)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('particulièrem
ent'),adj('déçue')]),gprep([prep(de)]),gn([det(mon),adj(dernier),n(acha
t)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(de)]),gn([det(mon),n(achat)])])
      gn([pron(je)]) ==> gn([pron(je)])
      gv([v(suis)]) ==> gv([v(être)])
      gadj([adv('particulièrement'),adj('déçue')]) ==>
gadj([adj('déçu')])
      gprep([prep(de)]) ==> gprep([prep(de)])
      gn([det(mon),adj(dernier),n(achat)]) ==> gn([det(mon),n(achat)])


keyword: 'déçus'
words list: [nous,avons,'été','déçus']
tagged sentence:
sentence([pron(nous),aux(avons),v('été'),adj('déçus')])
chunked sentence:
chunked_sentence([gn([pron(nous)]),gv([aux(avons),v('été')]),gadj([adj(
'déçus')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(nous)]),gv([v(être)]),gadj([adj('déçu'
)])])
      gn([pron(nous)]) ==> gn([pron(nous)])
      gv([aux(avons),v('été')]) ==> gv([v(être)])
      gadj([adj('déçus')]) ==> gadj([adj('déçu')])
```

B.17 Codes du programme noyau module5.pl

```
/* module 5: mapping to predicate-argument structures */
/* Wannachai Kampeera */

date('20130107').

:- ensure_loaded('derives.pl'), ensure_loaded('ontologies.pl').
:- op(500,xfx,':').

pred_arg_file_name('pred_arg_structures.pl').
chunks_cleaned_log_file('pred_arg_structures.log').
module5_input_file('chunked_sentences_cleaned.pl').
final_output_file('final_structures.txt').

mapping :-
     module5_input_file(FileName),
     load_module5_input(FileName, ChunkedSentences),
     clean_chunked_sentences(ListOfChunkedSentences) =
ChunkedSentences,
     map_to_str(ListOfChunkedSentences, PredArgStructures),
     write_pred_arg_to_file(PredArgStructures),
     write_pred_arg_log(PredArgStructures),
     write_final_output(PredArgStructures),
     !.

load_module5_input(FileName, SentencesList):-
     open(FileName, 'read', Stream),
     read(Stream, SentencesList),
     close(Stream).

map_to_str([], []).
map_to_str([ChunkedSentence|RestChunkedSentences],
[KeWoSeChClLaStr|Rest_KeWoSeChCl]):-
     ChunkedSentence = [KeyWord, WordsList, sentence(Sentence),
                  chunked_sentence(ListOfChunks),
                  clean_chunked_sentence(CleanChunkedSentence)],
     flatten(CleanChunkedSentence, FlattenChunkedSentence),
     mapping(FlattenChunkedSentence, LabeledSentence),
     flatten(LabeledSentence, PredArgStructure),
     KeWoSeChClLaStr = [KeyWord, WordsList,
                sentence(Sentence),
                chunked_sentence(ListOfChunks),
                clean_chunked_sentence(CleanChunkedSentence),
                labeled_chunked_sentence(LabeledSentence),
                     pred_arg_structure(PredArgStructure)],
```

```
      map_to_str(RestChunkedSentences, Rest_KeWoSeChCl),
      !.

mapping([], []).
mapping([WordGroup|RestWordGroups], [LabeledWords|RestLabeledWords]):-
      WordGroup =.. [GroupName, TaggedWordList],
      GroupName \== 'expression',
      labeling(TaggedWordList, LabeledWords),
      mapping(RestWordGroups, RestLabeledWords).
mapping([WordGroup|RestWordGroups], [LabeledWords|RestLabeledWords]):-
      WordGroup =.. [GroupName, TaggedWordList],
      GroupName == 'expression',
      copy_expression(TaggedWordList, LabeledWords),
      mapping(RestWordGroups, RestLabeledWords).

labeling([], []).
labeling([TaggedWord|RestWords], [LabeledWord|RestLabelWords]):-
      TaggedWord =.. [POS,Word ],
      labeling(POS, Word, LabeledWord),
      !,
      labeling(RestWords, RestLabelWords).

/* case 1: predicate, Adj/N/V/ADV, words belonging to keywords list */
labeling(POS, Word, pred(Word)):-
      member(POS,['adj', 'n', 'v', 'adv']),
      derives(DerivesList),
      member(Word, DerivesList).

/* case 2: arg1, n belonging to the client domain */
labeling(POS, Word, arg1('entreprise':POSWord)):-
      member(POS,['n', pron, det]),
      company(CompanyWords),
      member(Word, CompanyWords),
      POSWord =.. [POS, Word].

/* case 3: arg2, n belonging to the company domain */
labeling(POS, Word, arg2('client':POSWord)):-
      member(POS,['n', pron, det]),
      client(ClientWords),
      member(Word, ClientWords),
      POSWord =.. [POS, Word].

/* case 3: other nouns not belonging to the client/company domain */
labeling(POS, Word, arg('domaine':POSWord)):-
      member(POS,['n', pron]),
```

```
      company(ClientWords),
      company(CompanyWords),
      \+ member(Word, CompanyWords),
      \+ member(Word, ClientWords),
      POSWord =.. [POS, Word].


/* case 4: others, all POS, just copy */
labeling(_POS, Word, Word).

/* case 5: copy expressions */
copy_expression([], []).
copy_expression([TaggedWord|RestTaggedWords], [Word|RestWords]):-
      TaggedWord =.. [_POS, Word],
      copy_expression(RestTaggedWords, RestWords).

write_pred_arg_to_file(CleanChunkedSentences):-
      pred_arg_file_name(FileName),
      open(FileName, 'write', Stream),
      writeq(Stream, clean_chunked_sentences(CleanChunkedSentences)),
      write(Stream, '.'),
      close(Stream).

write_pred_arg_log(ChunkedSentences):-
      chunks_cleaned_log_file(FileName),
      open(FileName, 'write', Stream),
      write_pred_arg_log(ChunkedSentences, Stream),
      close(Stream).

write_pred_arg_log([], _Stream).
write_pred_arg_log([Sentence|RestSentences], Stream):-
      Sentence = [KeyWord, WordList, TaggedSentence, ChunkedSentences,
CleanChunks, LabeledSentence, PredArgStr],
      write(Stream, 'keyword: '),
      writeq(Stream, KeyWord),
      write(Stream, '\n'),
      write(Stream, 'words list: '),
      writeq(Stream, WordList),
      write(Stream, '\n'),
      write(Stream, 'tagged sentence: '),
      writeq(Stream, TaggedSentence),
      write(Stream, '\n'),
      write(Stream, 'chunked sentence: '),
      writeq(Stream, ChunkedSentences),
      write(Stream, '\n'),
```

```
      write(Stream, 'chunked sentence cleaned: '),
      writeq(Stream, CleanChunks),
      write(Stream, '\n'),
      write(Stream, 'labeled entence: '),
      writeq(Stream, LabeledSentence),
      write(Stream, '\n'),
      write(Stream, 'Predicate-argument structure: '),
      writeq(Stream, PredArgStr),
      write(Stream, '\n'),
      write(Stream, '\n'),
      write_pred_arg_log(RestSentences, Stream).

write_final_output(Sentences):-
      get_str_list(Sentences, ListOfStructures),
      list_to_set(ListOfStructures, SetOfStructures),
      final_output_file(FileName),
      open(FileName, 'write', Stream),
      write_str_status(SetOfStructures, Stream, 0),
      close(Stream).

get_str_list([], []).
get_str_list([Sentence|RestSentences], [PredArgStr|RestStructures]):-
      Sentence = [_KeyWord, _WordList, _TaggedSentence,
                _ChunkedSentences,
                _CleanChunks, _LabeledSentence,
                pred_arg_structure(PredArgStr)],
      get_str_list(RestSentences, RestStructures).

write_str_status([], _Stream, _Counter).
write_str_status([ListOfWords|RestStructures], Stream, Counter):-
      NewCounter is Counter + 1,
      write(Stream, 'structure '),
      write(Stream, NewCounter),
      write(Stream, ': '),
      write_final_word(ListOfWords, Stream),
      (
      \+ member(arg(_), ListOfWords),
      write(Stream, '\n'),
      write(Stream, 'status: OK')
          ;
      member(arg(_), ListOfWords),
      write(Stream, '\n'),
      write(Stream, 'status: to be reviewed "undefined domaine"')
      ),
      !,
```

```
      write(Stream, '\n'),
      write(Stream, '\n'),
      write_str_status(RestStructures, Stream, NewCounter).

write_final_word([], _Stream).
write_final_word([Word|RestWords], Stream):-
      write(Stream, Word),
      write(Stream, ' '),
      write_final_word(RestWords, Stream).

/* End of Programme */
```

B.18 Ontologies pour libeller les arguments ontologies.pl

```
client([
me,
je,
ma,
moi,
mon,
mes,
nous,
notre,
enfant,
consommatrice,
consommateur]).

company([
'La_Marque',
vous,
votre,
vos,
bûche,
'Nom_du_Produit',
chocolat,
'société',
truffe,
produit,
menu,
marque,
article,
pizza,
entreprise]).
```

B.19 Liste des dérivés pour libeller le prédicat derives.pl

```
derives([
    'décevoir',
    'déçu',
    'décevant',
    'déception',
    'ennui',
    'désappointement',
    'désappointer',
    'déconvenue',
    'désenchantement',
    'désillusion',
    'chagrin',
    'insatisfaisant',
    'insatisfaisante',
    'frustrer',
    'frustré',
    'frustrant'
     ]).
```

B.20 Sortie du module 5 pred_arg_structures.pl

```
clean_chunked_sentences([['déçoit',[cela,me,'déçoit'],sentence([pron(ce
la),pron(me),v('déçoit')]),chunked_sentence([gn([pron(cela)]),gn([pron(
me)]),gv([v('déçoit')])]),clean_chunked_sentence([gn([pron(cela)]),gn([
pron(me)]),gv([v('décevoir')])]),labeled_chunked_sentence([[arg(domaine
:pron(cela))],[arg2(client:pron(me))],[pred('décevoir')]]),pred_arg_str
ucture([arg(domaine:pron(cela)),arg2(client:pron(me)),pred('décevoir')]
)],['déçu',['celle-ci',me,a,beaucoup,'déçu'],sentence([pron('celle-
ci'),pron(me),aux(a),adv(beaucoup),v('déçu')]),chunked_sentence([gn([pr
on('celle-
ci')]),gn([pron(me)]),gv([aux(a),adv(beaucoup),v('déçu')])]),clean_chun
ked_sentence([gn([pron('celle-
ci')]),gn([pron(me)]),gv([v('décevoir')])]),labeled_chunked_sentence([[
arg(domaine:pron('celle-
ci'))],[arg2(client:pron(me))],[pred('décevoir')]]),pred_arg_structure(
[arg(domaine:pron('celle-
ci')),arg2(client:pron(me)),pred('décevoir')])],['déçoit',['La_Marque',
me,'déçoit'],sentence([n('La_Marque'),pron(me),v('déçoit')]),chunked_se
ntence([gn([n('La_Marque')]),gn([pron(me)]),gv([v('déçoit')])]),clean_c
hunked_sentence([gn([n('La_Marque')]),gn([pron(me)]),gv([v('décevoir')]
)]),labeled_chunked_sentence([[arg1(entreprise:n('La_Marque'))],[arg2(c
lient:pron(me))],[pred('décevoir')]]),pred_arg_structure([arg1(entrepri
se:n('La_Marque')),arg2(client:pron(me)),pred('décevoir')])],['décevant
',[je,trouve,ça,'très','décevant'],sentence([pron(je),v(trouve),pron(ça
),adv('très'),adj('décevant')]),chunked_sentence([gn([pron(je)]),gv([v(
trouve)]),gn([pron(ça)]),gadj([adv('très'),adj('décevant')])]),clean_ch
unked_sentence([gn([pron(je)]),gv([v(trouver)]),gn([pron(ça)]),gadj([ad
j('décevant')])]),labeled_chunked_sentence([[arg2(client:pron(je))],[tr
ouver],[arg(domaine:pron(ça))],[pred('décevant')]]),pred_arg_structure(
[arg2(client:pron(je)),trouver,arg(domaine:pron(ça)),pred('décevant')])
],['déception',[à,ma,grande,'déception'],sentence([prep(à),det(ma),adj(
grande),n('déception')]),chunked_sentence([gprep([prep(à)]),gn([det(ma)
,adj(grande),n('déception')])]),clean_chunked_sentence([gprep([prep(à)]
),gn([det(ma),n('déception')])]),labeled_chunked_sentence([[à],[arg2(cl
ient:det(ma)),pred('déception')]]),pred_arg_structure([à,arg2(client:de
t(ma)),pred('déception')])],['déception',[ce,courrier,pour,vous,faire,p
art,de,ma,'déception'],sentence([det(ce),n(courrier),prep(pour),pron(vo
us),v(faire),n(part),prep(de),det(ma),n('déception')]),chunked_sentence
([gn([det(ce),n(courrier)]),gprep([prep(pour)]),gn([pron(vous)]),expres
sion([v(faire),n(part),prep(de)]),gn([det(ma),n('déception')])]),clean_
chunked_sentence([gn([n(courrier)]),gprep([prep(pour)]),gn([pron(vous)]
),expression([v(faire),n(part),prep(de)]),gn([det(ma),n('déception')])]
),labeled_chunked_sentence([[arg(domaine:n(courrier))],[pour],[arg1(ent
reprise:pron(vous))],[faire,part,de],[arg2(client:det(ma)),pred('déception')]]),pred_arg_structure([arg(domaine:n(courrier)),pour,arg1(entrepr
ise:pron(vous)),faire,part,de,arg2(client:det(ma)),pred('déception')])]
```

,['déception',[quelle,'déception','même',pas,une,'décoration',sur,les,'
bûches'],sentence([adj(quelle),n('déception'),adj('même'),adv(pas),det(
une),n('décoration'),prep(sur),det(les),n('bûches')]),chunked_sentence(
[gn([adj(quelle),n('déception')]),gadj([adj('même')]),gadv([adv(pas)]),
gn([det(une),n('décoration'),prep(sur),det(les),n('bûches')])]),clean_c
hunked_sentence([gn([n('déception')]),gadj([adj('même')]),gadv([adv(pas
)]),gn([n('décoration'),prep(sur),n('bûche')])]),labeled_chunked_senten
ce([[pred('déception')],['même'],[pas],[arg(domaine:n('décoration')),su
r,arg1(entreprise:n('bûche'))]]),pred_arg_structure([pred('déception'),
'même',pas,arg(domaine:n('décoration')),sur,arg1(entreprise:n('bûche'))
])],['déception',[je,vous,fais,part,de,ma,'déception'],sentence([pron(j
e),pron(vous),v(fais),n(part),prep(de),det(ma),n('déception')]),chunked
_sentence([gn([pron(je)]),gn([pron(vous)]),expression([v(fais),n(part),
prep(de)]),gn([det(ma),n('déception')])]),clean_chunked_sentence([gn([p
ron(je)]),gn([pron(vous)]),expression([v(faire),n(part),prep(de)]),gn([
det(ma),n('déception')])]),labeled_chunked_sentence([[arg2(client:pron(
je))],[arg1(entreprise:pron(vous))],[faire,part,de],[arg2(client:det(ma
)),pred('déception')]]),pred_arg_structure([arg2(client:pron(je)),arg1(
entreprise:pron(vous)),faire,part,de,arg2(client:det(ma)),pred('décepti
on')])],['déception',[je,vous,fais,part,de,ma,'déception','vis-à-
vis',de,le,'Nom_du_Produit'],sentence([pron(je),pron(vous),v(fais),n(pa
rt),prep(de),det(ma),n('déception'),prep('vis-à-
vis'),prep(de),det(le),n('Nom_du_Produit')]),chunked_sentence([gn([pron
(je)]),gn([pron(vous)]),expression([v(fais),n(part),prep(de)]),gn([det(
ma),n('déception')]),gprep([prep('vis-à-
vis'),prep(de)]),gn([det(le),n('Nom_du_Produit')])]),clean_chunked_sent
ence([gn([pron(je)]),gn([pron(vous)]),expression([v(faire),n(part),prep
(de)]),gn([det(ma),n('déception')]),gprep([prep('vis-à-
vis'),prep(de)]),gn([n('Nom_du_Produit')])]),labeled_chunked_sentence([
[arg2(client:pron(je))],[arg1(entreprise:pron(vous))],[faire,part,de],[
arg2(client:det(ma)),pred('déception')],['vis-à-
vis',de],[arg1(entreprise:n('Nom_du_Produit'))]]),pred_arg_structure([a
rg2(client:pron(je)),arg1(entreprise:pron(vous)),faire,part,de,arg2(cli
ent:det(ma)),pred('déception'),'vis-à-
vis',de,arg1(entreprise:n('Nom_du_Produit'))])],['déception',[la,'décep
tion',de,un,enfant],sentence([det(la),n('déception'),prep(de),det(un),n
(enfant)]),chunked_sentence([gn([det(la),n('déception'),prep(de),det(un
),n(enfant)])]),clean_chunked_sentence([gn([n('déception'),prep(de),n(e
nfant)])]),labeled_chunked_sentence([[pred('déception'),de,arg2(client:
n(enfant))]]),pred_arg_structure([pred('déception'),de,arg2(client:n(en
fant))])],['déception',[quel,ne,est,pas,ma,'déception'],sentence([adj(q
uel),adv(ne),v(est),adv(pas),det(ma),n('déception')]),chunked_sentence(
[expression([adj(quel),adv(ne),v(est),adv(pas)]),gn([det(ma),n('décepti
on')])]),clean_chunked_sentence([expression([adj(quel),adv(ne),v(être),
adv(pas)]),gn([det(ma),n('déception')])]),labeled_chunked_sentence([[qu

el,ne,être,pas],[arg2(client:det(ma)),pred('déception')]]),pred_arg_str
ucture([quel,ne,être,pas,arg2(client:det(ma)),pred('déception')])],['dé
çu',[je,ai,'été','déçu',lors,de,mon,dernier,achat],sentence([pron(je),a
ux(ai),v('été'),adj('déçu'),adv(lors),prep(de),det(mon),adj(dernier),n(
achat)]),chunked_sentence([gn([pron(je)]),gv([aux(ai),v('été')]),gadj([
adj('déçu')]),gprep([adv(lors),prep(de)]),gn([det(mon),adj(dernier),n(a
chat)])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([ad
j('déçu')]),gprep([adv(lors),prep(de)]),gn([det(mon),n(achat)])]),label
ed_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')],[lor
s,de],[arg2(client:det(mon)),arg(domaine:n(achat))]]),pred_arg_structur
e([arg2(client:pron(je)),être,pred('déçu'),lors,de,arg2(client:det(mon)
),arg(domaine:n(achat))])],['déçue',[je,ai,'été','déçue',par,le,'côté',
âpre,de,cet,article],sentence([pron(je),aux(ai),v('été'),adj('déçue'),p
rep(par),det(le),n('côté'),adj(âpre),prep(de),det(cet),n(article)]),chu
nked_sentence([gn([pron(je)]),gv([aux(ai),v('été')]),gadj([adj('déçue')
]),gprep([prep(par)]),gn([det(le),n('côté'),adj(âpre),prep(de),det(cet)
,n(article)])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),ga
dj([adj('déçu')]),gprep([prep(par)]),gn([n('côté'),prep(de),n(article)]
)]),labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déç
u')],[par],[arg(domaine:n('côté')),de,arg1(entreprise:n(article))]]),pr
ed_arg_structure([arg2(client:pron(je)),être,pred('déçu'),par,arg(domai
ne:n('côté')),de,arg1(entreprise:n(article))])],['déçue',[je,me,suis,'t
rouvée','très','déçue',de,ce,produit],sentence([pron(je),pron(me),aux(s
uis),v('trouvée'),adv('très'),adj('déçue'),prep(de),det(ce),n(produit)]
),chunked_sentence([gn([pron(je)]),gv([pron(me),aux(suis),v('trouvée')]
),gadj([adv('très'),adj('déçue')]),gprep([prep(de)]),gn([det(ce),n(prod
uit)])]),clean_chunked_sentence([gn([pron(je)]),gv([pron_v(me),v_pron(t
rouver)]),gadj([adj('déçu')]),gprep([prep(de)]),gn([n(produit)])]),labe
led_chunked_sentence([[arg2(client:pron(je))],[me,trouver],[pred('déçu'
)],[de],[arg1(entreprise:n(produit))]]),pred_arg_structure([arg2(client
:pron(je)),me,trouver,pred('déçu'),de,arg1(entreprise:n(produit))])],['
déçue',[je,suis,'amèrement','déçue'],sentence([pron(je),v(suis),adv('am
èrement'),adj('déçue')]),chunked_sentence([gn([pron(je)]),gv([v(suis)])
,gadj([adv('amèrement'),adj('déçue')])]),clean_chunked_sentence([gn([pr
on(je)]),gv([v(être)]),gadj([adj('déçu')])]),labeled_chunked_sentence([
[arg2(client:pron(je))],[être],[pred('déçu')]]),pred_arg_structure([arg
2(client:pron(je)),être,pred('déçu')])],['déçue',[je,suis,'déçue',de,le
ur,remarque],sentence([pron(je),v(suis),adj('déçue'),prep(de),det(leur)
,n(remarque)]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj
('déçue')]),gprep([prep(de)]),gn([det(leur),n(remarque)])]),clean_chunk
ed_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([pr
ep(de)]),gn([n(remarque)])]),labeled_chunked_sentence([[arg2(client:pro
n(je))],[être],[pred('déçu')],[de],[arg(domaine:n(remarque))]]),pred_ar
g_structure([arg2(client:pron(je)),être,pred('déçu'),de,arg(domaine:n(r
emarque))])],['déçu',[je,suis,'déçu',de,mes,deux,derniers,achats,identi

ques],sentence([pron(je),v(suis),adj('déçu'),prep(de),det(mes),adj(deux),adj(derniers),n(achats),adj(identiques)]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçu')]),gprep([prep(de)]),gn([det(mes),adj(deux),adj(derniers),n(achats),adj(identiques)])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([prep(de)]),gn([det(mon),n(achat)])]),labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')],[de],[arg2(client:det(mon)),arg(domaine:n(achat))]]),pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),de,arg2(client:det(mon)),arg(domaine:n(achat))])],['déçue',[je,suis,'déçue',de,votre,marque,'La_Marque'],sentence([pron(je),v(suis),adj('déçue'),prep(de),det(votre),n(marque),n('La_Marque')]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gprep([prep(de)]),gn([det(votre),n(marque),n('La_Marque')])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([prep(de)]),gn([det(votre),n(marque),n('La_Marque')])]),labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')],[de],[arg1(entreprise:det(votre)),arg1(entreprise:n(marque)),arg1(entreprise:n('La_Marque'))]]),pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),de,arg1(entreprise:det(votre)),arg1(entreprise:n(marque)),arg1(entreprise:n('La_Marque'))])],['déçue',[je,suis,'déçue',par,une,tablette,de,chocolat,'Nom_du_Produit'],sentence([pron(je),v(suis),adj('déçue'),prep(par),det(une),n(tablette),prep(de),n(chocolat),n('Nom_du_Produit')]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gprep([prep(par)]),gn([det(une),n(tablette),prep(de),n(chocolat),n('Nom_du_Produit')])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([prep(par)]),gn([n(tablette),prep(de),n(chocolat),n('Nom_du_Produit')])]),labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')],[par],[arg(domaine:n(tablette)),de,arg1(entreprise:n(chocolat)),arg1(entreprise:n('Nom_du_Produit'))]]),pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),par,arg(domaine:n(tablette)),de,arg1(entreprise:n(chocolat)),arg1(entreprise:n('Nom_du_Produit'))])],['déçue',[je,suis,'déçue',pour,la,'première',fois,depuis,'25',ans],sentence([pron(je),v(suis),adj('déçue'),prep(pour),det(la),adj('première'),n(fois),prep(depuis),adj('25'),n(ans)]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gprep([prep(pour)]),gn([det(la),adj('première'),n(fois)]),gprep([prep(depuis)]),gn([adj('25'),n(ans)])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([prep(pour)]),gn([n(fois)]),gprep([prep(depuis)]),gn([n(ans)])]),labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')],[pour],[arg(domaine:n(fois))],[depuis],[arg(domaine:n(ans))]]),pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),pour,arg(domaine:n(fois)),depuis,arg(domaine:n(ans))])],['déçue',[je,suis,donc,'très','déçue',quant,à,la,éthique,de,votre,'société'],sentence([pron(je),v(suis),adv(donc),adv('très'),adj('déçue'),prep(quant),prep(à),det(la),n(éthique),prep(de),det(votre),n('société')]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadv

([adv(donc)]),gadj([adv('très'),adj('déçue')]),gprep([prep(quant),prep(
à)]),gn([det(la),n(éthique),prep(de),det(votre),n('société')])]),clean_
chunked_sentence([gn([pron(je)]),gv([v(être)]),gadv([adv(donc)]),gadj([
adj('déçu')]),gprep([prep(quant),prep(à)]),gn([n(éthique),prep(de),n('s
ociété')])]),labeled_chunked_sentence([[arg2(client:pron(je))],[être],[
donc],[pred('déçu')],[quant,à],[arg(domaine:n(éthique)),de,arg1(entrepr
ise:n('société'))]]),pred_arg_structure([arg2(client:pron(je)),être,don
c,pred('déçu'),quant,à,arg(domaine:n(éthique)),de,arg1(entreprise:n('so
ciété'))])],['déçu',[je,suis,'très','déçu',de,les,truffes,noir,'70','%'
],sentence([pron(je),v(suis),adv('très'),adj('déçu'),prep(de),det(les),
n(truffes),adj(noir),adj('70'),n('%')]),chunked_sentence([gn([pron(je)]
),gv([v(suis)]),gadj([adv('très'),adj('déçu')]),gprep([prep(de)]),gn([d
et(les),n(truffes),adj(noir)]),gn([adj('70'),n('%')])]),clean_chunked_s
entence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([prep(d
e)]),gn([n(truffe)]),gn([n(pourcent)])]),labeled_chunked_sentence([[arg
2(client:pron(je))],[être],[pred('déçu')],[de],[arg1(entreprise:n(truff
e))],[arg(domaine:n(pourcent))]]),pred_arg_structure([arg2(client:pron(
je)),être,pred('déçu'),de,arg1(entreprise:n(truffe)),arg(domaine:n(pour
cent))])],['déçue',[je,suis,'très','déçue',de,ce,produit],sentence([pro
n(je),v(suis),adv('très'),adj('déçue'),prep(de),det(ce),n(produit)]),ch
unked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('très'),adj('déç
ue')]),gprep([prep(de)]),gn([det(ce),n(produit)])]),clean_chunked_sente
nce([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([prep(de)])
,gn([n(produit)])]),labeled_chunked_sentence([[arg2(client:pron(je))],[
être],[pred('déçu')],[de],[arg1(entreprise:n(produit))]]),pred_arg_stru
cture([arg2(client:pron(je)),être,pred('déçu'),de,arg1(entreprise:n(pro
duit))])],['déçu',[je,suis,'très','déçu',par,le,menu],sentence([pron(je
),v(suis),adv('très'),adj('déçu'),prep(par),det(le),n(menu)]),chunked_s
entence([gn([pron(je)]),gv([v(suis)]),gadj([adv('très'),adj('déçu')]),g
prep([prep(par)]),gn([det(le),n(menu)])]),clean_chunked_sentence([gn([p
ron(je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([prep(par)]),gn([n(me
nu)])]),labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred(
'déçu')],[par],[arg1(entreprise:n(menu))]]),pred_arg_structure([arg2(cl
ient:pron(je)),être,pred('déçu'),par,arg1(entreprise:n(menu))])],['déçu
e',[je,suis,une,consommatrice,'très','déçue',de,votre,grande,marque],se
ntence([pron(je),v(suis),det(une),n(consommatrice),adv('très'),adj('déç
ue'),prep(de),det(votre),adj(grande),n(marque)]),chunked_sentence([gn([
pron(je)]),gv([v(suis)]),gn([det(une),n(consommatrice),adv('très'),adj(
'déçue')]),gprep([prep(de)]),gn([det(votre),adj(grande),n(marque)])]),c
lean_chunked_sentence([gn([pron(je)]),gv([v(être)]),[gn([n(consommatric
e)]),gadj([adj('déçu')])],gprep([prep(de)]),gn([det(votre),n(marque)])]
),labeled_chunked_sentence([[arg2(client:pron(je))],[être],[arg2(client
:n(consommatrice))],[pred('déçu')],[de],[arg1(entreprise:det(votre)),ar
g1(entreprise:n(marque))]]),pred_arg_structure([arg2(client:pron(je)),ê
tre,arg2(client:n(consommatrice)),pred('déçu'),de,arg1(entreprise:det(v

otre)),arg1(entreprise:n(marque))])],['déçu',[je,suis,vraiment,'déçu',de,vos,pizzas,'surgelées','Nom_du_Produit'],sentence([pron(je),v(suis),adv(vraiment),adj('déçu'),prep(de),det(vos),n(pizzas),adj('surgelées'),n('Nom_du_Produit')]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv(vraiment),adj('déçu')]),gprep([prep(de)]),gn([det(vos),n(pizzas),adj('surgelées'),n('Nom_du_Produit')])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([prep(de)]),gn([det(votre),n(pizza),n('Nom_du_Produit')])]),labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')],[de],[arg1(entreprise:det(votre)),arg1(entreprise:n(pizza)),arg1(entreprise:n('Nom_du_Produit'))]]),pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),de,arg1(entreprise:det(votre)),arg1(entreprise:n(pizza)),arg1(entreprise:n('Nom_du_Produit'))])],['déçue',[je,suis,'déçue'],sentence([pron(je),v(suis),adj('déçue')]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')])]),labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]]),pred_arg_structure([arg2(client:pron(je)),être,pred('déçu')])],['déçue',[je,suis,'très','déçue'],sentence([pron(je),v(suis),adv('très'),adj('déçue')]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('très'),adj('déçue')])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')])]),labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]]),pred_arg_structure([arg2(client:pron(je)),être,pred('déçu')])],['déçu',[je,ai,'été','très','déçu',ce,jour],sentence([pron(je),aux(ai),v('été'),adv('très'),adj('déçu'),det(ce),n(jour)]),chunked_sentence([gn([pron(je)]),gv([aux(ai),v('été')]),gadj([adv('très'),adj('déçu')]),gn([det(ce),n(jour)])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gn([n(jour)])]),labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')],[arg(domaine:n(jour))]]),pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),arg(domaine:n(jour))])],['déçue',[je,en,ai,'été','déçue'],sentence([pron(je),pron(en),aux(ai),v('été'),adj('déçue')]),chunked_sentence([gn([pron(je)]),gn([pron(en)]),gv([aux(ai),v('été')]),gadj([adj('déçue')])]),clean_chunked_sentence([gn([pron(je)]),gn([pron(en)]),gv([v(être)]),gadj([adj('déçu')])]),labeled_chunked_sentence([[arg2(client:pron(je))],[arg(domaine:pron(en))],[être],[pred('déçu')]]),pred_arg_structure([arg2(client:pron(je)),arg(domaine:pron(en)),être,pred('déçu')])],['déçue',[je,suis,'particulièrement','déçue',de,mon,dernier,achat],sentence([pron(je),v(suis),adv('particulièrement'),adj('déçue'),prep(de),det(mon),adj(dernier),n(achat)]),chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('particulièrement'),adj('déçue')]),gprep([prep(de)]),gn([det(mon),adj(dernier),n(achat)])]),clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]),gprep([prep(de)]),gn([det(mon),n(achat)])]),labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')],[de],[arg2(client:det(mon)),arg(domaine:n(achat))]]),pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),de,arg2(

client:det(mon)),arg(domaine:n(achat))])],['déçus',[nous,avons,'été','d
éçus'],sentence([pron(nous),aux(avons),v('été'),adj('déçus')]),chunked_
sentence([gn([pron(nous)]),gv([aux(avons),v('été')]),gadj([adj('déçus')
])]),clean_chunked_sentence([gn([pron(nous)]),gv([v(être)]),gadj([adj('
déçu')])]),labeled_chunked_sentence([[arg2(client:pron(nous))],[être],[
pred('déçu')]]),pred_arg_structure([arg2(client:pron(nous)),être,pred('
déçu')])]]).

```
keyword: 'déçoit'
words list: [cela,me,'déçoit']
tagged sentence: sentence([pron(cela),pron(me),v('déçoit')])
chunked sentence:
chunked_sentence([gn([pron(cela)]),gn([pron(me)]),gv([v('déçoit')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(cela)]),gn([pron(me)]),gv([v('décevoir
')])])
labeled sentence:
labeled_chunked_sentence([[arg(domaine:pron(cela))],[arg2(client:pron(m
e))],[pred('décevoir')]])
Predicate-argument structure:
pred_arg_structure([arg(domaine:pron(cela)),arg2(client:pron(me)),pred(
'décevoir')])

keyword: 'déçu'
words list: ['celle-ci',me,a,beaucoup,'déçu']
tagged sentence: sentence([pron('celle-
ci'),pron(me),aux(a),adv(beaucoup),v('déçu')])
chunked sentence: chunked_sentence([gn([pron('celle-
ci')]),gn([pron(me)]),gv([aux(a),adv(beaucoup),v('déçu')])])
chunked sentence cleaned: clean_chunked_sentence([gn([pron('celle-
ci')]),gn([pron(me)]),gv([v('décevoir')])])
labeled sentence: labeled_chunked_sentence([[arg(domaine:pron('celle-
ci'))],[arg2(client:pron(me))],[pred('décevoir')]])
Predicate-argument structure:
pred_arg_structure([arg(domaine:pron('celle-
ci')),arg2(client:pron(me)),pred('décevoir')])

keyword: 'déçoit'
words list: ['La_Marque',me,'déçoit']
tagged sentence: sentence([n('La_Marque'),pron(me),v('déçoit')])
chunked sentence:
chunked_sentence([gn([n('La_Marque')]),gn([pron(me)]),gv([v('déçoit')])
])
chunked sentence cleaned:
clean_chunked_sentence([gn([n('La_Marque')]),gn([pron(me)]),gv([v('déce
voir')])])
labeled sentence:
labeled_chunked_sentence([[arg1(entreprise:n('La_Marque'))],[arg2(clien
t:pron(me))],[pred('décevoir')]])
Predicate-argument structure:
pred_arg_structure([arg1(entreprise:n('La_Marque')),arg2(client:pron(me
)),pred('décevoir')])
```

B.21 Sortie du module 5 pred_arg_structures.log

```
keyword: 'décevant'
words list: [je,trouve,ça,'très','décevant']
tagged sentence:
sentence([pron(je),v(trouve),pron(ça),adv('très'),adj('décevant')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(trouve)]),gn([pron(ça)]),gadj([a
dv('très'),adj('décevant')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(trouver)]),gn([pron(ça)]),
gadj([adj('décevant')])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[trouver],[arg(domain
e:pron(ça))],[pred('décevant')]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),trouver,arg(domaine:pron(ça))
,pred('décevant')])

keyword: 'déception'
words list: [à,ma,grande,'déception']
tagged sentence: sentence([prep(à),det(ma),adj(grande),n('déception')])
chunked sentence:
chunked_sentence([gprep([prep(à)]),gn([det(ma),adj(grande),n('déception
')])])
chunked sentence cleaned:
clean_chunked_sentence([gprep([prep(à)]),gn([det(ma),n('déception')])])
labeled sentence:
labeled_chunked_sentence([[à],[arg2(client:det(ma)),pred('déception')]]
)
Predicate-argument structure:
pred_arg_structure([à,arg2(client:det(ma)),pred('déception')])

keyword: 'déception'
words list: [ce,courrier,pour,vous,faire,part,de,ma,'déception']
tagged sentence:
sentence([det(ce),n(courrier),prep(pour),pron(vous),v(faire),n(part),pr
ep(de),det(ma),n('déception')])
chunked sentence:
chunked_sentence([gn([det(ce),n(courrier)]),gprep([prep(pour)]),gn([pro
n(vous)]),expression([v(faire),n(part),prep(de)]),gn([det(ma),n('décept
ion')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([n(courrier)]),gprep([prep(pour)]),gn([pron(
vous)]),expression([v(faire),n(part),prep(de)]),gn([det(ma),n('déceptio
n')])])
```

```
labeled sentence:
labeled_chunked_sentence([[arg(domaine:n(courrier))],[pour],[arg1(entre
prise:pron(vous))],[faire,part,de],[arg2(client:det(ma)),pred('déceptio
n')]])
Predicate-argument structure:
pred_arg_structure([arg(domaine:n(courrier)),pour,arg1(entreprise:pron(
vous)),faire,part,de,arg2(client:det(ma)),pred('déception')])

keyword: 'déception'
words list:
[quelle,'déception','même',pas,une,'décoration',sur,les,'bûches']
tagged sentence:
sentence([adj(quelle),n('déception'),adj('même'),adv(pas),det(une),n('d
écoration'),prep(sur),det(les),n('bûches')])
chunked sentence:
chunked_sentence([gn([adj(quelle),n('déception')]),gadj([adj('même')]),
gadv([adv(pas)]),gn([det(une),n('décoration'),prep(sur),det(les),n('bûc
hes')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([n('déception')]),gadj([adj('même')]),gadv([
adv(pas)]),gn([n('décoration'),prep(sur),n('bûche')])])
labeled sentence:
labeled_chunked_sentence([[pred('déception')],['même'],[pas],[arg(domai
ne:n('décoration')),sur,arg1(entreprise:n('bûche'))]])
Predicate-argument structure:
pred_arg_structure([pred('déception'),'même',pas,arg(domaine:n('décorat
ion')),sur,arg1(entreprise:n('bûche'))])

keyword: 'déception'
words list: [je,vous,fais,part,de,ma,'déception']
tagged sentence:
sentence([pron(je),pron(vous),v(fais),n(part),prep(de),det(ma),n('décep
tion')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gn([pron(vous)]),expression([v(fais),n
(part),prep(de)]),gn([det(ma),n('déception')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gn([pron(vous)]),expression([v(f
aire),n(part),prep(de)]),gn([det(ma),n('déception')])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[arg1(entreprise:pron
(vous))],[faire,part,de],[arg2(client:det(ma)),pred('déception')]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),arg1(entreprise:pron(vous)),f
aire,part,de,arg2(client:det(ma)),pred('déception')])
```

```
keyword: 'déception'
words list: [je,vous,fais,part,de,ma,'déception','vis-à-
vis',de,le,'Nom_du_Produit']
tagged sentence:
sentence([pron(je),pron(vous),v(fais),n(part),prep(de),det(ma),n('décep
tion'),prep('vis-à-vis'),prep(de),det(le),n('Nom_du_Produit')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gn([pron(vous)]),expression([v(fais),n
(part),prep(de)]),gn([det(ma),n('déception')]),gprep([prep('vis-à-
vis'),prep(de)]),gn([det(le),n('Nom_du_Produit')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gn([pron(vous)]),expression([v(f
aire),n(part),prep(de)]),gn([det(ma),n('déception')]),gprep([prep('vis-
à-vis'),prep(de)]),gn([n('Nom_du_Produit')])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[arg1(entreprise:pron
(vous))],[faire,part,de],[arg2(client:det(ma)),pred('déception')],['vis
-à-vis',de],[arg1(entreprise:n('Nom_du_Produit'))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),arg1(entreprise:pron(vous)),f
aire,part,de,arg2(client:det(ma)),pred('déception'),'vis-à-
vis',de,arg1(entreprise:n('Nom_du_Produit'))])

keyword: 'déception'
words list: [la,'déception',de,un,enfant]
tagged sentence:
sentence([det(la),n('déception'),prep(de),det(un),n(enfant)])
chunked sentence:
chunked_sentence([gn([det(la),n('déception'),prep(de),det(un),n(enfant)
])])
chunked sentence cleaned:
clean_chunked_sentence([gn([n('déception'),prep(de),n(enfant)])])
labeled sentence:
labeled_chunked_sentence([[pred('déception'),de,arg2(client:n(enfant))]
])
Predicate-argument structure:
pred_arg_structure([pred('déception'),de,arg2(client:n(enfant))])

keyword: 'déception'
words list: [quel,ne,est,pas,ma,'déception']
tagged sentence:
sentence([adj(quel),adv(ne),v(est),adv(pas),det(ma),n('déception')])
```

```
chunked sentence:
chunked_sentence([expression([adj(quel),adv(ne),v(est),adv(pas)]),gn([d
et(ma),n('déception')])])
chunked sentence cleaned:
clean_chunked_sentence([expression([adj(quel),adv(ne),v(être),adv(pas)]
),gn([det(ma),n('déception')])])
labeled sentence:
labeled_chunked_sentence([[quel,ne,être,pas],[arg2(client:det(ma)),pred
('déception')]])
Predicate-argument structure:
pred_arg_structure([quel,ne,être,pas,arg2(client:det(ma)),pred('décepti
on')])

keyword: 'déçu'
words list: [je,ai,'été','déçu',lors,de,mon,dernier,achat]
tagged sentence:
sentence([pron(je),aux(ai),v('été'),adj('déçu'),adv(lors),prep(de),det(
mon),adj(dernier),n(achat)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([aux(ai),v('été')]),gadj([adj('déçu
')]),gprep([adv(lors),prep(de)]),gn([det(mon),adj(dernier),n(achat)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([adv(lors),prep(de)]),gn([det(mon),n(achat)])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
,[lors,de],[arg2(client:det(mon)),arg(domaine:n(achat))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),lors,de,arg
2(client:det(mon)),arg(domaine:n(achat))])

keyword: 'déçue'
words list: [je,ai,'été','déçue',par,le,'côté',âpre,de,cet,article]
tagged sentence:
sentence([pron(je),aux(ai),v('été'),adj('déçue'),prep(par),det(le),n('c
ôté'),adj(âpre),prep(de),det(cet),n(article)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([aux(ai),v('été')]),gadj([adj('déçu
e')]),gprep([prep(par)]),gn([det(le),n('côté'),adj(âpre),prep(de),det(c
et),n(article)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(par)]),gn([n('côté'),prep(de),n(article)])])
```

```
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
,[par],[arg(domaine:n('côté')),de,arg1(entreprise:n(article))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),par,arg(dom
aine:n('côté')),de,arg1(entreprise:n(article))])

keyword: 'déçue'
words list: [je,me,suis,'trouvée','très','déçue',de,ce,produit]
tagged sentence:
sentence([pron(je),pron(me),aux(suis),v('trouvée'),adv('très'),adj('déç
ue'),prep(de),det(ce),n(produit)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([pron(me),aux(suis),v('trouvée')]),
gadj([adv('très'),adj('déçue')]),gprep([prep(de)]),gn([det(ce),n(produi
t)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([pron_v(me),v_pron(trouver)])
,gadj([adj('déçu')]),gprep([prep(de)]),gn([n(produit)])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[me,trouver],[pred('d
éçu')],[de],[arg1(entreprise:n(produit))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),me,trouver,pred('déçu'),de,ar
g1(entreprise:n(produit))])

keyword: 'déçue'
words list: [je,suis,'amèrement','déçue']
tagged sentence:
sentence([pron(je),v(suis),adv('amèrement'),adj('déçue')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('amèrement'),a
dj('déçue')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
)])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu')])

keyword: 'déçue'
words list: [je,suis,'déçue',de,leur,remarque]
```

```
tagged sentence:
sentence([pron(je),v(suis),adj('déçue'),prep(de),det(leur),n(remarque)]
)
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gpr
ep([prep(de)]),gn([det(leur),n(remarque)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(de)]),gn([n(remarque)])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
,[de],[arg(domaine:n(remarque))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),de,arg(doma
ine:n(remarque))])

keyword: 'déçu'
words list: [je,suis,'déçu',de,mes,deux,derniers,achats,identiques]
tagged sentence:
sentence([pron(je),v(suis),adj('déçu'),prep(de),det(mes),adj(deux),adj(
derniers),n(achats),adj(identiques)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçu')]),gpre
p([prep(de)]),gn([det(mes),adj(deux),adj(derniers),n(achats),adj(identi
ques)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(de)]),gn([det(mon),n(achat)])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
,[de],[arg2(client:det(mon)),arg(domaine:n(achat))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),de,arg2(cli
ent:det(mon)),arg(domaine:n(achat))])

keyword: 'déçue'
words list: [je,suis,'déçue',de,votre,marque,'La_Marque']
tagged sentence:
sentence([pron(je),v(suis),adj('déçue'),prep(de),det(votre),n(marque),n
('La_Marque')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gpr
ep([prep(de)]),gn([det(votre),n(marque),n('La_Marque')])])
```

```
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(de)]),gn([det(votre),n(marque),n('La_Marque')])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
,[de],[arg1(entreprise:det(votre)),arg1(entreprise:n(marque)),arg1(entr
eprise:n('La_Marque'))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),de,arg1(ent
reprise:det(votre)),arg1(entreprise:n(marque)),arg1(entreprise:n('La_Ma
rque'))])

keyword: 'déçue'
words list:
[je,suis,'déçue',par,une,tablette,de,chocolat,'Nom_du_Produit']
tagged sentence:
sentence([pron(je),v(suis),adj('déçue'),prep(par),det(une),n(tablette),
prep(de),n(chocolat),n('Nom_du_Produit')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gpr
ep([prep(par)]),gn([det(une),n(tablette),prep(de),n(chocolat),n('Nom_du
_Produit')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(par)]),gn([n(tablette),prep(de),n(chocolat),n('Nom_du_Pro
duit')])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
,[par],[arg(domaine:n(tablette)),de,arg1(entreprise:n(chocolat)),arg1(e
ntreprise:n('Nom_du_Produit'))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),par,arg(dom
aine:n(tablette)),de,arg1(entreprise:n(chocolat)),arg1(entreprise:n('No
m_du_Produit'))])

keyword: 'déçue'
words list: [je,suis,'déçue',pour,la,'première',fois,depuis,'25',ans]
tagged sentence:
sentence([pron(je),v(suis),adj('déçue'),prep(pour),det(la),adj('premièr
e'),n(fois),prep(depuis),adj('25'),n(ans)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')]),gpr
ep([prep(pour)]),gn([det(la),adj('première'),n(fois)]),gprep([prep(depu
is)]),gn([adj('25'),n(ans)])])
```

```
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(pour)]),gn([n(fois)]),gprep([prep(depuis)]),gn([n(ans)])]
)
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
,[pour],[arg(domaine:n(fois))],[depuis],[arg(domaine:n(ans))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),pour,arg(do
maine:n(fois)),depuis,arg(domaine:n(ans))])

keyword: 'déçue'
words list:
[je,suis,donc,'très','déçue',quant,à,la,éthique,de,votre,'société']
tagged sentence:
sentence([pron(je),v(suis),adv(donc),adv('très'),adj('déçue'),prep(quan
t),prep(à),det(la),n(éthique),prep(de),det(votre),n('société')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadv([adv(donc)]),gadj([
adv('très'),adj('déçue')]),gprep([prep(quant),prep(à)]),gn([det(la),n(é
thique),prep(de),det(votre),n('société')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadv([adv(donc)]),
gadj([adj('déçu')]),gprep([prep(quant),prep(à)]),gn([n(éthique),prep(de
),n('société')])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[donc],[pred('
déçu')],[quant,à],[arg(domaine:n(éthique)),de,arg1(entreprise:n('société'))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,donc,pred('déçu'),quant,
à,arg(domaine:n(éthique)),de,arg1(entreprise:n('société'))])

keyword: 'déçu'
words list: [je,suis,'très','déçu',de,les,truffes,noir,'70','%']
tagged sentence:
sentence([pron(je),v(suis),adv('très'),adj('déçu'),prep(de),det(les),n(
truffes),adj(noir),adj('70'),n('%')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('très'),adj('d
éçu')]),gprep([prep(de)]),gn([det(les),n(truffes),adj(noir)]),gn([adj('
70'),n('%')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(de)]),gn([n(truffe)]),gn([n(pourcent)])])
```

```
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
,[de],[arg1(entreprise:n(truffe))],[arg(domaine:n(pourcent))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),de,arg1(ent
reprise:n(truffe)),arg(domaine:n(pourcent))])

keyword: 'déçue'
words list: [je,suis,'très','déçue',de,ce,produit]
tagged sentence:
sentence([pron(je),v(suis),adv('très'),adj('déçue'),prep(de),det(ce),n(
produit)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('très'),adj('d
éçue')]),gprep([prep(de)]),gn([det(ce),n(produit)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(de)]),gn([n(produit)])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
,[de],[arg1(entreprise:n(produit))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),de,arg1(ent
reprise:n(produit))])

keyword: 'déçu'
words list: [je,suis,'très','déçu',par,le,menu]
tagged sentence:
sentence([pron(je),v(suis),adv('très'),adj('déçu'),prep(par),det(le),n(
menu)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('très'),adj('d
éçu')]),gprep([prep(par)]),gn([det(le),n(menu)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(par)]),gn([n(menu)])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
,[par],[arg1(entreprise:n(menu))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),par,arg1(en
treprise:n(menu))])

keyword: 'déçue'
```

```
words list:
[je,suis,une,consommatrice,'très','déçue',de,votre,grande,marque]
tagged sentence:
sentence([pron(je),v(suis),det(une),n(consommatrice),adv('très'),adj('d
éçue'),prep(de),det(votre),adj(grande),n(marque)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gn([det(une),n(consommat
rice),adv('très'),adj('déçue')]),gprep([prep(de)]),gn([det(votre),adj(g
rande),n(marque)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),[gn([n(consommatri
ce)]),gadj([adj('déçu')])],gprep([prep(de)]),gn([det(votre),n(marque)])
])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[arg2(client:n
(consommatrice))],[pred('déçu')],[de],[arg1(entreprise:det(votre)),arg1
(entreprise:n(marque))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,arg2(client:n(consommatr
ice)),pred('déçu'),de,arg1(entreprise:det(votre)),arg1(entreprise:n(mar
que))])

keyword: 'déçu'
words list:
[je,suis,vraiment,'déçu',de,vos,pizzas,'surgelées','Nom_du_Produit']
tagged sentence:
sentence([pron(je),v(suis),adv(vraiment),adj('déçu'),prep(de),det(vos),
n(pizzas),adj('surgelées'),n('Nom_du_Produit')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv(vraiment),adj(
'déçu')]),gprep([prep(de)]),gn([det(vos),n(pizzas),adj('surgelées'),n('
Nom_du_Produit')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(de)]),gn([det(votre),n(pizza),n('Nom_du_Produit')])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
,[de],[arg1(entreprise:det(votre)),arg1(entreprise:n(pizza)),arg1(entre
prise:n('Nom_du_Produit'))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),de,arg1(ent
reprise:det(votre)),arg1(entreprise:n(pizza)),arg1(entreprise:n('Nom_du
_Produit'))])

keyword: 'déçue'
```

```
words list: [je,suis,'déçue']
tagged sentence: sentence([pron(je),v(suis),adj('déçue')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adj('déçue')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
)])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu')])

keyword: 'déçue'
words list: [je,suis,'très','déçue']
tagged sentence: sentence([pron(je),v(suis),adv('très'),adj('déçue')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('très'),adj('d
éçue')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
)])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu')])

keyword: 'déçu'
words list: [je,ai,'été','très','déçu',ce,jour]
tagged sentence:
sentence([pron(je),aux(ai),v('été'),adv('très'),adj('déçu'),det(ce),n(j
our)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([aux(ai),v('été')]),gadj([adv('très
'),adj('déçu')]),gn([det(ce),n(jour)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gn([n(jour)])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
,[arg(domaine:n(jour))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),arg(domaine
:n(jour))])
```

```
keyword: 'déçue'
words list: [je,en,ai,'été','déçue']
tagged sentence:
sentence([pron(je),pron(en),aux(ai),v('été'),adj('déçue')])
chunked sentence:
chunked_sentence([gn([pron(je)]),gn([pron(en)]),gv([aux(ai),v('été')]),
gadj([adj('déçue')])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gn([pron(en)]),gv([v(être)]),gad
j([adj('déçu')])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[arg(domaine:pron(en)
)],[être],[pred('déçu')]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),arg(domaine:pron(en)),être,pr
ed('déçu')])

keyword: 'déçue'
words list: [je,suis,'particulièrement','déçue',de,mon,dernier,achat]
tagged sentence:
sentence([pron(je),v(suis),adv('particulièrement'),adj('déçue'),prep(de
),det(mon),adj(dernier),n(achat)])
chunked sentence:
chunked_sentence([gn([pron(je)]),gv([v(suis)]),gadj([adv('particulièrem
ent'),adj('déçue')]),gprep([prep(de)]),gn([det(mon),adj(dernier),n(acha
t)])])
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(je)]),gv([v(être)]),gadj([adj('déçu')]
),gprep([prep(de)]),gn([det(mon),n(achat)])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(je))],[être],[pred('déçu')]
,[de],[arg2(client:det(mon)),arg(domaine:n(achat))]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(je)),être,pred('déçu'),de,arg2(cli
ent:det(mon)),arg(domaine:n(achat))])

keyword: 'déçus'
words list: [nous,avons,'été','déçus']
tagged sentence:
sentence([pron(nous),aux(avons),v('été'),adj('déçus')])
chunked sentence:
chunked_sentence([gn([pron(nous)]),gv([aux(avons),v('été')]),gadj([adj(
'déçus')])])
```

```
chunked sentence cleaned:
clean_chunked_sentence([gn([pron(nous)]),gv([v(être)]),gadj([adj('déçu'
)])])
labeled sentence:
labeled_chunked_sentence([[arg2(client:pron(nous))],[être],[pred('déçu'
)]])
Predicate-argument structure:
pred_arg_structure([arg2(client:pron(nous)),être,pred('déçu')])
```

B.22 Sortie du module 5 final_structures.txt

```
structure 1: arg(domaine:pron(cela)) arg2(client:pron(me))
pred(décevoir)
status: to be reviewed "undefined domaine"

structure 2: arg(domaine:pron(celle-ci)) arg2(client:pron(me))
pred(décevoir)
status: to be reviewed "undefined domaine"

structure 3: arg1(entreprise:n(La_Marque)) arg2(client:pron(me))
pred(décevoir)
status: OK

structure 4: arg2(client:pron(je)) trouver arg(domaine:pron(ça))
pred(décevant)
status: to be reviewed "undefined domaine"

structure 5: à arg2(client:det(ma)) pred(déception)
status: OK

structure 6: arg(domaine:n(courrier)) pour arg1(entreprise:pron(vous))
faire part de arg2(client:det(ma)) pred(déception)
status: to be reviewed "undefined domaine"

structure 7: pred(déception) même pas arg(domaine:n(décoration)) sur
arg1(entreprise:n(bûche))
status: to be reviewed "undefined domaine"

structure 8: arg2(client:pron(je)) arg1(entreprise:pron(vous)) faire
part de arg2(client:det(ma)) pred(déception)
status: OK

structure 9: arg2(client:pron(je)) arg1(entreprise:pron(vous)) faire
part de arg2(client:det(ma)) pred(déception) vis-à-vis de
arg1(entreprise:n(Nom_du_Produit))
status: OK

structure 10: pred(déception) de arg2(client:n(enfant))
status: OK

structure 11: quel ne être pas arg2(client:det(ma)) pred(déception)
status: OK

structure 12: arg2(client:pron(je)) être pred(déçu) lors de
arg2(client:det(mon)) arg(domaine:n(achat))
status: to be reviewed "undefined domaine"
```

```
structure 13: arg2(client:pron(je)) être pred(déçu) par
arg(domaine:n(côté)) de arg1(entreprise:n(article))
status: to be reviewed "undefined domaine"

structure 14: arg2(client:pron(je)) me trouver pred(déçu) de
arg1(entreprise:n(produit))
status: OK

structure 15: arg2(client:pron(je)) être pred(déçu)
status: OK

structure 16: arg2(client:pron(je)) être pred(déçu) de
arg(domaine:n(remarque))
status: to be reviewed "undefined domaine"

structure 17: arg2(client:pron(je)) être pred(déçu) de
arg2(client:det(mon)) arg(domaine:n(achat))
status: to be reviewed "undefined domaine"

structure 18: arg2(client:pron(je)) être pred(déçu) de
arg1(entreprise:det(votre)) arg1(entreprise:n(marque))
arg1(entreprise:n(La_Marque))
status: OK

structure 19: arg2(client:pron(je)) être pred(déçu) par
arg(domaine:n(tablette)) de arg1(entreprise:n(chocolat))
arg1(entreprise:n(Nom_du_Produit))
status: to be reviewed "undefined domaine"

structure 20: arg2(client:pron(je)) être pred(déçu) pour
arg(domaine:n(fois)) depuis arg(domaine:n(ans))
status: to be reviewed "undefined domaine"

structure 21: arg2(client:pron(je)) être donc pred(déçu) quant à
arg(domaine:n(éthique)) de arg1(entreprise:n(société))
status: to be reviewed "undefined domaine"

structure 22: arg2(client:pron(je)) être pred(déçu) de
arg1(entreprise:n(truffe)) arg(domaine:n(pourcent))
status: to be reviewed "undefined domaine"

structure 23: arg2(client:pron(je)) être pred(déçu) de
arg1(entreprise:n(produit))
status: OK
```

```
structure 24: arg2(client:pron(je)) être pred(déçu) par
arg1(entreprise:n(menu))
status: OK

structure 25: arg2(client:pron(je)) être arg2(client:n(consommatrice))
pred(déçu) de arg1(entreprise:det(votre)) arg1(entreprise:n(marque))
status: OK

structure 26: arg2(client:pron(je)) être pred(déçu) de
arg1(entreprise:det(votre)) arg1(entreprise:n(pizza))
arg1(entreprise:n(Nom_du_Produit))
status: OK

structure 27: arg2(client:pron(je)) être pred(déçu)
arg(domaine:n(jour))
status: to be reviewed "undefined domaine"

structure 28: arg2(client:pron(je)) arg(domaine:pron(en)) être
pred(déçu)
status: to be reviewed "undefined domaine"

structure 29: arg2(client:pron(nous)) être pred(déçu)
status: OK
```

Analyse Linguistique et Formalisation pour le Traitement Automatique de la Paraphrase

Résumé : Les relations paraphrastiques entre plusieurs ensembles de paraphrases peuvent se décrire en termes de suites de transformations textuelles. Pour qu'il ait paraphrase, il faut qu'une substitution lexicale noyau se mette en route entrainant d'autres modifications syntaxiques, lexicales et morphologiques.

Après avoir décrit les mécanismes de paraphrasage récurrents, nous avons proposé deux formalisations. La première est théorique et explique les différentes relations paraphrastiques entretenues par les paraphrases entre-elles. La deuxième, tournée vers des applications, formalise les structures paraphrastiques sous-forme de prédicats-arguments. Nous considérons cette dernière adaptée au traitement automatique de la paraphrase.

Nous avons à la suite implémenté un système d'extraction de structures paraphrastiques. Il s'agit d'un système opérationnel appliqué à un volume de données relevant de notre domaine d'étude, et dont le but est de donner un exemple concret d'emploi possible de notre formalisation.

Mots-clés : paraphrase, structures paraphrastiques, traitement automatique de la paraphrase, extraction des structures paraphrastiques

Linguistic Analysis and Formalisation for Paraphrase Processing

Abstract : The relations between sets of paraphrases can be described as series of textual transformations. To rephrase, an initial lexical substitution starts, then triggers other syntactic, lexical and morphological changes.

After having described the frequent paraphrasing mechanisms in our corpus, we propose two formalisations. The first one is theoretical, explaining the different paraphrasing relationships maintained by the paraphrases between each other. The second formalises paraphrase structures as predicate-argument ones. We consider the latter suitable for paraphrase processing.

Finally we have implemented a paraphrase structures extraction system. This is a compact operational system for the volume of data within our domain, the aim of which is to provide a concrete example of a possible use of our formalisation.

Keywords : paraphrase, paraphrase structures, paraphrase processing, paraphrase structures extraction