Université de Montréal

# Implémentation des collocations pour la réalisation de texte multilingue

par Florie Lambrey

Département de linguistique et traduction
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures et postdoctorales
en vue de l'obtention du grade de Maître
en linguistique
option recherche

Décembre, 2016

© Florie Lambrey, 2016

# Résumé

La génération automatique de texte (GAT) produit du texte en langue naturelle destiné aux humains à partir de données non langagières. L'objectif de la GAT est de concevoir des générateurs réutilisables d'une langue à l'autre et d'une application à l'autre. Pour ce faire, l'architecture des générateurs automatiques de texte est modulaire : on distingue entre la génération profonde qui détermine le contenu du message à exprimer et la réalisation linguistique qui génère les unités et structures linguistiques exprimant le message.

La réalisation linguistique multilingue nécessite de modéliser les principaux phénomènes linguistiques de la manière la plus générique possible. Or, les collocations représentent un de ces principaux phénomènes linguistiques et demeurent problématiques en GAT, mais aussi pour le Traitement Automatique des Langues en général. La Théorie Sens-Texte analyse les collocations comme des contraintes de sélection lexicale. Autrement dit, une collocation est composée de trois éléments : (i) la base, (ii) le collocatif, choisi en fonction de la base et (iii) d'une relation sémantico-lexicale. Il existe des relations sémantico-lexicales récurrentes et systématiques. Les fonctions lexicales modélisent ces relations. En effet, des collocations telles que *peur bleue* ou *pluie torrentielle* instancient une même relation, l'intensification, que l'on peut décrire au moyen de la fonction lexicale Magn : Magn(PEUR) = BLEUE, Magn(PLUIE) = TORRENTIELLE, etc. Il existe des centaines de fonctions lexicales.

Ce mémoire présente la méthodologie d'implémentation des collocations dans un réalisateur de texte multilingue, GÉCO, à l'aide des fonctions lexicales standard syntagmatiques simples et complexes. Le cœur de la méthodologie repose sur le regroupement des fonctions lexicales ayant un fonctionnement similaire dans des patrons génériques. Au total, plus de 26 000 fonctions lexicales ont été implémentées, représentant de ce fait une avancée considérable pour le traitement des collocations en réalisation de texte multilingue.

**Mots-clés** : génération automatique de texte ; réalisation linguistique ; collocation ; fonctions lexicales ; Théorie Sens-Texte ; traitement automatique des langues ; linguistique

# Abstract

Natural Language Generation (NLG) produces text in natural language from non-linguistic content. NLG aims at developing generators that are reusable across languages and applications. In order to do so, these systems' architecture is modular: while the deep generation module determines the content of the message to be expressed, the text realization module maps the message into its most appropriate linguistic form.

Multilingual text realization requires to model the core linguistic phenomena that one finds in language. Collocations represent one of the core linguistic phenomena that remain problematic not only in NLG, but also in Natural Language Processing in general. The Meaning-Text theory analyses collocations as constraints on lexical selection. In other words, a collocation is made up of three constituents: (i) the base, (ii) the collocate, chosen according to (iii) a semantico-lexical relation. Some of these semantico-lexical relations are systematic and shared by many collocations. Lexical functions are a system for modeling these relations. In fact, collocations such as *heavy rain* or *strong preference* instantiate the same relation, intensity, can be described with the lexical function Magn: Magn(RAIN) = HEAVY, Magn(PREFERENCE) = STRONG, etc. There are hundreds of lexical functions.

Our work presents a methodology for the implementation of collocations in a multilingual text realization engine, GÉCO, that relies on simple and complex syntagmatic standard lexical functions. The principal aspect of the methodology consists of regrouping lexical functions that show a similar behavior into generic patterns. As a result, 26 000 lexical functions have been implemented, which is a considerable progress in the treatment of collocations in multilingual text realization.

**Keywords** : natural language generation ; linguistic realisation ; collocation ; lexical functions ; Meaning-Text theory ; linguistics ; natural language processing

# Table des matières

# Liste des tableaux

# Liste des figures

# Liste des sigles

FL : fonction lexicale

RCon : représentation conceptuelle

RSem : représentation sémantique

RSyntP : représentation syntaxique profonde

RSyntS : représentation syntaxique de surface

RMorphP : représentation morphologique profonde

RMorphS : représentation morphologique de surface

UL : unité lexicale

TST : Théorie Sens-Texte

GAT : génération automatique de texte

GATM : génération automatique de texte multilingue

DEC : Dictionnaire Explicatif et Combinatoire

LEC : Lexicographie Explicative et Combinatoire

LFG : Grammaire Lexicale Fonctionnelle

TAG : Grammaire d'Arbres Adjoints

# Liste des abréviations

Adj. : Adjectif

Art. : Article

V. : Verbe

# Remerciements

Pour commencer, je veux adresser mes remerciements à mon directeur de mémoire, François Lareau, pour sa grande disponibilité, ses encouragements tout au long de la rédaction de ce mémoire et pour m'avoir engagée sur son projet GÉCO, financé par la subvention FRQSC 2016-NP-191042 et la subvention interne de l'université de Montréal-CRSH.

Je tiens également à remercier Patrick Drouin, Valérie Beaulieu et l'équipe administrative du Bureau des Étudiants Internationaux pour m'avoir aidée dans mes démarches vis-à-vis de l'immigration.

J'adresse aussi mes remerciements à Daphnée Azoulay, Marjan Alipour, Angélique Lafrance, Benoît Robichaud et l'ensemble de mes collègues étudiants de l'Observatoire Linguistique Sens-Texte pour la qualité de leur travail et leur bonne humeur à toute épreuve.

Enfin, je remercie particulièrement ma famille et Nicolas Mornard pour leur soutien absolu durant ces deux années.

# Introduction

Le présent mémoire a pour objet le traitement des collocations dans le cadre de la Génération Automatique de Texte (GAT). Cette sous-branche du Traitement Automatique des Langues (TAL) a pour objectif de produire du texte dans une langue donnée, c'est-à-dire du contenu linguistique conforme aux règles de cette langue, à partir de contenu qui n'est pas en langue naturelle, comme des données numériques brutes, des représentations logiques ou encore des informations entreposées dans des bases de données. Certains générateurs de texte sont multilingues, c'est-à-dire qu'ils génèrent du texte dans plusieurs langues.

La GAT a été créée pour répondre à plusieurs besoins. (Danlos, 1987a) spécifie que les linguistes s'en servaient au départ essentiellement pour tester la validité de différentes grammaires, mais la GAT s'intègre maintenant dans plusieurs autres types d'applications. Par exemple, (Li, Zhu, & Jin, 2015) ont recours à un module de GAT dans leur système de traduction automatique hybride. De même, il existe des cas d'intégration de générateurs automatiques de texte dans des systèmes de résumés automatiques ou des systèmes de question / réponse. Par ailleurs, plusieurs applications exploitent des générateurs créés dans le milieu académique. Enfin, certains générateurs peuvent s'employer de manière autonome. La GAT est donc une technologie utilisée dans des domaines variés.

Cependant, cette diversité d'emplois a engendré un manque d'uniformité en ce qui concerne l'architecture, les tâches et le type de données utilisées par la GAT. Jusqu'à l'apport majeur de (Reiter & Dale, 2000) sur la question, il n'existait pas de consensus sur la manière de concevoir un générateur automatique de texte. Ce problème est d'autant plus renforcé par le fait que chaque générateur est dépendant de la tâche qu'il doit accomplir. Par exemple, un générateur intégré à un système de dialogue devrait incorporer un historique de discours et être capable de produire des textes courts, ce qui n'est pas nécessairement le cas d'un générateur de descriptifs d'entreprises, par exemple.

Malgré ces divergences, la GAT découpe traditionnellement le processus de génération en deux grandes étapes (Danlos, 1987a; Reiter & Dale, 2000) : le « quoi-dire » ou détermination du contenu du message à exprimer, et le « comment-le-dire », c'est-à-dire la détermination de

la forme linguistique grammaticalement correcte la plus appropriée pour exprimer ce contenu. Cette organisation structure les générateurs de texte autour de deux modules principaux. D'une part, le module de génération profonde crée un plan de document regroupant les concepts formant l'énoncé. D'autre part, le module de réalisation linguistique, ou module de génération de surface, « traduit » le contenu du message en texte. Pour ce faire, le module de réalisation linguistique doit faire face aux phénomènes linguistiques spécifiques et aux idiosyncrasies de la langue. L'élaboration et l'utilisation de ressources, comme des dictionnaires et grammaires, permet de faciliter cette tâche. La réalisation linguistique est d'autant plus intéressante qu'elle permet d'intégrer une perspective multilingue à la GAT. En effet, quand un texte est produit dans plusieurs langues, le message du texte demeure le même, seule la forme du texte change. Autrement dit, le « quoi-dire » change peu, voire pas, et chaque langue représente un paramètre du module « comment-le-dire ». Dans le cadre de ce mémoire, nous nous focalisons sur l'aspect linguistique de la GAT. Il est donc évident que nous allons nous concentrer sur la réalisation linguistique.

Les principaux phénomènes linguistiques traités par la réalisation linguistique sont liés aux aspects syntaxiques, comme l'ordonnancement des mots, aux phénomènes morphosyntaxiques, par exemple l'accord entre le sujet et le verbe, ou encore aux difficultés morphologiques, comme la flexion verbale. Ces aspects sont traités par plusieurs réalisateurs de texte comme JSrealB (Molins & Lapalme, 2015), SimpleNLG (Vaudry & Lapalme, 2013), etc. Cependant, ces problèmes présupposent que le choix lexical a déjà été fait correctement en amont. Mais comment faire ce choix? Doit-on choisir les unités lexicales séparément? Plusieurs exemples en français laissent entendre que le choix lexical peut être contraint. Alors qu'il est possible de dire que quelqu'un est *grièvement blessé* ou *gravement blessé*, il n'est pas possible de qualifier quelqu'un de *grièvement malade*, alors que *gravement malade* fonctionne. De même, *peur bleue* ou *colère noire* sont des énoncés compréhensibles alors que *colère bleue* ou *peur noire* ne le sont pas. Ces paires de mots sont des collocations, c'est-à-dire des restrictions de sélection lexicale. Dans certains contextes, l'emploi de certaines unités lexicales est contraint par la présence d'autres unités lexicales. Dans les exemples ci-dessus, *grièvement*, *gravement*, *bleue* et *noire* sont des collocatifs, c'est-à-dire des unités lexicales dont le choix est déterminé respectivement par les lexèmes *blessé*, *malade*, *peur* et *colère*. Certaines collocations sont

récurrentes dans les langues, comme les verbes supports. Alors qu'en français, une personne peut *faire un pas*, l'équivalent anglais est *take a step* (litt. 'prendre un pas′), et l'équivalent espagnol est *dar un paso* (litt. 'donner un pas′). Le traitement des collocations présente donc non seulement un intérêt qualitatif en rendant le texte plus fluide, mais aussi un intérêt multilingue.

Malheureusement, les collocations ne sont pas traitées de manière exhaustive en réalisation linguistique. Tout d'abord, la plupart des réalisateurs de texte partent d'un niveau plus concret de représentation linguistique, où le choix lexical a déjà été effectué. Il s'agit de réalisation de surface. Dans d'autres cas, seules quelques collocations sont intégrées au réalisateur de texte à un niveau plus abstrait, généralement sémantique, voire conceptuel. Il s'agit de réalisation profonde. Dans les deux cas, faire un traitement exhaustif des collocations implique de faire face à deux sous-problèmes. Premièrement, comme le montrent les exemples plus haut, les collocations possèdent un comportement imprévisible et idiosyncratique, que ce soit dans une perspective monolingue ou multilingue. Dès lors, il est difficile de modéliser leur fonctionnement de manière à englober tous ces différents comportements dans un programme informatique. Le premier problème est donc un problème formel, ou théorique portant sur les collocations en tant qu'objet d'étude linguistique. Deuxièmement, la création et l'organisation de ressources linguistiques en réalisation de texte est une tâche fastidieuse et coûteuse, mais nécessaire lorsque l'on souhaite traiter des phénomènes linguistiques précis. Comme les collocations sont récurrentes et nombreuses, leur intégration dans ces ressources doit se faire de manière optimale afin de limiter le coût, en temps et en argent, du développement et maintien de ces ressources. Cette remarque est surtout valable dans une perspective multilingue, où le développement de ressource s'opère selon deux méthodes : le partage de ressources ou l'adaptation de ressources. Le deuxième problème est donc un problème technique, ou appliqué, portant sur les problématiques d'intégration mêmes des collocations au sein du programme informatique.

L'objectif de ce mémoire est de proposer une intégration exhaustive des collocations dans un réalisateur profond de texte multilingue, que nous appelons GÉCO (Lambrey & Lareau, 2015). À cette fin, nous répondrons aux deux problèmes liés à l'implémentation des collocations. Tout d'abord, nous fournirons une modélisation des collocations s'inspirant d'une théorie

linguistique existante, la Théorie Sens-Texte (Kahane, 2003; I. Mel′čuk, 1973, 1997). Cette théorie, en plus de fournir un modèle complet de la langue, propose le mécanisme des fonctions lexicales pour rendre compte des phénomènes de restrictions de sélection lexicale récurrents dans les langues. Par la suite, nous présenterons la méthodologie d'implémentation des collocations dans GÉCO. Comme nous le verrons, GÉCO est le successeur de MARQUIS (Lareau & Wanner, 2007; Wanner, Nicklaß, et al., 2007), un générateur automatique de texte multilingue complet dont le module de réalisation de texte est organisé selon les principes de la Théorie Sens-Texte. Tous deux suivent les principes du partage de ressources entre les différentes langues traitées. Notre stratégie d'implémentation se focalise sur l'étape de lexicalisation, c'est-à-dire l'attribution de correspondants lexicaux aux unités de sens du message à générer, pour intégrer les collocations dans la phrase.

L'organisation de ce mémoire tourne autour de quatre chapitres. Le premier fait un état de l'art sur la génération automatique de texte et se focalise donc sur la réalisation de texte et l'aspect multilingue. Nous verrons que le processus de lexicalisation est une étape clé pour la génération de collocations. Le deuxième chapitre présente une synthèse du traitement des collocations, que ce soit leur définition ou leur modélisation, en traitement automatique des langues. Par la suite, le troisième chapitre fait une présentation de GÉCO, tant d'un point de vue technique que théorique. Nous y aborderons en détail les principes méthodologiques de modélisation et d'implémentation des collocations. Le quatrième chapitre récapitule les résultats de l'implémentation et propose une analyse ainsi qu'une évaluation de ces résultats. Enfin, la conclusion présente une synthèse de ce travail.

# Chapitre 1 La génération automatique de texte

Ce chapitre présente la génération automatique de texte (GAT) et se focalise sur l'étape de réalisation linguistique. Nous commençons par faire une présentation générale de la GAT. Par la suite, nous confrontons différents réalisateurs linguistiques profonds et de surface sur plusieurs aspects. Enfin, nous élaborons sur la notion de lexicalisation avant d'expliquer pourquoi cette étape est centrale pour le traitement des collocations dans le module de réalisation linguistique.

## 1.1 Contextualisation

L'objectif d'un générateur automatique de texte est de produire des textes dans une ou plusieurs langues naturelles, à partir de contenu non linguistique, c'est-à-dire des données brutes, des représentations abstraites, etc. Ces textes doivent respecter les normes de la ou des langue(s) prise(s) en compte par le générateur. Ils doivent également répondre à un objectif communicatif précis. Autrement dit, un texte est généré pour répondre à un besoin de communication, comme la génération d'une réponse à une question ou encore la production automatique d'un bulletin météorologique. Un générateur automatique de texte est donc un programme qui transmet des informations en fonction d'un contexte communicationnel précis.

Cette définition de la GAT implique que le développement de générateurs se fait principalement dans une perspective appliquée même si (Danlos, 1987a, 1987b) spécifie que les linguistes ont développé des générateurs dans le milieu de la recherche afin de tester la validité de différentes grammaires. (Roussarie, 2000) précise que, d'un point de vue théorique, un générateur peut aussi être perçu comme une simulation de locuteur. Néanmoins, dans ce mémoire, nous nous intéressons à la GAT dans sa perspective appliquée. De même, nous ne traiterons pas des générateurs à base de patrons, comme le célèbre Eliza (Weizenbaum, 1966). En effet, (Matthiessen & Bateman, 1991, p. 3) insistent sur le caractère abstrait, non langagier, des données traitées par un générateur de texte:

> Text generation can be characterized as one mode of information processing: information that is stored at a higher level of abstraction than wordings (grammatical

structures and lexical terms) is organized and re-expressed over a number of steps so that it can be presented as a worded text.

(Reiter, 1995; Reiter & Dale, 2000; Steinlin, 2003; Van Deemter, Krahmer, & Theune, 2005) indiquent que les générateurs à base de patrons ont quant à eux un niveau de représentation concret comme point de départ. Ce sont souvent des textes à variables qu'il suffit de manipuler directement, c'est-à-dire sans passer par des niveaux de représentations intermédiaires, pour obtenir le texte final. En d'autres termes, leur fonctionnement ne se base pas sur des principes linguistiques. Nous nous focalisons sur la GAT dans une perspective linguistique dans ce mémoire et laissons de côté ces générateurs.

La GAT s'intègre dans plusieurs types d'applications. (Li et al., 2015) incluent un module de GAT dans leur système de traduction automatique hybride. De même, (Danlos, 1987a) mentionnait déjà des cas d'intégration de générateurs automatiques de texte dans des systèmes de résumés automatiques ou des systèmes de question / réponse. Par ailleurs, lors de leur participation aux séminaires RALI/OLST, Éric Charton et Félix Hervé-Bachand, employés chez Pages Jaunes, ont présenté leur application de la génération de texte dans le cadre de la production automatique de descriptifs d'entreprises. D'autres applications emploient des générateurs créés dans le milieu académique. C'est le cas de Fog (Goldberg, 1993), ou encore de MeteoVis qui s'appuie sur JSRealB (Molins, 2014; Molins & Lapalme, 2015) ou encore PATExpert employant MARQUIS (Lareau & Wanner, 2007; Wanner, Nicklaß, et al., 2007) pour n'en citer que quelques-uns. Enfin, certains générateurs, comme MARQUIS, fonctionnent de manière autonome.

Cette diversité d'emplois entraîne une grande diversité des textes générés, comme le montrent les exemples (1) à (3). Le type d'application, le sous-domaine linguistique, les ressources utilisées, la théorie linguistique sous-jacente ainsi que l'objectif communicationnel déterminent le contenu mais aussi la forme des textes générés. Par exemple :

(1) Description of the relation "is A Section Of": A section may belong to zero or more Courses. For example, S1 is a Section and belongs to the Course CS100. S2 is a Section and does not belong to any Courses. S3 is a Section and belongs to three Courses, Math100, Physics100, and Eng100.

(2) You should replace (**setq** x 1) with (**setf** x 1). **Setf** can be used to assign a value to any generalized-variable. **Setq** can only be used to assign a value to a simple-variable. A generalized-variable is a storage location that can be named by any accessor function.

(3) The air quality index is 3, which means the air quality is satisfactory. This is due to the ozone concentration. The NO2 concentration, the SO2 concentration and the PM 10 concentration do not have influence on air quality. The current air quality index (3) is the highest today. The lowest was 2 (at midnight). Between midnight and 7AM, the air quality index remained stable at 2 and between 8AM and 9AM, it remained stable at 3.

L'exemple (1) est une production du générateur MODELEXPLAINER (Lavoie & Rambow, 1997; Lavoie, Rambow, & Reiter, 1996). Il génère des descriptions de modèles de données utilisés par des logiciels orientés objets. Il fonctionne sans connaissances du domaine traité par le modèle de données, ce qui améliore sa réutilisabilité. L'exemple (2) provient du générateur PENMAN (Mann, Matthiessen, 1982; Matthiessen & Bateman, 1991; Swartout & Smoliar, 1987). Cette production a été générée dans le cadre d'un système de dialogue. Contrairement à l'exemple précédent, ce texte est beaucoup plus fluide (moins de répétitions, variation verbale, etc.) et il se base sur le cadre théorique de la Linguistique Systémique Fonctionnelle. Enfin, l'exemple (3) est généré par MARQUIS, un générateur multilingue de bulletins de qualité de l'air se basant sur la théorie Sens-texte et des ressources linguistiques riches. Ces quelques exemples illustrent la diversité des textes produits. Cette diversité est un défi pour la GAT et implique que chaque générateur est développé en fonction du type de texte à générer. Il y a donc une spécialisation des générateurs, ce qui empêche leur réutilisabilité. Cependant, la plupart des générateurs respectent la même structuration du processus de génération. C'est ce que nous allons voir à présent.

### 1.1.1 Processus de génération

Lorsque l'on construit un générateur de texte, la première question que l'on se pose est: comment se découpe et s'organise le processus de génération? Traditionnellement, la génération est perçue comme un processus séquentiel que l'on scinde en sept étapes, répertoriées ci-dessous. Les lignes suivantes sont reprises de (Reiter & Dale, 2000, p. 49):

1 – **Sélection du contenu**: Le système détermine le contenu du message à exprimer. Cette sélection s'opère en accord avec un objectif communicatif et peut être guidée par un modèle utilisateur.

2 – **Structuration du document**: La structuration du document fournit la structure discursive, et/ou rhétorique, du message en le segmentant en unités d'informations. Le résultat est un plan de document que l'on retrouve sous forme logique, en réseau ou hiérarchisée.

3 – **Lexicalisation**: La lexicalisation a pour objectif de fournir une correspondance entre les sens du message et leurs équivalents lexicaux. Ces équivalents peuvent être des lexèmes ou des constructions syntaxiques.

4 – **Génération d'expressions référentielles**: La génération d'expressions référentielles détermine la manière la plus appropriée de faire référence à une entité (cataphore et anaphore).

5 – **Agrégation**: L'agrégation regroupe les structures du plan de document en phrases et paragraphes et détermine l'ordre d'apparition des informations.

6 – **Réalisation linguistique**: Son objectif est d'appliquer des règles et procédés d'ordre grammatical aux représentations abstraites de sorte à ce que le texte de sortie soit aux normes syntaxiques et morphologiques de la langue. Il est possible d'avoir recours à des « patrons » de phrases ou des représentations syntaxiques, comme des arbres de dépendance.

7 – **Réalisation de structure**: Cette tâche s'assure que le texte respecte le format requis (balises HTML, etc.) par le module de présentation.

(Reiter & Dale, 2000) distinguent deux types de tâches : celles liées au contenu (sélection de contenu, lexicalisation, génération d'expressions référentielles et réalisation linguistique) et celles liées à la forme (structuration du contenu, agrégation et réalisation de structure). Beaucoup d'auteurs respectent ce découpage, notamment (Danlos & Roussarie, 2000; Matthiessen & Bateman, 1991; Roussarie, 2000; Steinlin, 2003) pour n'en citer que quelques-uns.

(Steinlin, 2003, p.18) insiste sur l'interdépendance de certaines tâches. Pour lui, la lexicalisation et la réalisation syntaxique fonctionnent de concert. Il précise que « les choix

syntaxiques ne semblent pas devoir être traités indépendamment des choix lexicaux. En effet, le lexique guide les constructions syntaxiques (sous-catégorisation) et, inversement, les choix syntaxiques ont un impact sur les possibilités de lexicalisation ». Dès lors, comment doit-on organiser ces tâches dans un générateur? Doivent-elles être ordonnées de manière séquentielle ou interactionnelle?

(Matthiessen & Bateman, 1991) répartissent ces tâches en deux modules principaux : le module de génération profonde et le module de génération de surface, ou réalisation linguistique. (Danlos, 1987a) utilise une autre dénomination et parle des modules « quoi-dire » et « comment-le-dire ». Le module de génération profonde s'occupe de la sélection de contenu et la structuration du document. Son objectif est de déterminer le message à exprimer mais aussi sa structuration. Le module de génération de surface traduit ce message en langue naturelle. Il intègre donc les tâches de lexicalisation, agrégation, la génération d'expressions référentielles, la réalisation linguistique et la réalisation de structure. Cette répartition bipartite organise les tâches séquentiellement. Pourtant Matthiessen et Bateman (1991, pp.11-12) reviennent sur la validité de cette structuration d'un point de vue théorique, comme le montre la citation ci-dessous. Ils préfèrent une architecture interactionnelle plutôt que séquentielle.

> Although the distinction between deep and surface can be useful for certain purposes, it has also been harmful as it has invited the interpretation that all the important choices are deep ones. This distinction is now being questioned: many 'surface' phenomena such as the selection among synonyms and the sequencing of the elements of the clause turn out to have very deep significance and have to be modelled in deep generation. (…) The general point here is that all the stages in text generation are meaningful, but they represent meaningful options at different orders of abstraction.

Dans leur ouvrage, (Reiter & Dale, 2000, p. 49) proposent un architecture tripartite du processus de génération. Le Tableau I reprend leur idée.

L'intérêt de cette architecture est la séparation rigide entre les modules en fonction des tâches qu'ils effectuent mais aussi en fonction des ressources requises. Les auteurs indiquent que le module de planification de document doit utiliser des ressources sur le domaine de l'application et laisser les choix d'ordre linguistique au module de micro-planification. Par exemple, alors que le planificateur de document décide d'intégrer l'entité 'Mars 1995' au message à l'aide de sa base de données, le micro-planificateur est en charge de l'expression

linguistique de cette entité grâce aux connaissances linguistiques : faut-il faire référence à 'Mars 1995' à l'aide d'un pronom anaphorique, une paraphrase, etc.? Les ressources employées par le module de réalisation sont essentiellement « grammaticales ».

| **Modules** | **Tâches liées au contenu** | **Tâches liées à la forme** |
|---|---|---|
| Planificateur de document | Détermination du contenu | Structuration du contenu |
| Micro-planificateur | Lexicalisation | Agrégation |
| | Génération d'expressions référentielles | |
| Réalisateur | Réalisation linguistique | Réalisation de structure |

Tableau I Architecture tripartite d'un générateur automatique de texte

Cette architecture est consensuelle. (Steinlin, 2003, p. 20) propose une synthèse de ces deux approches à l'aide de la Figure 1 :

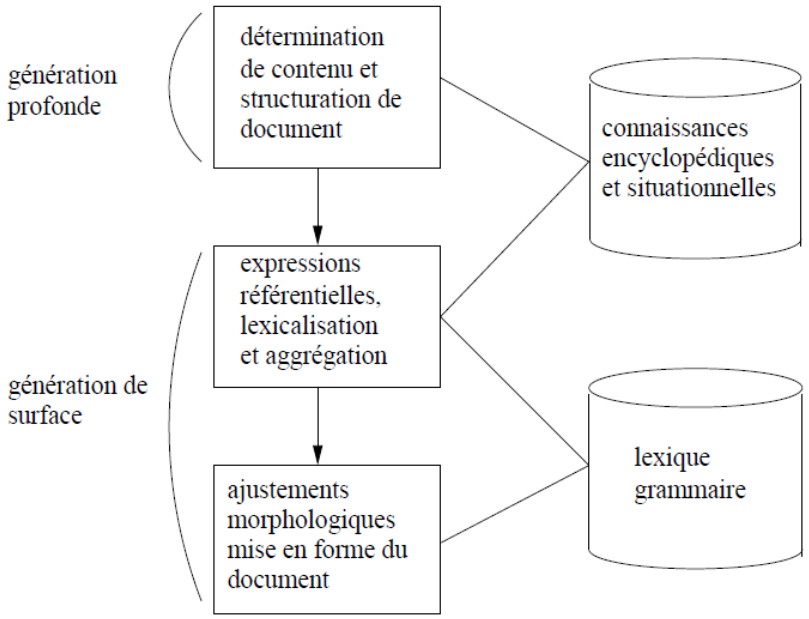



Figure 1. Architecture tripartite (Steinlin, 2003, p. 20)

Ici, la micro-planification est intégrée dans le module de réalisation linguistique. Dans les deux cas, l'étape de sélection du contenu est séparée de l'étape de réalisation de ce contenu. La

différence majeure de ces divisions revient à se poser la question suivante : À quel niveau d'abstraction débute la réalisation linguistique? Faut-il intégrer la lexicalisation dans le module de réalisation linguistique? Enfin, quel impact cela peut-il avoir sur l'intégration des collocations en GAT? Pour répondre à ces questions, nous allons aborder l'aspect multilingue en GAT.

### 1.1.2 Aspect multilingue

La génération automatique de texte multilingue vise à produire du texte dans plusieurs langues naturelles à l'aide d'un générateur unique. Dans cette sous-section, nous répondrons à deux questions : Comment intégrer une perspective multilingue dans un générateur automatique de texte? Quels sont les enjeux majeurs liés à la GAT multilingue?

(Reiter & Dale, 2000, p. 55) incorporent la perspective multilingue dans le module de micro-planification, en particulier lors de l'étape de lexicalisation:

> Lexicalization is hardest in multilingual systems, where the same information must be expressed in different languages.

Cette approche est corroborée par (Steinlin, 2003) qui envisage la GAT multilingue (GATM) comme la création d'un module de génération profonde unique sur lequel s'appuient plusieurs modules de génération de surface, chacun adapté à une langue. Autrement dit, la GATM est perçue comme un enjeu lié à la génération de surface. Selon ces auteurs, un texte généré dans plusieurs langues transmet le même contenu, seule sa réalisation linguistique diffère. Cette perspective revient à faire une analogie entre le multilinguisme et le paraphrasage: générer du texte dans plusieurs langues signifie générer des paraphrases inter-langagières. La lexicalisation semble donc être l'étape qui permet d'introduire l'aspect multilingue dans un système de GAT.

La création de générateurs automatiques de texte multilingues n'est pas une tâche aisée. L'enjeu principal repose sur le développement et le maintien de ressources linguistiques. En effet, développer des grammaires et dictionnaires monolingues pour chaque langue traitée par le générateur est une solution coûteuse tant en temps qu'en main d'œuvre. De même, le risque d'une telle approche est la redondance de règles, comme l'indiquent (Reiter & Dale, 2000, p. 55):

> Of course one could simply build completely independent lexicalization systems for each target language, but this is wasteful; for example, the rule 'avoid long enumerated lists of days unless there are pragmatic reasons to use this construction' probably applies to many other languages in addition to English. Ideally lexicalization rules should be shared as much as possible across languages, but our understanding of how to do this is still very incomplete.

En réalité, il existe deux approches principales liées à la gestion de ressources multilingues en TAL : le partage de grammaires et l'adaptation de grammaires.

D'après (Santaholma, 2008), l'adaptation de grammaires est une technique visant à la modification d'une grammaire existante afin de couvrir une nouvelle langue dans une application donnée. Par exemple, (Kim et al., 2003) présentent leur adaptation de la grammaire ParGram japonaise (Masuichi & Ohkuma, 2003) vers le coréen. Les grammaires ParGram sont encodées suivant le formalisme LFG (Bresnan, 2001; Dalrymple, 2006) et offrent une couverture large des langues traitées (anglais, français, allemand, japonais, norvégien et urdu). Chaque grammaire est constituée de règles de constructions de structures syntaxiques (c-structures) et leurs matrices de traits équivalentes (f-structures), représentant également des structures syntaxiques. (Kim et al., 2003) précisent que ce projet a permis de mettre en lumière le parallélisme des f-structures entre plusieurs langues typologiquement similaires. Au bout de seulement deux mois, l'équipe a créé le cœur de la grammaire du coréen en important des règles telles quelles (ordonnancement des mots, etc.), en en retravaillant certaines (négation phrastique) et en en créant de toute pièce (double accusatif). De même, leur méthode a permis de développer semi-automatiquement le lexique du coréen. Au final, le temps et l'effort liés au développement de la grammaire du coréen ont été considérablement optimisés. Malgré tout, l'adaptation de ressource présente un inconvénient de taille, selon (Kim et al., 2003) : elle n'est possible qu'entre des langues typologiquement similaires.

(Santaholma, 2008) fournit également une définition du partage de ressources. Il s'agit de créer une grammaire dont les règles, ou un ensemble de règles, sont directement partagées entre les langues. Comparativement aux ressources monolingues, ou à l'adaptation de ressources, le partage permet de limiter la taille du code car les règles centrales ne sont écrites qu'une seule fois. Cette approche rejoint le point de vue de (Reiter & Dale, 2000) car elle évite la redondance entre les règles et facilite donc le maintien de la grammaire. Le système de

(Santaholma, 2008) repose sur une grammaire partagée de 80 règles créée dans le cadre de la grammaire Regulus (Rayner, Hockey, & Bouillon, 2006), dont 43 sont applicables sur les langues du système (anglais, finnois, japonais et grec). Ce noyau est composé de règles génériques et paramétrées. Ces règles gèrent les fondations communes de différents phénomènes linguistiques comme l'ordonnancement des mots, l'accord, etc. Elles font appel à des macros, ou fonctions, permettant d'insérer les paramètres correspondants à chaque langue. En plus de ce noyau, il existe des règles plus spécifiques, ne s'appliquant qu'à trois, voire deux ou bien même une seule langue. Seul 25 % des règles sont spécifiques à une langue donnée. Grâce à cette organisation, l'intégration du grec s'est fait en seulement deux semaines. De même, cette approche intègre des langues typologiquement diverses. Alors que le système de (Santaholma, 2008) propose le partage d'une grammaire structurale, (J. A. Bateman, 1997; J. A. Bateman, Kruijff-Korbayová, & Kruijff, 2005; J. Bateman, Matthiessen, Nanri, & Zeng, 1991) insistent sur l'importance du caractère fonctionnel d'une telle grammaire. Leur approche, instanciée dans le générateur KPML, se base sur la notion de similarité et divergence fonctionnelle, (J. Bateman et al., 1991, p. 966) :

> Our basic assumption is that commonality across languages is functional in the first instance, not structural or realizational: i.e., functionality has to be preserved across languages, but structural realizations may well differ.

Les langues accomplissent différentes fonctions. C'est en représentant ces fonctions dans une grammaire qu'il est possible d'atteindre un niveau de description linguistique hautement détaillé et commun entre les langues. La réalisation de ces fonctions est ensuite décrite dans la grammaire à l'aide de contraintes sur les langues. Nous reviendrons plus en détail sur le fonctionnement de KPML dans la section 1.2.2.2. D'autres exemples de partage de ressources existent, notamment (Avgustinova & Uszkoreit, 2000; Boguslavsky, Iomdin, & Sizov, 2004; Lareau & Wanner, 2007). Dans les faits, l'idée derrière le partage de ressources est donc de fournir une grammaire suffisamment abstraite pour rendre compte des constructions de plusieurs langues. Ces principes abstraits récurrents sont ensuite réalisés, ou exprimés, différemment en fonction d'une langue spécifique.

En résumé, l'aspect multilingue représente un enjeu à traiter en génération de surface. Or, afin d'optimiser le traitement de plusieurs langues, il est plus avantageux de recourir à des

ressources faisant une description suffisamment abstraite des phénomènes linguistiques récurrents. La lexicalisation joue un grand rôle dans la gestion de plusieurs langues. Il se pourrait que cette étape puisse servir également pour le traitement des collocations. Nous allons maintenant aborder la GAT d'un point de vue pratique et comparer plusieurs réalisateurs de texte.

## 1.2 Réalisation linguistique

Le terme « réalisation de surface » est ambigu. Il fait référence d'une part aux engins s'attaquant à la tâche de réalisation linguistique telle qu'elle est décrite par (Reiter & Dale, 2000), comme Jsreal et SimpleNLG, et d'autre part aux engins englobant l'ensemble des tâches du module de réalisation linguistique. C'est le cas de KPML, Marquis, G-TAG, etc. Ces réalisateurs partent d'un niveau plus abstrait, et font donc de la réalisation profonde. Par mesure de clarté, nous utiliserons respectivement les dénominations « réalisateur de surface » et « réalisateur profond » pour faire référence à chaque type d'engin.

La présente section aborde en premier deux réalisateurs de surface. Par la suite, nous présentons plusieurs réalisateurs profonds. Enfin, nous comparons ces approches et expliquons le choix du type d'engin que nous utiliserons pour le travail de ce mémoire.

### 1.2.1 Réalisateurs de surface

Dans cette sous-section nous présentons deux réalisateurs de surface et leurs équivalents bilingues: JSreal/JSrealB et SimpleNLG/SimpleNLG-EnFr.

#### 1.2.1.1 JSreal et JSrealB

JSreal (Daoust, 2014; Daoust & Lapalme, 2015) est un réalisateur de texte pour le français, écrit en JavaScript et destiné à la programmation web. Il peut s'employer seul ou en combinaison.

JSreal prend en entrée des spécifications d'arbres syntaxiques en constituants immédiats écrites en fonctions JavaScript (JS). Les mots, syntagmes, etc. sont les « unités » manipulées. Les unités de base sont les catégories grammaticales (nom, verbe, etc.). Chacune déclenche l'application d'une fonction prenant un lemme comme argument et retournant un objet JS avec

les propriétés et méthodes correspondants. Les syntagmes sont des unités déclenchant des fonctions prenant d'autres unités comme arguments. Les lemmes et leurs caractéristiques (traits) sont répertoriés dans un dictionnaire à part. L'application des fonctions permet de construire une structure de données sous forme d'arbre regroupant toutes les propriétés des lemmes. Cet arbre est ensuite parcouru et fournit la liste des tokens de la phrase finale.

Il existe une contrepartie bilingue français-anglais à JSreal, JSrealB (Molins, 2014; Molins & Lapalme, 2015). Il s'appuie sur un dictionnaire spécifique pour chaque langue. Chaque entrée spécifie ses caractéristiques, comme sa partie du discours, son genre, etc. Les caractéristiques fournissent donc des informations sur la réalisation morphosyntaxique et morphologique des entrées en pointant vers la bonne terminaison dans des tables de flexion. Chaque langue possède ses tables de flexion spécifiques.

Par ailleurs, l'architecture de JSrealB est très différente de JSreal. En effet, le cœur de son fonctionnement repose sur la codification des règles de transformation des éléments terminaux sous forme de tables de flexion. Ce mécanisme d'appariement de tables de flexion est standardisé pour faciliter l'implémentation de nouvelles langues flexionnelles dans JSrealB. Son processus de réalisation linguistique prend en entrée une représentation syntaxique en constituants immédiats spécifiant les unités lexicales de la phrase, leur regroupement en syntagme et l'ordre des mots. Une fois la fonction grammaticale de chaque élément déterminée, ce réalisateur fléchit les éléments de la phrase un à un selon un processus de propagation des caractéristiques à l'aide des règles de flexion. (Molins, 2014) ne précise pas le processus computationnel liant les éléments terminaux à leur table de flexion. La réalisation s'opère en cascade: une fois le sujet grammatical réalisé, les caractéristiques sont propagées sur le syntagme verbal, ce qui permet de l'accorder correctement.

Au final, JSrealB prend en charge des phénomènes morphosyntaxiques et morphologiques de base (conjugaison de temps simples, accord en genre et nombre, etc.). Cependant, l'auteur ne mentionne pas comment traiter d'autres phénomènes fréquents comme la réalisation d'une question, l'inclusion d’éléments adverbiaux de phrase, la gestion de l'élision, etc. Néanmoins, l'approche par table de flexion est fort intéressante parce qu’elle peut être rapprochée de certains courants de morphologie flexionnelle. Je renvoie le lecteur intéressé à l'article de (Bonami & Boyé, 2010) particulièrement.

#### 1.2.1.2 SimpleNLG et SimpleNLG-EnFr

SimpleNLG (Gatt & Reiter, 2009) est un réalisateur de texte sous forme d'une librairie Java et propose une interface de visualisation et contrôle du processus de réalisation. Il décrit des types lexicaux et phrastiques et comment les combiner.

Le processus de réalisation se fait en quatre étapes : (i) initialisation des constituants avec les unités lexicales correspondantes, (ii) détermination des traits lexicaux, (iii) combinaison des constituants en structures et (iv) linéarisation de la structure résultante. SimpleNLG prend en entrée des constituants simples ou des morceaux de phrases préfabriquées (*canned text*). Le processus de réalisation revient à attribuer des traits, morphologiques et syntaxiques, aux constituants en entrée. Les opérations (ii) et (iii) se basent respectivement sur un module lexical et un module phrastique. Le module lexical définit un dictionnaire, les règles morphologiques et des classes d'unités lexicales (nom, verbe, etc.). Ce module prend en charge la flexion mais aussi d'autres phénomènes comme le positionnement des adjectifs par rapport au nom. Le module syntaxique définit la composition en syntagme des éléments de la phrase ainsi que les phénomènes morphosyntaxiques (accord en genre et en nombre par exemple). SimpleNLG est donc comme son nom l'indique un réalisateur très simple d'emploi et facile à adapter.

SimpleNLG-EnFR est une extension bilingue à Simple NLG, (Vaudry & Lapalme, 2013). Il possède la même couverture grammaticale que SimpleNLG et se base sur le français fondamental (1$^{er}$ degré) (Ministère de l'Éducation Nationale, 1959) pour élaborer sa couverture grammaticale française. Son dictionnaire français est composé à partir de l'échelle Dubois-Buyse d'orthographe usuelle française (Ters, Mayers, & Reichenbach, 1969), contenant les mots les plus fréquents de la langue. Beaucoup de l'architecture de SimpleNLG a été gardée pour le français, mais quelques règles grammaticales ont dû être créées.

### 1.2.2 Réalisateurs profonds

Dans cette sous-section, nous présentons des réalisateurs profonds, c'est-à-dire des programmes dont le fonctionnement couvre toutes les tâches prévues dans le module de réalisation linguistique : la lexicalisation, la génération d'expressions référentielles, l'agrégation, la réalisation linguistique et la réalisation de structure. Comme le remarquent

(Essers & Dale, 1998, p. 11) : « The more abstract nature of the input representation appears to require a greater commitment to the underlying theory ». Les réalisateurs profonds présentés ici partent d'un niveau plus abstrait que les réalisateurs de surface. Tous se basent sur une théorie linguistique. Nous comparons quatre réalisateurs profonds : FUF, KPML, TAG (GENI, RTGen, G-TAG) et MARQUIS.

#### 1.2.2.1 FuF/Surge

FUF (Elhadad, 1993; Elhadad & Robin, 1996; Essers & Dale, 1998) est un formalisme pour la génération de surface se basant sur les principes des grammaires d'unification fonctionnelle. Son fonctionnement se base sur l'unification de graphes pour combiner une structure d'entrée (spécification de phrase) avec une grammaire de la langue de sortie. Le résultat est une structure spécifiée syntaxiquement qui est par la suite linéarisée pour produire la phrase finale. L'une des grammaires développée pour FUF est Surge (Elhadad & Robin, 1996), une grammaire de l'anglais respectant également les principes des grammaires d'unification fonctionnelle.

FUF prend en entrée des descriptions fonctionnelles (FD) décrivant le sens d'une phrase et une grammaire (aussi décrites en descriptions fonctionnelles). Ces descriptions fonctionnelles apparaissent sous la forme de matrices attributs-valeurs enchâssées. Le format de sortie est une phrase en anglais exprimant le sens voulu et suivant les normes grammaticales décrites par la grammaire, (Elhadad, 1993). FUF n'intègre pas de ressources dictionnairiques pour guider le processus de génération de surface car toute l'information est encodée dans les descriptions fonctionnelles de Surge (Essers & Dale, 1998).

Le processus de réalisation profonde s'effectue en deux étapes. La première consiste à unifier les descriptions fonctionnelles du sens de la phrase à générer, c'est-à-dire enrichir la structure d'entrée, avec les spécifications syntaxiques et morphosyntaxiques de la grammaire. Ces spécifications permettent de gérer l'ordonnancement des mots, les constructions syntaxiques, l'accord, etc. Une fois l'unification terminée et la structure d'entrée enrichie, FUF linéarise la structure pour obtenir une phrase et prend en charge les contraintes morphologiques.

À priori, FUF n'est pas multilingue. Intégrer une nouvelle langue à ce générateur revient à développer une nouvelle grammaire correspondant aux normes de cette nouvelle langue. De même, FUF n'intègre aucune collocation.

#### 1.2.2.2 KPML/Nigel

KPML (Komet-Penman Multilingual) est une extension multilingue du système PENMAN (J. A. Bateman et al., 2005; J. Bateman et al., 1991; Mann et al., 1982). Il aborde le processus de génération de surface d'un point de vue fonctionnel : chaque choix linguistique est motivé par l'accomplissement d'une fonction. Cette perspective fonctionnaliste alimente l'aspect multilingue de KPML : les différentes langues partagent un certain nombre de fonctions. Seule la réalisation linguistique des fonctions diffère d'une langue à l'autre (J. A. Bateman, 1997). Ce principe est au cœur du cadre théorique sur lequel repose KPML : la grammaire fonctionnelle systémique.

La grammaire de KPML est encodée comme un réseau de système orienté (lisible de gauche à droite). Chaque intersection représente un choix grammatical minimal entre différents traits fonctionnels. Plus on avance dans le graphe, plus les choix deviennent spécifiques et précis. Cette architecture en réseau couvre tous les aspects linguistiques (phonologie, morphologie, syntaxe, etc.). La Figure 2 représente un fragment de réseau de huit systèmes (J. A. Bateman, 1997, p. 12).

Le générateur part d'une structure d'entrée abstraite au format SPL (Kasper, 1989). Il traverse ses réseaux et produit un ensemble de déclarations de réalisation qui caractérisent le texte à générer, ces déclarations s'instanciant par des contraintes sur la forme grammaticale de la phrase. Ces contraintes sont spécifiées sous forme de matrices de traits grammaticaux. KPML arrive donc à généraliser beaucoup de phénomènes linguistiques grâce à ces réseaux. KPML s'utilise avec Nigel, une grammaire de l'anglais. (Essers & Dale, 1998) ajoutent que KPML nécessite une description à part des unités lexicales sémantiquement pleines, en plus de la structure en entrée. Ces unités lexicales doivent être associées à un type sémantique, que l'on retrouve dans la structure SPL.

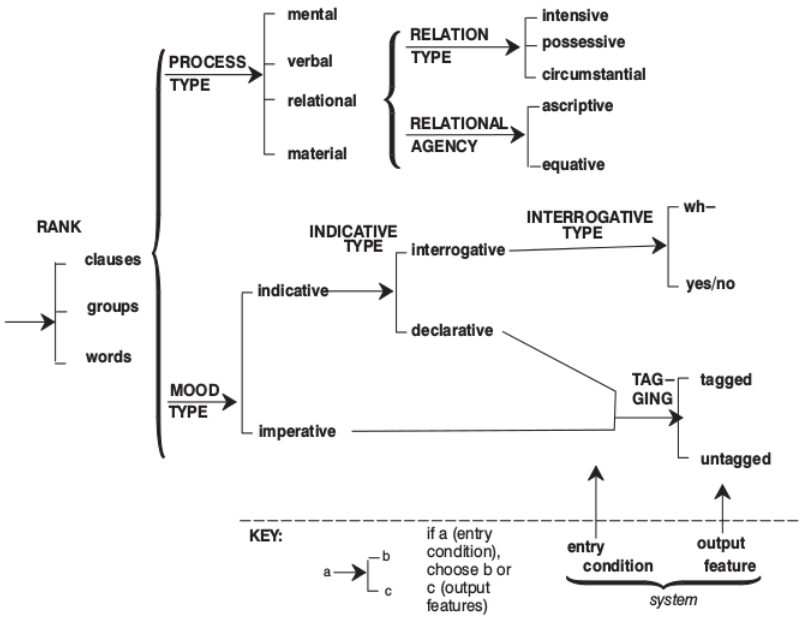



Figure 2. Réseau de systèmes sous KPML (Bateman, 1997, p. 12)

KPML est un réalisateur profond attrayant, mais ne fournit pas de détails quant au traitement des collocations et la possibilité de les encoder dans les réseaux de systèmes.

#### 1.2.2.3 Générateurs basés sur TAG

Dans cette sous-section, nous présentons plusieurs générateurs de surface se basant sur des variantes du formalisme TAG (Abeillé, 1993; Danlos, 1998; Joshi & Vijay-Shanker, 2001). (Abeillé, 1993) et (Steinlin, 2003) font une présentation générale de TAG, que nous allons reprendre ici.

TAG représente la phrase à l'aide d'arbres élémentaires se combinant. Chaque arbre élémentaire correspond à une unique entité sémantique. Pour être bien formés, ces arbres élémentaires doivent posséder au moins une tête lexicale non vide. De même, l'arbre élémentaire correspondant à un prédicat doit contenir un nœud pour chacun des arguments qu'il sous-catégorise. Deux types d'arbres élémentaires existent : les arbres initiaux et les arbres auxiliaires. Les arbres initiaux arborent des nœuds de substitution alors que les arbres auxiliaires possèdent des nœuds « pieds ». Il existe deux opérations pour combiner les arbres entre eux : la substitution et l'adjonction. La substitution insère un arbre élémentaire ou dérivé dans un arbre initial et unifie les propriétés du nœud de l'arbre initial où la substitution s'opère

avec celles du nœud racine de l'arbre élémentaire inséré. L'adjonction consiste à insérer un arbre auxiliaire dans un autre arbre à un nœud spécifique. La combinaison des arbres élémentaires résulte en un arbre dérivé représentant une phrase complète. Le formalisme prévoit également la création d'un arbre de dérivation annexe représentant la séquence de combinaisons et les arbres élémentaires requis lors de la dérivation. Lorsque les arbres élémentaires spécifient explicitement les unités lexicales, on parle d'arbres élémentaires lexicalisés. TAG part donc d'une représentation sémantique et en dérive l'arbre syntaxique et les éléments lexicaux correspondants. Cela lui permet de faire de la génération non déterministe, c'est-à-dire que plusieurs arbres dérivés sont créés pour une même représentation sémantique.

Le premier générateur de surface que nous présentons est GENI (Gardent & Kow, 2007; Gardent & Perez-Beltrachini, 2010). Il se base sur la variante FLTAG (*Features Lexicalised Tree Adjoining Grammar*). Ce formalisme guide le processus de dérivation à l'aide de l'unification de traits (TOP et BOTTOM). GENI exploite une grammaire de l'anglais, SemXTAG, où chaque arbre élémentaire s'associe à une représentation sémantique (*flat semantics*) où les arguments manquants sont des variables d'unification. Les opérations de substitution et d'adjonction instancient ces variables pour créer un arbre dérivé complètement lexicalisé. Leur grammaire est compilée à partir d'une métagrammaire constituée d'arbres élémentaires définis par la combinaison d'un ou plusieurs fragments d'arbres et chaque fragment d'arbre encapsule une caractéristique linguistique spécifique. Elle contient 1300 arbres élémentaires couvrant plusieurs phénomènes linguistiques (copules, topicalisation, adjoints, passifs, etc.). La Figure 3 illustre le fonctionnement de SemXTAG sur une représentation sémantique de l'anglais et est reprise de (Gardent & Kow, 2007, p. 3).

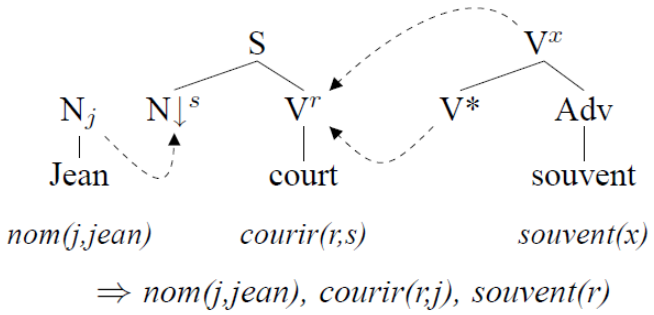



Figure 3. Exemple d'unification dans GenI

Ce générateur de surface utilise leur grammaire de manière réversible : GENI est donc capable de faire de l'analyse de texte en plus de la génération. Le processus de génération de surface de GENI se divise en quatre étapes : la sélection lexicale (choix des arbres élémentaires), la substitution (toutes les substitutions possibles), l'adjonction (toutes les adjonctions possibles) et l'extraction des chaînes textuelles. Dans la même veine, le générateur RTGen (Gardent & Perez-Beltrachini, 2010) se base également sur la grammaire SemXTAG. Son objectif est d'améliorer les performances techniques du processus de génération, mais nous ne rentrerons pas plus dans les détails. Les auteurs ne spécifient pas si GENI et RTGen sont capables de générer des collocations. De même, leur grammaire ne couvre que l'anglais.

Le deuxième générateur de surface présenté ici est décrit dans Steinlin (2003) et se base sur G-TAG (Danlos, 1998). Ce formalisme est spécifiquement fait pour la GAT et a été instancié dans FLAUBERT (Meunier, 1997). Contrairement à GenI et RTGen, G-TAG prend en entrée une représentation logique basée sur le formalisme Login (Aït-Kaci & Nasr, 1986) instanciant deux types de concepts : les entités (*thing*) et relations (*relation*). G-TAG utilise des arbres de G-dérivation, plus riches linguistiquement que les arbres de dérivations. Les nœuds sont étiquetés par des entrées lexicales accompagnées de traits morphologiques et syntaxiques. De même, les arcs spécifient les rôles thématiques. Le processus de génération de surface se fait en deux étapes. Premièrement, chaque concept de la structure de départ est associé à un ensemble d'arbres de G-dérivations sous-spécifiés contenus dans une base lexicale. Chacun de ces arbres est identifié par une entrée lexicale et porte des traits indiquant le type de construction qu'ils peuvent générer. Lorsqu'il s'agit d'un concept prédicatif (une relation), les feuilles arborent des nœuds variables correspondant aux positions argumentales du concept. Un algorithme de lexicalisation choisit l'arbre le plus approprié et lexicalisent les arguments du concept en instanciant les nœuds variables. La deuxième étape consiste à créer l'arbre dérivé à partir de l'arbre de G-dérivation. Pour ce faire, les unités lexicales de l'arbre de G-dérivation sont associées à des arbres élémentaires dans une grammaire TAG. Cette grammaire fournit les fonctions syntaxiques des arguments de l'arbre de G-dérivation. Enfin, l'arbre dérivé est traité par le module de post-traitement pour former le texte final. G-TAG est un formalisme partant d'une structure conceptuelle, donc très abstraite.

Dans une optique multilingue, il faudrait développer deux ressources : (i) la base conceptuelle associant les concepts aux arbres de G-dérivation lexicalisé et (ii) la grammaire TAG associant chaque unité lexicale avec un ensemble d'arbres élémentaires. Il en va de même pour GenI et RTGen. Ces formalismes semblent converger avec le principe du partage de grammaire. Malheureusement, à notre connaissance, aucun travail ne porte là-dessus. De même, les ouvrages sur les générateurs de surface utilisant TAG ne mentionnent pas les collocations à l'exception de (Steinlin, 2003). Dans son mémoire, l'auteur présente une méthodologie d'intégration de quelques collocations à l'aide des fonctions lexicales de la Théorie Sens-Texte (I. Mel′čuk, 1997). Cet outil permet de décomposer une collocation en deux sous-éléments : la base, choisie librement, et le collocatif, choisi en fonction de la base. Nous approfondirons la notion de fonction lexicale dans le Chapitre 2. Les quelques collocations ainsi générées permettent d'insérer un verbe support, une préposition ou un modificateur adjectival ayant un sens d'intensification. Nous reviendrons en détail sur le travail de (Steinlin, 2003) dans le Chapitre 2.

#### 1.2.2.4 MARQUIS

MARQUIS (Lareau & Wanner, 2007; Wanner, Bohnet, et al., 2007; Wanner, Nicklaß, et al., 2007; Wanner, Bohnet, Bouayad-Agha, Lareau, & Nicklaß, 2010) est un générateur complet de bulletins météorologiques de qualité de l'air. Autrement dit, il effectue à la fois la génération profonde et la génération de surface. MARQUIS est multilingue et prend en charge l'anglais, l'allemand, l'espagnol, le catalan, le portugais, le français, le finnois et le polonais. Voici une brève présentation de ce générateur qui sera poursuivie au Chapitre 3 du présent mémoire.

Le module de génération profonde produit un plan de document sous forme de graphe conceptuel à la (Sowa, 2000). Le module de réalisation linguistique de MARQUIS est structuré selon les principes de la Théorie Sens-Texte (I. Mel′čuk, 1973, 1997). Brièvement, cette théorie propose une modélisation fonctionnelle de la langue s'articulant en une série de correspondances entre les sens d'un côté et leurs équivalents textuels ou phonologiques de l'autre. MARQUIS part donc de la représentation conceptuelle du message et la dérive sur plusieurs niveaux intermédiaires de représentation linguistique pour obtenir du texte en bout

de ligne, voir Figure 4. Le passage d'un niveau à l'autre se fait par un ensemble de règles de transduction, composant la grammaire de la langue.

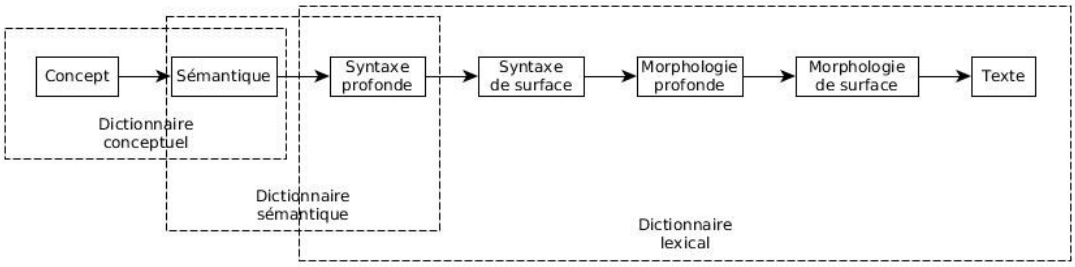



Figure 4. Réalisation profonde de MARQUIS et dictionnaires associés

La structure conceptuelle d'entrée est universelle, c'est-à-dire qu'elle est partagée par toutes les langues du système. MARQUIS en dérive les structures sémantiques équivalentes dans chaque langue. Le reste du processus de génération devient donc spécifique aux langues. Malgré tout, MARQUIS suit les principes du partage de ressources : les règles de transduction (la grammaire) appliquées entre chaque niveau de représentation linguistique sont suffisamment génériques pour être partagées le plus possibles entre les langues. Les règles de transduction récupèrent l'information pertinente dans des dictionnaires riches pour assurer la grammaticalité des textes générés.

Tout comme G-TAG, MARQUIS génère plusieurs collocations à l'aide des fonctions lexicales. Il en intègre une trentaine au total, alors que (Steinlin, 2003) en implémente trois. Les fonctions lexicales sont intégrées à la représentation syntaxique profonde de la phrase. Nous reviendrons sur le mécanisme exact dans le Chapitre 3.

Le processus de lexicalisation de MARQUIS s'effectue à plusieurs endroits. Tout d'abord, le passage entre la structure conceptuelle et la structure sémantique choisit les unités de sens représentant le message à transmettre pour chaque langue. Il s'agit de la phase de sémanticisation, ou lexicalisation profonde. En effet, ces choix ont un impact sur le choix final des unités lexicales. Ensuite, l'interface sémantique-syntaxe profonde permet d'intégrer toutes les unités lexicales sémantiquement pleines (verbes, noms, adjectifs et adverbes). Enfin, la transition entre le niveau de syntaxe profonde et le niveau de syntaxe de surface insère les unités lexicales vides de sens (prépositions, articles, etc.). Étendre la tâche de lexicalisation autorise un meilleur contrôle des choix lexicaux et marque bien l'interdépendance entre le

lexique et la syntaxe. MARQUIS est donc intéressant sur plusieurs points: (i) son développement est ancré dans une perspective multilingue et (ii) il fait déjà un traitement non exhaustif des collocations.

### 1.2.3 Comparaison

Les réalisateurs de surface ont l'avantage d'être faciles à mettre en place et encodés de manière standard. En comparaison, les réalisateurs profonds requièrent beaucoup d'experts en linguistique, tant pour leur développement que leur utilisation. Ils sont plus complexes, structurellement parlant et ont un coût en temps et en main d'œuvre élevé. Cependant, ils permettent de rendre compte de phénomènes linguistiques plus variés et sont linguistiquement robustes. En d'autres termes, le choix de passer par un réalisateur de surface ou un réalisateur profond revient à trouver un équilibre entre couverture linguistique ou puissance de réalisation d'une part et complexité d'autre part. Dans le cadre de ce mémoire, nous allons utiliser un réalisateur profond car nous cherchons à implémenter les collocations à l'aide d'une modélisation linguistique de ce phénomène.

Les réalisateurs profonds décrits ici passent tous par une approche symbolique : les connaissances linguistiques sont explicitées au moyen de ressources, en l'occurrence des grammaires et dictionnaires. La plupart abordent les phénomènes linguistiques sous un angle fonctionnel. De même, la plupart passent par l'unification de graphes. Cependant, ils se distinguent vis-à-vis de leur répartition des connaissances linguistiques : FUF et les générateurs basés sur TAG encodent la grammaire au moyen d'arbres plus ou moins lexicalisés. Ils ne requièrent donc pas de dictionnaires à part entière. Au contraire, KPML et MARQUIS dissocient grammaires et dictionnaires. L'avantage de cette méthode est la modélisation de comportements linguistiques applicables à plusieurs langues. Ces réalisateurs profonds partagent certains aspects de leur grammaire sur plusieurs langues. Il n'est pas étonnant que ces deux générateurs de surface soient multilingues. Au contraire, les approches associant directement le lexique aux graphes/arbres, impliquent de développer une grammaire pour chaque langue ou de trouver des mécanismes de portage de grammaire d'une langue à l'autre.

Par ailleurs, contrairement à KPML, Marquis intègre déjà quelques collocations à l'aide des fonctions lexicales de la Théorie Sens-Texte. C'est également le cas du travail de (Steinlin, 2003). Cependant, MARQUIS base tout son processus de réalisation profonde sur la Théorie Sens-Texte, et évite donc de mélanger deux théories linguistiques au sein d'un même programme.

Dans le cadre de ce mémoire, nous allons nous baser sur GÉCO, une extension de MARQUIS. GÉCO est un projet en cours de développement dirigé par François Lareau à l'Observatoire de linguistique Sens-Texte. Il reprend l'architecture de MARQUIS et vise à y implémenter de manière exhaustive les collocations.

## 1.3 Retour sur la lexicalisation

L'objectif de ce mémoire est de faire un traitement exhaustif des collocations en GATM. Nous allons donc nous inspirer de l'aspect multilingue en GAT et chercher à modéliser les collocations à un niveau suffisamment abstrait pour rendre compte de leur diversité d'emploi. Ce niveau est celui où s'opère la lexicalisation. En effet, nous avons énoncé dans ce chapitre les différentes étapes constituant le processus de génération, ainsi que différents générateurs de surface. Cet exposé a permis de mettre en avant l'importance de la tâche de lexicalisation, en particulier pour l'implémentation des collocations.

Premièrement, il existe une interdépendance entre les choix lexicaux et les choix de constructions syntaxiques. Cela s'illustre tout à fait dans le choix de la partie du discours. Par exemple, le sens 'amour' peut se lexicaliser de trois façons possibles, soit le nom AMOUR, le verbe AIMER et AMOUREUX, qui peut s'utiliser dans la construction ÊTRE AMOUREUX, où ÊTRE n'est qu'une coquille syntaxique assurant la bonne formation de la phrase (dans ce cas précis, ÊTRE AMOUREUX forme une collocation). Le choix d'un lexème d'une partie du discours ou d'une autre aura nécessairement des répercussions sur les positions qu'il pourra occuper dans la phrase.

Par ailleurs, (Stede, 1996) précise qu'il existe cinq configurations de correspondances entre concept et lexème, repris par (Steinlin, 2003) : (i) le choix lexical correspond à une relation univoque entre le concept et le lexème, (ii) le concept correspond à plusieurs lexèmes

entretenant de ce fait des relations de synonymie (ou quasi-synonymie), (iii) le concept se lexicalise par le lexème correspondant ou un hyponyme de celui-ci, (iv) plusieurs concepts se lexicalisent par la même lexie (homonymie lexicale) et (v) le concept ne possède pas de correspondant lexical dans la langue.

Ces difficultés amènent au deuxième point : la lexicalisation, en raison de sa complexité, ne doit pas forcément être intégrée à un seul endroit du processus. Dans le cas de G-TAG, la lexicalisation s'opère sur deux niveaux : lors de la construction de l'arbre de G-dérivation et lors de la composition de l'arbre dérivé. De même, dans MARQUIS, la lexicalisation s'opère sur trois niveaux. Répartir le processus de lexicalisation sur plusieurs niveaux est également une idée présentée par (Polguère, 1998, 2000).

Dans la suite de ce mémoire, nous allons donc nous concentrer sur le réalisateur profond GÉCO pour intégrer les collocations. Nous reprenons l'architecture de MARQUIS, ainsi que sa mise en place du processus de lexicalisation. Avant de passer à la phase pratique, il convient de définir avec précision la notion de collocation et de proposer un outil pour leur modélisation. C'est ce que nous allons voir tout de suite.

# Chapitre 2 Les collocations

Dans ce chapitre, nous présentons les collocations. Nous contextualisons ce concept et en confrontons plusieurs définitions. Nous verrons que le mécanisme des fonctions lexicales de la Théorie Sens-Texte (Kahane, 2003; I. Mel′čuk, 1973, 1997; I. Mel′čuk, 1988) est un outil très utile pour les modéliser. Après avoir fait une typologie des différents types de fonctions lexicales dans la deuxième section, nous présenterons quelques générateurs automatiques de texte proposant une implémentation des fonctions lexicales.

## 2.1 Définitions

Les collocations, un sous-ensemble d'unités polylexicales (*multiword expressions*), ont beaucoup préoccupé la littérature linguistique (Benson, 1990; Firth, 1968; Nérima, Seretan, & Wehrli, 2006; Sag, Baldwin, Bond, Copestake, & Flickinger, 2002; Sinclair, 1991) en particulier depuis l'apparition de la linguistique de corpus et du TAL, même si le premier exemple cité de ce phénomène revient à (Bally, 1909) :

(4) grièvement blessé vs. gravement malade

Les premières définitions font état de l'aspect empirique des collocations. Par exemple, (Cruse, 1986) les identifie comme des unités lexicales qui apparaissent souvent ensemble. (Benson, 1990) les traite comme des combinaisons arbitraires récurrentes et (Sinclair, 1991) les définit comme l'occurrence de deux ou plusieurs mots à proximité l'un de l'autre dans le texte, pour ne citer que ces auteurs. À la base, les collocations sont donc définies principalement en termes de fréquence d'occurrence préférentielle : ce sont des combinaisons de « mots » apparaissant ensemble fréquemment, comme le montrent les exemples (4) à (7).

(5) faire un pas vs. *poser un pas

(6) gros fumeur vs. *grand fumeur

(7) pluie de critiques vs. *pluie d'applaudissements

La linguistique a cherché à comprendre ce phénomène et beaucoup d'encre a coulé sur le sujet des collocations et des expressions polylexicales de manière générale. Beaucoup de définitions ont ainsi vu le jour, notamment :

> (Sag et al., 2002, p. 2) : les collocations sont « des interprétations idiosyncrasiques qui dépassent la limite du mot-clé »
>
> (Manning & Schütze, 1999, p. 151) : les collocations sont des « expressions qui correspondent à une manière conventionnelle de parler »
>
> (Fontenelle, 1992, p. 222) : « le terme collocation se réfère à la combinaison syntagmatique des items lexicaux et elle est indépendante de la catégorie des mots ou de la structure syntaxique »
>
> (Van Der Wouden, 1992, p. 449) : « nous utilisons le terme COLLOCATION comme terme générique pour décrire les restrictions idiosyncrasiques sur la distribution des lexèmes »

Malgré cette foison définitoire, il n'existe pas de définition rigoureuse consensuelle de ce qu'est une collocation. De même, les collocations présentent un intérêt non négligeable pour la GAT. En effet, le choix des éléments d'une collocation n'est pas libre. Ces contraintes sur le choix lexical doivent impérativement figurer dans un bon générateur automatique de texte afin de produire du texte de qualité. Pourtant, il n'est pas si simple d'intégrer les collocations dans un système de GAT. En effet, deux enjeux principaux sont à mentionner en ce qui concerne les collocations.

Tout d'abord, comme le dit (I. Mel′čuk, 2013), le nombre de collocations dans une langue donnée est très élevé. (Sag et al., 2002) indiquent que les expressions polylexicales (dont les collocations font partie) forment 41% des entrées de la version 1.7 de Wordnet (Fellbaum, 1998). Pour (Jackendoff, 1997), il en existerait autant que de mots individuels. Dès lors, comment rendre compte de l'ensemble de ces phénomènes d'un point de vue linguistique mais aussi appliqué, voire lexicographique? Si l'on veut que les logiciels de GAT fassent un traitement approfondi des collocations, il faut pouvoir représenter l'ensemble des collocations dans les ressources. D'ailleurs, plusieurs dictionnaires de collocations à la large couverture existent comme le COBUILD (Sinclair, 1995) et le Dictionary of English Collocations (Kjellmer, 1994), pouvant servir en GAT. Cependant, les collocations sont souvent simplement listées comme des entrées à part entière. Il ne s'agit donc que d'un traitement de

surface. Par ailleurs, développer et maintenir une telle liste de collocations n'est pas une solution viable pour un générateur.

En plus de cette omniprésence, les collocations sont arbitraires, ce qui se reflète par un fort taux de variabilité. Par « arbitraire », nous entendons qu'il n'est pas possible de substituer l'un des éléments d'une collocation au risque de former une collocation fautive, ou moins appropriée:

(8) *crack a joke* (fr. 'faire une blague') vs. **make a joke*

(9) **crack an effort* vs *make an effort* (fr. 'faire un effort')

Les exemples (8) et (9) montrent non seulement que les collocations sont récurrentes à travers les langues mais que le degré d'arbitraire est lui-même récurrent inter-linguistiquement. Alors que le français utilise le même verbe support *faire* pour les deux collocations, l'anglais emploie deux verbes supports différents et non interchangeables, *crack* et *make*. Cet arbitraire est au cœur des difficultés de définition, d'analyse et d'implémentation des collocations. D'ailleurs, les collocations englobent plus de phénomènes que l'insertion des verbes supports:

(10) *heavy smoker* (fr. 'gros fumeur')

(11) carré de chocolat

Dans l'exemple (10), les unités lexicales de la collocation forment une paire Adjectif – Nom mais forment la paire Nom – Nom (en plus du mot outil) dans l'exemple (11). Comme l'indiquent (Nérima et al., 2006), les collocations peuvent se manifester dans toutes les combinaisons syntaxiques possibles. Il n'existe donc pas un schéma syntaxique spécifique permettant de repérer une collocation.

(I. Mel′čuk, 2013, p. 1) résume ces problèmes :

> En même temps les phrasèmes représentent un défi incontournable pour toutes les théories « purement » formalistes : syntaxiquement réguliers, ils ne peuvent pourtant pas être « générés »; par conséquent, on doit chercher une bonne façon de les décrire.

Comme lui, plusieurs auteurs (McKeown & Radev, 2000; Sag et al., 2002; Van Der Wouden, 1992) insistent donc sur la nécessité de fournir une description rigoureuse des collocations afin de pouvoir les classer, mais aussi de généraliser leur comportement. En effet, définir précisément ce qu'est une collocation revient à fournir un ensemble de critères permettant de

les identifier et les classifier. Nous allons confronter trois définitions du terme « collocation » et justifier pourquoi une approche fonctionnelle de ce phénomène à l'aide des fonctions lexicales est primordiale pour un traitement qualitatif des collocations en GAT.

### 2.1.1 Fréquence et contiguïté

Pour (McKeown & Radev, 2000), les « mots » (on suppose qu'ils parlent de lexèmes), se combinent de trois manières différentes dans les langues. Tout d'abord, il est possible de les combiner librement pour former une expression libre. Ces combinaisons suivent les règles de la syntaxe et de la grammaire et chaque mot de l'expression peut être remplacé librement par un synonyme sans altérer le sens de l'expression. Par opposition, les expressions idiomatiques sont des combinaisons rigides de mots. Autrement dit, il est difficile d'appliquer des processus syntaxiques sur ces expressions, comme la modification adjectivale, ou la construction passive. De même, les idiomes présentent un sens non compositionnel, c'est-à-dire que le sens général de l'expression n'est pas équivalent à la somme des sens de ses composants. Il n'existe pas non plus de règles générales de construction de ces idiomes : leur composition syntaxique est arbitraire. Les auteurs proposent un continuum sur la possibilité combinatoire des lexèmes avec les choix libres d'un côté et les expressions idiomatiques de l'autre. Les collocations représentent l'entre deux de ce continuum, comme le montre le Tableau II, (McKeown & Radev, 2000, p. 511) :

| Idioms | Collocations | Free Word Combinations |
|---|---|---|
| To kick the bucket | To trade actively | To take the bus |
| Dead end | Table of content | The end of the road |
| To catch up | Orthogonal projection | To buy a house |

Tableau II Continuum de possibilités combinatoires selon (McKeown & Radev 2000, p. 511)

Autrement dit, les collocations sont des combinaisons de mots qui apparaissent ensemble plus souvent qu'attendu, dont le sens est compositionnel (chaque élément de la collocation contribue au sens général de cette dernière) mais dont la forme et le sens ne suivent pas les règles de construction syntaxique ou sémantique. Il s'agit d'un phénomène d'ordre lexical.

(McKeown & Radev, 2000) emploient donc un critère essentiellement fréquentiel pour définir ce qu'est une collocation. De même, la contiguïté semble constituer un élément important de la définition.

Pourtant, cette définition n'est pas suffisante car les limites entre choix libres de mots, collocations et expressions idiomatiques sont floues. De même, les auteurs n'appliquent pas leur définition sur les combinaisons de mots qui incluent un élément lexical récurrent. De ce fait, des collocations comme *take a step*, *go crazy* ou *avoir faim* ne sont pas prises en compte en raison de l'abondance des lexèmes *take*, *go* et *avoir* dans la langue. D'ailleurs, les auteurs spécifient explicitement au sujet de *go* (p. 513):

> Generally, collocations are those word pairs which occur frequently together in the same environment, but do not include lexical items which have a high overall frequency in language. The latter include words such as go, know, etc. which can combine with just about any other words.

Cela se reflète dans leur classification bipartite des collocations. Premièrement, il existe des collocations grammaticales, telles que *come to*, *put on*, *by accident*, etc. Elles sont composées d'une base, l'élément lexical, et d'un *collocator*, l'élément grammatical. Deuxièmement, il existe des collocations sémantiques, c'est-à-dire des combinaisons de mots restreints lexicalement. Pour un contexte donné, le *collocator* ne peut pas, ou difficilement, être substitué par un synonyme. Techniquement, rien ne distingue ces deux classes de collocation. Dans les deux cas, le choix d'un élément est conditionné par l'autre, seule la partie du discours change.

La définition de (McKeown & Radev, 2000) est donc incomplète : la fréquence et la contiguïté ne sont pas des critères définitoires suffisants pour représenter rigoureusement le concept de collocation.

### 2.1.2 Figement et aspect catégoriel

(Nérima et al., 2006) font une synthèse des différentes approches liées aux collocations. Ils reprennent entre autres les idées de (Benson, 1990; Cruse, 1986; Manning & Schütze, 1999; Sag et al., 2002) et d'autres. C'est pourquoi nous avons intégré leur travail dans notre analyse. Ils commencent par distinguer les choix de lexèmes faits librement des

expressions polylexicales. Les expressions polylexicales sont définies comme « des unités lexicales constituées par plusieurs mots orthographiques, non nécessairement contigus ». Contrairement à (McKeown & Radev, 2000), ici les unités polylexicales forment un tout, une seule unité. De ce fait, la fréquence et la contiguïté jouent un moins grand rôle dans cette définition.

Leur classification se base sur les propriétés catégorielles et le degré de figement syntaxique et sémantique. Il existe selon les auteurs trois types d'unités polylexicales.

Tout d'abord, les mots composés, de catégorie lexicale, sont des expressions dont les composants sont contigus mais dont la signification n'est pas nécessairement compositionnelle. Cela inclut des unités lexicales comme *pomme de terre* mais aussi d'autres comme *c'est-à-dire* par exemple. Les mots composés forment des unités indivisibles syntaxiquement (**pomme très de terre*, **c'est-à-vraiment-dire*) mais leur sens n'est pas toujours compositionnel. En effet, la notion de *pomme de terre* n'est pas construite à partir de celle de *pomme* ni de celle de *terre*. Par contre, un *sèche-linge* sert bien à sécher le linge.

Le deuxième type d'expressions polylexicales selon (Nérima et al., 2006) représente des unités de catégorie syntaxique. Il s'agit des expressions idiomatiques comme *poser un lapin à quelqu'un* et *casser sa pipe*. Ces idiomes sont flexibles syntaxiquement (les verbes se conjuguent) dans une certaine mesure (**casser ses pipes*). Les exemples fournis montrent bien le caractère non compositionnel de ces expressions.

Enfin, les collocations forment le dernier type d'unités polylexicales d'après (Nérima et al., 2006). Ce sont aussi des unités syntaxiques, dans le sens où ce sont « des associations conventionnelles de mots, arbitraires et récurrentes ». Autrement dit, elles forment des syntagmes. Les composants d'une collocation ne sont pas nécessairement contigus. Leur signification est compositionnelle, ce qui explique pourquoi il est possible de remplacer l'un des deux termes, même si cela rend la collocation moins appropriée. Voici quelques exemples de collocations pour (Nérima et al., 2006) : *gros fumeur*, *caresser l'espoir*, *exercer une profession*. Les collocations sont beaucoup plus fluides que les expressions idiomatiques syntaxiquement, ce qui rend la définition floue.

(Nérima et al., 2006) se servent donc de critères catégoriels, en plus des scores de fréquences et de la notion de contiguïté pour définir ce qu'est une collocation. De même, ils intègrent à cette description le concept de figement tant d'un point de vue sémantique que syntaxique. Cependant, ils insistent sur le fait que malgré tous ces critères, les collocations restent encore difficiles à extraire, l'extraction des collocations étant leur objectif. Encore une fois, une définition « structurelle » de ce qu'est une collocation ne semble pas suffisante, en particulier dans le contexte de la GAT.

### 2.1.3 Définition fonctionnelle

Nous présentons maintenant le point de vue de la Théorie Sens-Texte (I. Mel′čuk, 1997) sur la question des collocations. Nous allons essentiellement nous baser sur la publication de (I. Mel′čuk, 2013) sur le sujet car elle représente l'aboutissement de longues années de travail, notamment (I. Mel′čuk, 1998; I. A. Mel′čuk, 1988; I. Mel′čuk, Arbatchewsky-Jumarie, & Clas, 1999; I. Mel′čuk, Clas, & Polguère, s. d.; Žolkovskij & Mel'čuk, 1967). Mel′čuk emploie le terme de phrasème pour désigner tout énoncé multilexémique non libre, c'est-à-dire une « configuration de deux ou plusieurs lexèmes syntaxiquement liés ». Autrement dit, dans un phrasème, le choix d'une unité lexicale est déterminé par le choix d'une autre unité lexicale. Par opposition, un énoncé multilexémique libre est composé d'unités lexicales sélectionnées indépendamment les unes des autres par le locuteur. Il existe évidemment un nombre infini d'énoncés multilexémiques libres.

Le cœur de la définition de Mel′čuk repose donc sur la notion de contrainte, ou de restriction de choix lexical. D'ailleurs, il indique explicitement qu'il préfère la notion de contrainte à celle de figement car ce terme fait référence à plusieurs phénomènes (figement d'un point de vue sémantique, syntaxique ou fréquentiel). Mel′čuk affirme également qu'il existe beaucoup de phrasèmes dans les langues. Ils peuvent être morphologiques ou syntaxiques.

À partir de là, il fournit deux axes de classification des phrasèmes. L'axe paradigmatique concerne les contraintes de sélection lexicale ou sémantico-lexicale des composants alors que l'axe syntagmatique renvoie aux contraintes de combinaison et de composition des composants. Ces deux axes constituent une typologie des phrasèmes, reprise dans le Tableau III (I. Mel′čuk, 2013). L'axe paradigmatique permet de distinguer les phrasèmes lexicaux des

phrasèmes sémantico-lexicaux. Un phrasème lexical (*porter son attention sur*, *au bout du rouleau*) porte un sens ‘s’ construit librement mais le choix des unités lexicales servant à exprimer ce sens ‘s’ est contraint. Un phrasème sémantico-lexical possède un sens ‘s’ qui n’est pas sélectionné librement et dont le choix des lexèmes est aussi contraint (*sans imprévu*). L'axe syntagmatique distingue entre les phrasèmes sémantiquement compositionnels et non compositionnels. Un phrasème sémantiquement compositionnel possède un sens ‘s’ dénotant une union linguistique entre ses composants. Par exemple, l'énoncé *une pomme d’Espagne* est une expression compositionnelle car le sens de l'énoncé résulte de l'union des sens de ses composants. À l'inverse, lorsque le sens ‘s’ du phrasème n'est pas égal à l'union linguistique de ses composants, alors il est jugé non compositionnel. C'est le cas de la locution *château en Espagne*, qui n'a rien à voir avec le sens ‘château en Espagne’ mais plutôt ‘projet chimérique’. Ces deux axes de classifications mettent au jour trois types de phrasèmes.

| Nature des contraintes | Compositionnalité des phrasèmes | Non compositionnels | compositionnels |
|---|---|---|---|
| lexicales | | LOCUTIONS | COLLOCATIONS |
| Sémantico-lexicales | | impossible | CLICHÉS |

Tableau III Typologie des phrasèmes par (I. Mel′čuk, 2013)

Les locutions sont des phrasèmes lexicaux sémantiquement non compositionnels. Ce sont des expressions idiomatiques. En fonction du degré d’inclusion du sens des composants, il est possible de sous-diviser cette partie en trois sous-types : les locutions fortes, les semi-locutions et les locutions faibles. Tout d’abord, une locution forte est complètement non compositionnelle, comme la locution *château en Espagne*. Deuxièmement, une semi-locution est plus compositionnelle et inclut des exemples comme *loup de mer* ou *casser les oreilles*. Dans ce cas-ci, la semi-locution récupère le sens d'une seule de ses composantes et inclut un sens additionnel comme pivot sémantique (argument sémantique du prédicat). Dans le cas de *loup de mer*, la semi-locution récupère le sens ‘mer’, n’exploite pas le sens ‘loup’ et ajoute le sens ‘homme’ qui sert de pivot sémantique. Enfin, une locution faible est compositionnelle : elle inclut tous les sens de ses composants, et rajoute un sens extérieur. Par exemple, la

locution-faible *rouge à lèvres* a comme sens ‘un produit coloré rouge pour les lèvres’ où le sens ‘produit’ est rajouté. Cette dernière catégorie de locution paraît peu fiable : un rouge à lèvres n’a pas nécessairement la couleur rouge.

Les clichés sont des phrasèmes sémantico-lexicaux compositionnels. Le choix du sens d’un cliché est contraint par le message conceptuel à transmettre. De même, le choix lexical est contraint. Ils ne constituent pas des unités lexicales. Mel′čuk donne les exemples suivants de clichés. Nous n’approfondirons pas ce point.

(12) sauf imprévu

(13) Quel âge avez-vous?

Enfin, les collocations dénotent des phrasèmes lexicaux semi-compostionnels. Autrement dit, il s’agit d'une paire de mots dont le choix d'un élément lexical est contraint par le choix de l'autre élément lexical (fait librement) en fonction d’un sens à exprimer. (I. Mel′čuk, 1996, p. 37) exprimait déjà ce principe de restriction de sélection lexicale:

> In the process of text production, the speaker, whether a human or a computer, is faced, among other difficulties, with the problem of lexical choices: to go from a given Semantic Representation (SemR) to a corresponding Deep-Syntactic Representation (DSyntR), the speaker has to select lexical units (LUs), i.e., lexemes and/or phrasemes, that he will use to build his sentences.

Elles sont décrites en Théorie Sens-Texte à l’aide des fonctions lexicales (FL) (Jousse, 2003, 2010; Kahane, 2003; Kahane & Polguère, 2001; I. Mel′čuk, 1973, 1996, 1998; I. A. Mel′čuk, 1988; Žolkovskij & Mel’čuk, 1967). En voici une définition canonique (I. Mel′čuk, 2013, p.3):

> une FL **f** décrit la relation sémantico-lexicale entre la base d'une collocation et son collocatif : **f**(base) = {collocatif}.

L’idée principale derrière les fonctions lexicales est qu’il est possible de systématiser la relation sémantico-lexicale liant les deux éléments d'une collocation à l’aide d'une fonction qui prend en argument le composant choisi librement (la base) et qui fournit comme valeur l’ensemble des composants exprimant la relation sémantico-lexicale (le(s) collocatif(s)). Faisons une démonstration par l’exemple.

(14) Magn(brouillard) = dense

(15) Magn(malade) = gravement

(16) Magn(dormir) = « comme un loir »

Dans le cas des exemples (14) à (16), il existe une relation d'intensification entre les unités lexicales *brouillard*, *malade* et *dormir* respectivement avec *dense*, *gravement* et *comme un loir*. La relation sémantico-lexicale d'intensification est représentée par la FL Magn. Autrement dit, si l'on applique Magn sur *brouillard*, elle retournera la valeur *dense*. Il en va de même pour les autres exemples. Cette relation est systématique, et donc représentable par un ratio analogique:

$$\frac{brouillard}{dense} = \frac{malade}{gravement} = \frac{dormir}{comme\ un\ loir}$$

Dans chaque cas on cherche à exprimer l'intensification. Il existe donc une relation sémantico-lexicale entre ces paires d'unités lexicales, présentement une relation d'intensification. Par ailleurs, alors que les lexèmes *brouillard*, *malade* et *dormir* sont choisis librement, ce n'est pas le cas de *dense*, *gravement* et *comme un loir*. Par exemple, **brouillard comme un loir* n'est pas un énoncé valide du français. Le fait d'exprimer une relation sémantico-lexicale particulière, ici l'intensification, est ce qui contraint le choix lexical ce qui résulte en une collocation.

Les FL permettent de décrire les collocations de manière systématique et fonctionnelle, retirant de ce fait le flou existant autour de ce terme. Cet aspect fonctionnel a également l'avantage d'être implémentable dans un système de GAT, et de TAL de manière générale. (Van Der Wouden, 1992, p. 451) en faisait déjà l'éloge:

> These functions are impressively successful in covering a large number of candidate collocations. […] What they offer is not JUST a notation for collocations. In effect, they imply that there are a relatively restricted number of central collocation types (currently around 15) and hence the collocations a given word enters into can be more-or-less exhaustively typed.

Nous nous servirons donc des fonctions lexicales pour modéliser les collocations dans le générateur de texte GÉCO.

## 2.2 Présentation des fonctions lexicales

Les FL ont été développées dans le cadre de la linguistique Sens-Texte (Kahane, 2003; I. Mel′čuk, 1997; I. A. Mel′čuk, 1988; Žolkovskij & Mel’čuk, 1967). Cette section a pour but

de familiariser le lecteur sur les différents types existants de FL : combien de FL existent et comment sont-elles classifiées ? Quel formalisme doit-on utiliser pour les représenter ? Enfin, comment sont-elles implémentées dans des générateurs de texte?

### 2.2.1 Typologie des fonctions lexicales

Il existe plusieurs axes de classification des fonctions lexicales.

#### 2.2.1.1 Axe standard/non standard

La première classification qu'il est possible de faire repose sur le caractère standard ou non d'une FL et a été abordé à maintes reprises (Jousse, 2003, 2010; I. Mel′čuk, 1996; I. Mel′čuk et al., 1999; Polguère, 2007). (I. Mel′čuk et al., 1995, p. 127-128) fournissent quatre critères qui permettent de juger du caractère standard d'une fonction lexicale :

1. Pour toute paire de lexies L1 et L2, les lexies **f**(L1) et **f**(L2) montrent des relations sémantico-syntaxiques (presque) identiques à ces lexies.
2. En règle générale, **f**(L1) et **f**(L2) sont différents : f(L1) ≠ f(L2)
3. La fonction **f** a un nombre élevé d'arguments (= de mots-clés). [En d'autres mots, le sens '**f**' est très abstrait et très général et s'applique à beaucoup d'autres sens]
4. La fonction **f** a un nombre élevé d'éléments dans sa valeur (= d'expressions)

Magn, telle qu'instanciée dans les exemples de (14) à (16), correspond donc tout à fait à une FL standard car elle exprime une relation sémantico-lexicale analogue entre chacun des mots-clés (critère 1), ceux-ci étant différents les uns des autres (critère 2), variés (critère 3) et les valeurs retournées sont elles aussi diverses (critère 4).

Dans le même ouvrage, Mel′čuk et ses collègues insistent bien sur le fait qu'il existe des collocations imprévisibles, au sens très spécifique et non généralisable. Autrement dit, la relation sémantico-lexicale entre les éléments de la collocation est si spécifique qu'elle ne s'applique qu'à un petit ensemble de mots-clés, voire un seul mot-clé, et, évidemment, ne renvoie que des valeurs similaires. Un exemple de ce genre de fonction lexicale non standard est fourni ici :

(17) 'à peine cuit'(steak) = saignant

Dans ce cas-ci, la relation existant entre *steak* et *saignant* est ‘à peine cuit’. Or, cette relation est très spécifique et ne peut pas s'appliquer à d'autres mots-clés, pas même des mots-clés du champ sémantique des aliments. Il n’est donc pas possible de reproduire la relation ‘à peine cuit’ sur autre chose que ce qui s’apparente à de la viande, et le nombre de valeurs retournées ne sera pas très élevé. Il faut donc en conclure que la distinction principale entre une fonction lexicale standard et une fonction lexicale non standard est quantitative. Cependant, c'est ce côté quantitatif qui permet de rendre une FL généralisable et systématique. Comme le travail fourni dans ce mémoire est de générer des collocations à l’aide de règles génériques et systématiques, le traitement des fonctions lexicales non standard n'a pas été étudié.

(I. Mel′čuk, 2013, p. 137) indique également que les FL standard sont aussi indiquées dans la description du paraphrasage :

> Les règles universelles de paraphrasage, qui reposent sur le recours aux FL, permettent d'obtenir automatiquement toutes les paraphrases similaires dans toute langue – pourvu, bien entendu, qu'on dispose, pour cette langue, d'un dictionnaire où les FL (autrement dit, les collocations) sont spécifiées pour chaque lexie vedette.

Au contraire, les FL non standard ne participent pas dans le paraphrasage. On note également le caractère universel des FL standard. Nous aimerions justement attirer l'attention du lecteur sur une remarque pertinente de (Van Der Wouden, 1992, p. 452) au sujet du caractère universel des FL standard :

> It is, given their very general nature indeed, intuitively plausible that the lexical functions Mel′čuk and his group have been studying are universal cross-linguistically. However, the way they are implemented across the various languages of the world may differ dramatically.

Par exemple, nous avons vu que Magn permet d'introduire un modificateur adjectival représentant le sens d'intensification. Cependant, (Van Der Wouden, 1992) spécifie qu'en néerlandais, l'intensification se marque à la fois avec un modificateur adjectival mais aussi à l'aide d'une composition morphologique :

(18) *ijskoud* (litt. 'glace-froid', *froid glacial*)

(19) *stomdronken* (litt. 'muet-saoul', saoul comme un cochon)

(20) *stomvervelend* (litt. 'muet-ennuyant', ennuyeux comme la pluie)

Cet auteur affirme donc que les fonctions lexicales sont des restrictions sur la combinaison lexicale mais également à un niveau plus haut (syntagme) ou plus bas (morphologique).

Enfin, (Polguère, 2007) et (Jousse, 2003, 2010) précisent qu'il existe des FL semi-standard et des FL localement-standard. (Jousse, 2010, p. 78) définit une FL semi-standard comme suit :

> Les FL semi-standard sont, en quelque sorte, des extensions des FL standard permettant d'ajouter une composante de sens qui n'est pas pris en compte dans les FL standard dont elles sont dérivées.

L'auteure fournit deux exemples marquant la distinction entre une FL standard et semi-standard :

(21) $CausOper_1$(silence)= réduire [quelqu'un au ~]

(22) En échange de quelque chose. $CausOper_1$(silence) = acheter [le ~ de quelqu'un ]

Dans les exemples (21) et (22), la composante de sens ajoutée est 'en échange de quelque chose'. Cette composante ne dénote pas un sens généralisable.

Le concept de FL localement-standard a été proposé par (Polguère, 2007). Contrairement aux FL standard, l'universalité de la relation qu'elles instancient n'a pas été démontrée. Autrement dit, il s'agit de relations standard pour une langue donnée. Les exemples (23) et (24) illustrent deux cas de FL localement-standard :

(23) De_nouveau$Func_0$(hostilité) = les ~ reprennent

(24) Essayer_de$Oper_2$(augmentation) = réclamer une ~

Ces FL n'ont pas été traitées dans le présent mémoire.

#### 2.2.1.2 Axe paradigmatique/syntagmatique

Il est également possible de classer les FL en fonction du type de relation qu'elles modélisent. Cette relation peut être soit d'ordre paradigmatique ou syntagmatique (Alonso Ramos & Tutin, 1996; I. Mel′čuk, 1996). Traditionnellement, les FL étaient réparties en deux groupes par (Žolkovskij & Mel'čuk, 1967). Le premier faisait état des « substitutes », où la valeur de la FL sert à remplacer le mot-clé dans la phrase. C'est cette idée que l'on retrouve pour qualifier les FL paradigmatiques. Voici quelques exemples de FL paradigmatiques :

(25) $S_1$(meurtre) = meurtrier [N]

(26) $S_2$(meurtre) = victime

(27) Anti(mépris) = respect

(28) Syn(espoir) = espérance

De même, (Alonso Ramos & Tutin, 1996) indiquent que le mot-clé et la valeur retournée par une FL paradigmatique doivent partager une relation sémantique, cette relation pouvant être l'équivalence, l'inclusion ou l'intersection. Les FL paradigmatiques permettent donc d'exprimer des relations de dérivation sémantique. Ces auteurs indiquent également qu'il faut être prudent lorsque l'on emploie la dichotomie paradigmatique/syntagmatique car le sens de ces termes n'est pas tout le temps explicite. Comme nous le verrons dans la sous-section suivante, certains encodages des FL pallient aux défauts de ces définitions.

L'axe syntagmatique fait référence à la combinatoire des unités lexicales, et donc aux collocations. Des relations syntaxiques existent entre des unités lexicales qui apparaissent ensemble, ou qui co-occurrent (I. Mel′čuk, 1996). Cette classe fait référence aux « parameters » de (Žolkovskij & Mel'čuk, 1967). Ce sont ces FL qui permettent notamment l'insertion d'un verbe support, de modificateurs adjectivaux ou adverbiaux, ou encore des prépositions. Voici quelques exemples (la base est marquée par le symbole '~') :

(29) $Oper_{12}$(attention) = faire ~ à N

(30) $Oper_2$(attention) = attirer ART ~

(31) Ver(heure) = ~ exacte

(32) $Loc_{ad}$(filet) = dans ART ~

Il est vrai que dans certains cas, cette dichotomie ne permet pas de classer certaines FL. Par exemple, Figur, Gener ou encore $A_1$ sont traditionnellement classées parmi les FL paradigmatiques, cependant dans certains cas, leur valeur peut se combiner avec le mot clé :

(33) Gener (colère) = sentiment de ~

(34) Figur (fumée) = écran de ~

(35) $A_1$(mépris) = couvert de ~

Ce fait est décrit par (Alonso Ramos & Tutin, 1996). Elles proposent une notation spécifique pour distinguer l'emploi syntagmatique ou paradigmatique de ces FL.

Pour conclure, les FL que nous traitons dans ce mémoire sont celles pouvant être généralisées en règles universelles, c'est-à-dire applicables aux différentes langues traitées par GÉCO, et traitant de la combinatoire des unités lexicales. Il s'agit donc des FL standard et syntagmatiques.

#### 2.2.1.3 Axe simple/complexe/configuration

Parmi les FL standard, certaines sont qualifiées de FL simples. (I. Mel′čuk, 1996) indique qu'une FL standard simple ne peut pas être représentée à l'aide d'autres FL. Autrement dit, la relation instanciée par la FL standard simple est en quelque sorte non décomposable. (Kahane & Polguère, 2001) mettent l'emphase sur l'idée qu'il existe près de soixante relations « primitives » qui peuvent par la suite être combinées entre-elles. La notion de FL simple découle donc du type de relation qu'elle instancie. Il existe environ soixante FL standard simples. En voici quelques exemples tirées du DEC IV (I. Mel′čuk et al., 1999) :

(36) $\text{Func}_0$(pluie) = ART ~ tomber

(37) $\text{Real}_1$(promesse) = tenir ART ~

(38) $\text{Labreal}_{12}$(mémoire) = conserver N en ~

(39) Propt(corde) = avec ART ~

Cette liste est assez restreinte et ne semble pas nécessairement rendre compte de l'ensemble des collocations. Heureusement, il est possible de combiner les FL entre elles afin de créer des relations plus complexes entre les éléments de la collocation.

Les FL complexes représentent une manière de combiner les FL ensemble. (I. Mel′čuk, 1996, p. 73) précise ce qu'est une FL complexe:

> A Complex LF is a combination of syntactically related LFs that has a single, or cumulative, expression covering the meaning of the combination as a whole.

Une FL complexe est donc une FL constituée à partir de deux ou plusieurs autres FL, dont le sens englobe celui de ses constituants. Ces différents constituants entretiennent une relation en syntaxe profonde. Prenons par exemple :

(40) $\text{IncepOper}_1$(ami) = devenir

(41) AntiBon(choix) = mauvais

Les constituants de ces FL sont intrinsèquement liés entre eux. Il est possible de représenter cette relation comme suit :

(42) Incep → II → Oper

(43) Anti ← ATTR ← Bon

La littérature Sens-Texte précise bien qu'une FL complexe n'est pas une composition fonctionnelle au sens mathématique (I. Mel′čuk et al., 1995). Autrement dit, pour une FL complexe **fg** appliquée sur un mot-clé L, la valeur retournée L′ est égale à **fg**(L) et non à **f**(**g**(L)). Ce n'est pas toujours vrai. Par exemple, dans le cas de (44), la FL $A_1$ est une dérivation adjectivale s'appliquant sur le reste de la FL :

(44) $A_1Real_1$(chapeau) = coiffé de ART ~

Il est également possible de combiner des FL simples et des FL complexes pour former des configurations de FL. À la différence des FL complexes, les constituants de configuration de FL ne sont pas syntaxiquement liés (Jousse, 2010; I. Mel′čuk, 1996). Une configuration de FL est donc une suite de FL simples, ou complexes, possédant le même mot-clé, cette suite ayant une valeur globale cumulative qui exprime de façon indécomposable le sens de la suite entière (Jousse, 2010). Par exemple :

(45) Magn.$Oper_1$(affection)=déborder [Posséder ($Oper_1$) de l'affection, en particulier une affection de grand intensité (Magn)]

Dans le cas d'une configuration de FL, le dernier constituant est celui qui fournit l'information syntaxique de la valeur. En (45), $Oper_1$ permet d'indiquer que la valeur retournée sera un élément verbal. Tous les éléments précédents ne font qu'apporter du sens. Ici, Magn apporte l'idée d'intensification seulement. Enfin, on peut noter que les constituants d'une configuration de FL peuvent se réaliser de manière distincte. L'exemple (45) pourrait donc se lexicaliser par *ressentir de l'affection profonde* où $Oper_1$ et Magn sont exprimés séparément. Bien entendu, cette formulation paraît plus libre que *déborder*. Néanmoins, cela nous permet de dire que pour toute configuration de fonctions lexicales **f.g** portant sur le mot clé L, la formulation suivante est valide: **f.g**(L) = f(L) ∩ g(L).

### 2.2.2 Encodage des fonctions lexicales

(Jousse, 2010) mentionne que les lexicographes font souvent l'amalgame entre le principe des FL et leur formalisme.

Nous présentons deux types d'encodage : l'encodage traditionnel et l'encodage explicite. L'encodage traditionnel, et celui utilisé dans les DEC (I. Mel′čuk et al., 1995) ressemble à ce que nous avons utilisé précédemment:

(46) $Oper_1$(colère) = éprouver de la ~

(47) Magn(colère) = ~ noire

(48) $A_1$(colère) = N en ~

L'encodage explicite est fourni dans (Kahane & Polguère, 2001). Ce nouvel encodage a été conçu pour pallier aux défauts de l'encodage plus traditionnel du formalisme DEC: les fonctions lexicales ne sont pas modélisées de manière complètement explicite, ce qui est dommageable tant d'un point de vue théorique que computationnel. Leur réponse est d'encoder séparément le contenu sémantique de la FL et son comportement syntaxique.

Dans leur approche, le contenu sémantique d'une FL est une configuration de relations de type prédicat-argument entre des sens primitifs. Ces sens primitifs correspondent aux fonctions lexicales standard simples énoncées précédemment. À ces sens se rajoutent différentes notations comme '#' (sens du mot-clé), 'Ω' (participant sémantique non spécifié), '^' (exprime la spécification) etc.

En plus du contenu sémantique, un cadre syntaxique est fourni, représentant à la fois la partie du discours de la valeur ainsi que la diathèse des dépendants syntaxiques (I, II, etc.). Cet encodage ne fournit pas de représentation syntaxique plus élaborée car le plus important est de marquer l'organisation communicationnelle de la structure.

Au final, cet encodage permet de représenter une FL à l'aide d'une matrice contenant les informations sémantiques et syntaxiques. Voici les représentations respectives de $Oper_1$(colère), Magn(colère), $A_1$(colère) des exemples (46), (47) et (48) d'après le texte de Kahane et Polguère (2001).

(49) $\left(V^{\#}_{[1,\#]}\right)$(colère) = éprouver de la ~

(50) $\binom{\{\#\}\hat{}Magn}{A[\#\hat{}]}$(colère) = ~ noire

(51) $\binom{\{1\}\hat{}\#}{A[1,\#]}$(colère) = N en ~

Pour les auteurs, les dérivations sémantiques (liens paradigmatiques) et les collocations sont conceptuellement liées. Cela transparait dans leur encodage car la seule distinction existant entre une FL paradigmatique et une FL syntagmatique est la présence du '#' dans la partie syntaxique. Ce symbole renvoie au sens du mot-clé. De ce fait, quand ce symbole est présent, la FL introduit une valeur en plus du mot-clé, donc un collocatif. Par contre, dans le cas où ce symbole est absent de la partie syntaxique, alors la FL ne retourne pas le mot-clé, seulement sa valeur. Les exemples (50) à (51) démontrent bien ce propos. Ce mécanisme permet de mieux comprendre le fonctionnement de certaines fonctions lexicales jugées paradigmatiques mais pouvant avoir un comportement syntagmatique.

Dans leur article, les auteurs mentionnent que les sens primitifs peuvent se combiner pour former des sens complexes. Voici l'exemple donné :

(52) Caus[1, Minus[Manif[#]]] [= 'X cause une diminution dans la manifestation de #']

Même si la FL complexe décrite en (52) n'est pas une composition au sens mathématique du terme, elle présente une composition d'un point de vue sémantique. C'est grâce à ce principe de composition sémantique que nous avons implémenté les FL complexes dans GÉCO. L'encodage explicite est important pour ce travail car il permet de rendre le mécanisme des FL plus facilement implémentable dans notre réalisateur de texte. Comme nous le verrons par la suite, plusieurs de ces principes sont implémentés tels quels dans GÉCO.

### 2.2.3 Les fonctions lexicales en GAT

Les fonctions lexicales ont été implémentées dans plusieurs logiciels de TAL (Apresjan, Boguslavsky, & Tsinman, 2007). Par exemple, (Heylen, Maxwell, & Verhagen, 1994) et (Maxwell & Heylen, 1994) présentent le potentiel des FL pour la traduction automatique. De même, nous avons vu au chapitre précédent que le générateur Marquis se sert des FL pour la génération automatique de texte multilingue. Steinlin (2003) a aussi travaillé sur l'incorporation de trois FL ($Oper_i$, Magn et $Loc_{in}$) pour le formalisme G-TAG. Traditionnellement, la Théorie Sens-Texte indique que les FL n'apparaissent pas dans la

RSem, mais sont présentes en RSyntP, et leurs valeurs sont intégrées à la phrase en RSyntS, comme la Figure 5 le montre.

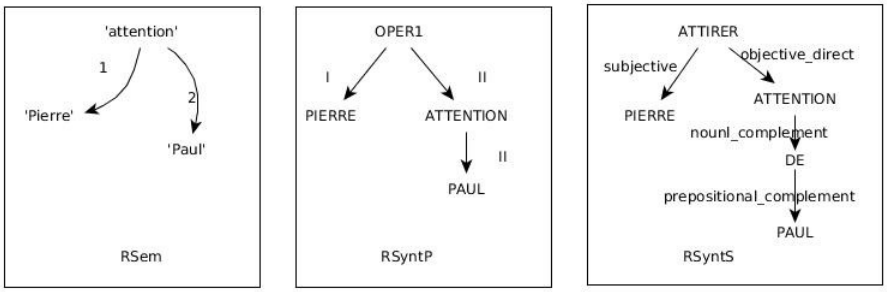



Figure 5. Lexicalisation des fonctions lexicales et collocatifs

Cependant, (Lidija Iordanskaja, Kim, & Polguère, 1996, p. 280) abordent plusieurs problèmes et questionnements liés à l'implémentation des FL dans un système de TAL : quand faut-il intégrer les FL dans un système de TAL? Est-ce que leurs noms doivent figurer en RSyntP ou bien leurs valeurs directement? Comment doivent-elles être encodées dans les ressources employées par le système de TAL?

Cette section a pour objectif de présenter plusieurs travaux en GAT se basant sur les FL. Nous comparerons leur méthodologie et présenterons les problèmes liés à l'implémentation des FL en GAT.

#### 2.2.3.1 Heid & Raab (1989)

(Heid & Raab, 1989) présentent succinctement leur méthodologie de structuration de l'information collocationnelle dans des dictionnaires en vue de faire de la génération automatique de texte multilingue. Ils proposent d'organiser les dictionnaires de la manière la plus modulaire possible afin de séparer l'information pertinente aux différents aspects linguistiques. Ils proposent de ce fait trois dictionnaires :

1. Un dictionnaire sémantique englobant l'inventaire des lexicalisations de concepts;
2. Un dictionnaire syntaxique répertoriant les lexèmes pour chaque langue ainsi que les informations sur la réalisation de ces lexèmes;
3. Un dictionnaire morphologique contenant les classes flexionnelles pour chaque langue.

Ils insistent sur l'interdépendance de ces dictionnaires. En effet, l'information contenue dans l'un est utile pour choisir celle de l'autre. De ce fait, les dictionnaires doivent être structurés en classes et contenir des références vers les autres dictionnaires.

Ils définissent les collocations d'après (Hausmann, 1985, p. 119):

> One partner determines, another is determined. In other words: collocations have a basis and a co-occurring collocate.

Cette vision des collocations entraine deux heuristiques quant à leur processus de génération :

1. Il faut d'abord lexicaliser la base, puis le collocatif
2. L'information sur les possibilités de combinaisons doit être entreposée dans le dictionnaire sémantique, sous l'entrée de la base

Étant donné leur définition des collocations, les auteurs précisent qu'il est naturel d'exploiter la notion de fonction lexicale pour encoder les possibilités de combinaisons. Ils organisent leur dictionnaire sémantique de façon à ce que chaque entrée présente son inventaire de FL, voir exemple (53).

(53) Structuration d'une entrée du dictionnaire de lexicalisation :

```
(problem
      (…)
      (caus func (create, poss))
      (real (solve, …))
      (...))
```

En plus d'intégrer les FL pour chaque lexème, ils se basent sur les informations récupérées sur le travail théorique des FL pour prédire les types de FL applicables aux lexèmes appartenant à une même classe sémantique. Par exemple, les noms dénotant un état mental, comme *admiration*, *colère*, *joie*, etc., possèdent quasiment tous la FL $\text{Oper}_1$ dont la valeur est *ressentir*. Autrement dit, ils opèrent une généralisation sur l'application et le comportement des FL en fonction des propriétés sémantiques de la base. Ce principe de généralisation des FL est d'autant plus intéressant en GAT multilingue. En effet, les auteurs montrent que les noms exprimant de l'information manipulée par un ordinateur, comme *répertoire*, *fichier*, *message*, etc., possèdent non seulement les mêmes FL avec les mêmes valeurs, mais cette généralisation

est valable pour le français (I-NounSF) et l'allemand (I-NounSG). L'exemple suivant illustre ce propos.

(54) $LiquFunc_0$(I-NounSF)= supprimer

(55) $LiquFunc_0$(I-NounSF)= löschen

Évidemment, ces généralisations présentent des exceptions. Ces dernières doivent figurer explicitement dans le dictionnaire sémantique.

Les auteurs poursuivent en abordant la question de la génération de paraphrases. Deux types d'informations doivent figurer dans le générateur: l'information statique et l'information procédurale. L'information statique présente l'information sur les variantes syntaxiques et lexicales et doit être intégrée de manière déclarative. L'information procédurale est un ensemble d'heuristiques qui guident le choix de variantes. Ces informations permettent de spécifier les relations de paraphrasage entre différents candidats de lexicalisation et de guider le choix du bon candidat. Cela englobe la description des comportements syntaxiques des bases et collocatifs. D'ailleurs, les chercheurs mentionnent qu'il serait intéressant d'étudier dans quelle mesure il est possible de décrire le comportement syntaxique des collocations à l'aide de règles génériques.

Ce qu'il faut retenir de leur approche est que les FL doivent être mentionnées à la fois dans les dictionnaires du système de GAT et dans les heuristiques guidant la lexicalisation (règles, processus computationnel). Leur travail ouvre la question de la généralisation: est-il possible de généraliser l'application des FL, en fonction du type sémantique de la base mais aussi du point de vue de son comportement syntaxique? Ces questions trouvent réponse dans notre traitement des collocations, comme nous le verrons en détail lors de la présentation de la méthodologie d'implémentation, au Chapitre 3.

#### 2.2.3.2 Iordanskaja et al. (1992; 1996)

(L. Iordanskaja, Kim, Kittredge, Lavoie, & Polguère, 1992) présentent le générateur bilingue LFS (Labour Force Surveys). Son module de réalisation linguistique part d'une représentation conceptuelle et indépendante de la langue. Elle est donc partagée par les deux langues du système, l'anglais et le français, et sert à produire des structures sémantiques sous forme de graphe, celles-ci spécifiques à chaque langue. Ce haut degré d'abstraction permet de

traiter les différences sémantiques entre le français et l'anglais. Les exemples (56) et (57) montrent à quel point un message identique peut se réaliser de diverses manières en sémantique, et par conséquence en syntaxe:

(56) Employment remained virtually unchanged.

(57) L'emploi a peu varié.

Le module de planification de texte de LFS est élaboré, ce qui est visible au travers de la complexité et longueur des textes générés. L'intérêt de ce générateur pour notre étude est qu'il exploite les FL et présente une grammaire à large couverture. L'objectif des auteurs est donc de se servir du modèle linguistique puissant fourni par la Théorie Sens-Texte afin de simplifier la mise à niveau et la réutilisation du générateur vers des textes plus complexes et variés.

La réalisation linguistique pour l'anglais dans LFS s'opère grandement à partir des travaux sur GOSSIP (Kittredge, Iordanskaja, & Polguère, 1988). Le module traitant du français a été créé entièrement. LFS se base sur quatre niveaux majeurs de représentation linguistique : structure sémantique (graphes sémantiques), structure syntaxique profonde (arbre de dépendance profond), structure syntaxique de surface (arbres de dépendance de surface) et la structure morphologique (chaîne morphologique). Il part de la structure sémantique et la dérive jusqu'à obtenir le texte final. Cette architecture est dérivée des modèles Sens-Texte.

La lexicalisation s'opère au niveau sémantique. La structure sémantique (RSem) est modifiée : les sémantèmes sont remplacés par des lexèmes et les traits sémantiques, sous forme d'attributs, sont remplacés par des traits grammaticaux. Il en résulte une structure sémantique réduite (RSemR) sous forme de graphe. Cette représentation sert d'intermédiaire entre la structure sémantique et la structure syntaxique profonde (RSyntP). Pour passer de RSemR à RSyntP, le logiciel arborise les relations prédicat-arguments de la RSemR et obtient ainsi un arbre de dépendance représentant les dépendances profondes. Les lexèmes grammaticaux n'apparaissent qu'au niveau syntaxique de surface.

Les auteurs perçoivent les FL comme un moyen de remettre à plus tard la réalisation d'idiosyncrasies lexicales après avoir effectué des choix syntaxiques majeurs. Elles permettent également de générer des paraphrases. Les FL sont intégrées dans l'interface RSemR ↔ RSyntP. Ensuite, leurs valeurs sont introduites lors de l'interface RSyntP ↔ RSyntS excepté

pour les LF sémantico-pragmatiques (Syn), introduites en RSemR, voir Figure 6. L'utilisation des FL suit deux objectifs : (i) contrôler le choix des lexèmes dans les cas de cooccurrence lexicale restreinte et (ii) choisir une construction syntaxique correcte ou appropriée. L'implémentation des FL pose plusieurs problèmes (L. Iordanskaja et al., 1996).

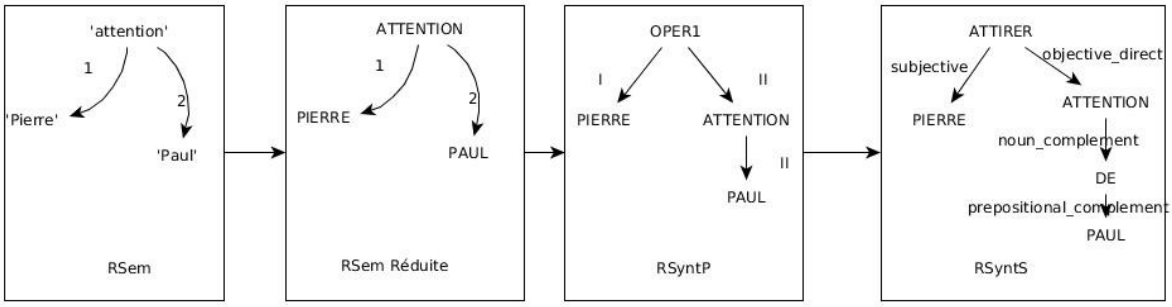



Figure 6. Lexicalisation des fonctions lexicales dans Iordanskaja et al. (1996)

Tout d'abord, la stratégie de génération fournie par la théorie Sens-texte vise à produire toutes les représentations possibles à partir de la représentation de départ. Autrement dit, pour une RSem donnée, le système fournit toutes les RSyntP possibles, même lorsque le résultat est fautif, ce qui bloque le processus de génération. De même, l'emploi des FL augmente le risque de résultats fautifs. En effet, alors que les noms des FL sont intégrés en RSyntP, leurs valeurs ne sont intégrées qu'au niveau suivant, en RSyntS. D'un point de vue pratique, le risque est de générer une RSyntP présentant une FL sans valeur et donc de se retrouver avec une RSyntS fautive et lacunaire. Par ailleurs, l'objectif de la GAT est de produire un seul énoncé. De ce fait, LFS ne produit qu'une seule structure de sortie à chaque étape du processus de génération. Afin de ne pas produire une RSyntS lacunaire, les auteurs tentent d'abord de vérifier en amont si la FL introduite en RSyntP possède bel et bien une valeur. Cependant, cette méthode présente un problème de taille: il arrive que l'utilisation d'une FL dépende de l'emploi d'une autre FL. Dans ce cas, il devient vite difficile de vérifier les valeurs de différentes FL. Ils décident donc d'introduire les valeurs des FL directement en RSyntP et de garder une trace de l'application de la FL.

Les FL peuvent également être introduites dans le passage entre RSyntP et RSyntS. C'est le cas de $Loc_{in}$ lorsqu'il marque la préposition requise par un verbe dans son régime. L'exemple fournit est HABITER, qui requiert un $Loc_{in}$ car le choix de la préposition dépend du nom se

réalisant comme le second actant de HABITER (*habiter en Ontario, habiter à Québec*). Techniquement, $Loc_{in}$ devrait figurer dans l'entrée du nom en question, et non dans l'entrée du verbe. Les auteurs mentionnent que les FL introduites à ce moment permettent aussi d'opérer des transformations purement syntaxiques, comme des nominalisations, dans le cas suivant : le nœud Y est un actant du nœud X mais ne possède pas la bonne partie du discours en RSyntP, par exemple Y est un verbe au lieu d'un nom. La FL $S_0$ permet alors de nominaliser ce nœud pour satisfaire aux exigences de X. Ce cas de figure est une conséquence directe de la séparation entre lexicalisation (RSem ↔ RSemR) et arborisation (RSemR ↔ RSyntP), rendant propice la création d'arbres de dépendance fautifs.

Enfin, les auteurs mentionnent que certaines FL comme $Loc_{in}$ apparaissent à différents niveaux du processus de réalisation. Dans le cas de $Loc_{in}$, il est possible de l'introduire en RSemR afin de contrôler le processus de lexicalisation. Cela nous pousse à nous demander: ne serait-il pas plus efficace d'introduire les FL, et leurs valeurs, le plus tôt possible dans le processus de réalisation?

Les entrées de dictionnaire dans LFS sont encodées comme dans un Dictionnaire Explicatif et Combinatoire. Autrement dit, une entrée pour un lexème L inclue toutes ses FL à l'aide de paires ⟨ FL, {valeurs $L_1$, ... $L_n$} ⟩. Chaque valeur possède une entrée séparée décrivant leur information.

Pour conclure, la méthodologie proposée par (L. Iordanskaja et al., 1992; Lidija Iordanskaja et al., 1996) est finalement d'introduire les FL sur plusieurs niveaux en fonction des besoins du générateur. Ils préconisent d'insérer les valeurs des FL le plus tôt possible. L'insertion des FL dans leur système se fait principalement lors de l'interface entre la structure sémantique réduite et la structure syntaxique profonde à l'aide de règles de transition. L'application de ces règles est déterminée par des conditions, notamment celle indiquant qu'il ne faut introduire une FL que si elle possède une valeur pour la lexie correspondante. Ces règles font donc un appel aux dictionnaires.

#### 2.2.3.3 Lareau et al. (2011a, 2012)

Ce travail se distingue de ceux cités précédemment en raison de l'implémentation des FL dans une architecture basée sur le formalisme des Grammaires Lexicales Fonctionnelles

(LFG). (Lareau, Dras, Börschinger, & Dale, 2011a; Lareau, Dras, Börschinger, & Turpin, 2012) cherchent à générer des commentaires de football australien en anglais et en arrernte, une langue aborigène australienne. Malgré des différences significatives entre ces deux langues, certains phénomènes sont récurrents et similaires, comme l'emploi de verbes supports. C'est pourquoi l'équipe recourt aux FL.

L'architecture de leur générateur est présentée dans la Figure 7 (Lareau et al., 2011a, p. 98). Le module de réalisation profonde fournit une correspondance entre une structure sémantique et sa f-structure (structure de traits encodant les fonctions syntaxiques). Cette f-structure est ensuite envoyée à XLE (Maxwell & Kaplan, 1993) qui accomplit la suite du processus de réalisation. Le cadre LFG prévoit différentes représentations linguistiques ainsi que des règles pour dériver ces niveaux, (Bresnan, 2001; Dalrymple, 2006). Cependant, l'important ici est de comprendre que la f-structure énonce des relations fonctionnelles abstraites ainsi que des dépendances syntaxiques (relation sujet/objet, etc.). L'inventaire des relations possibles varie d'une langue à l'autre mais il existe néanmoins plusieurs concepts communs à plusieurs langues (Dalrymple, 2006), d'où l'intérêt d'exploiter une telle structure comme input à un réalisateur de texte. De même, on pourrait ajouter que les relations syntaxiques instanciées dans la f-structure sont suffisamment abstraites pour ressembler à celles présentes dans le niveau de syntaxe profonde (RSyntP) en Théorie Sens-Texte. Il parait donc évident d'intégrer les FL dans la f-structure.

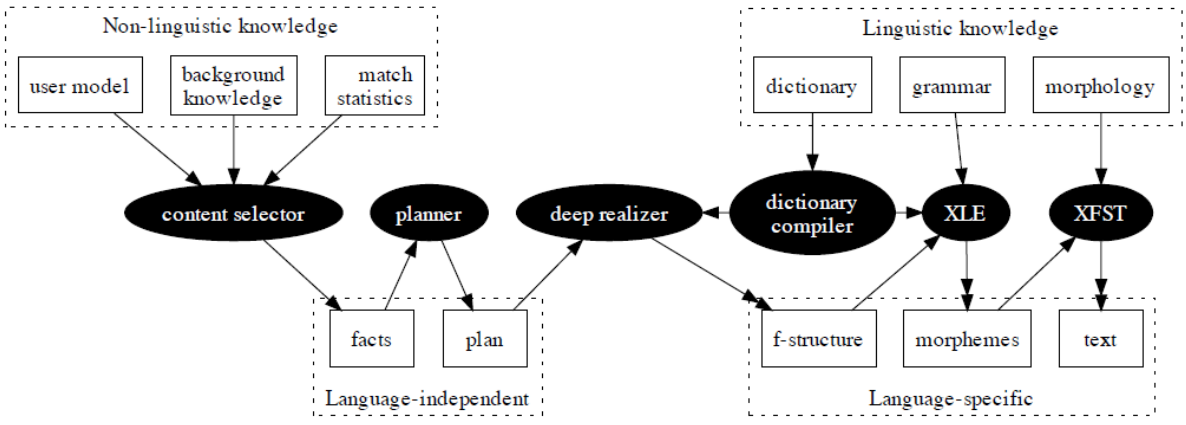



Figure 7. Architecture d'un GAT basé sur LFG (Lareau et al., 2011a, p. 98)

Cependant, les auteurs précisent que les FL ne peuvent pas être directement intégrées dans la f-structure car cette dernière n'accepte pas les prédicats à variables ou les structures

indéterminées, comme celle représentée en Figure 8, (Lareau et al., 2011a) représentant la phrase (58), où le prédicat n'est pas intégré directement à la structure.

(58) Bradshaw kicked a beautiful goal (litt. 'Bradshaw a botté un joli but')

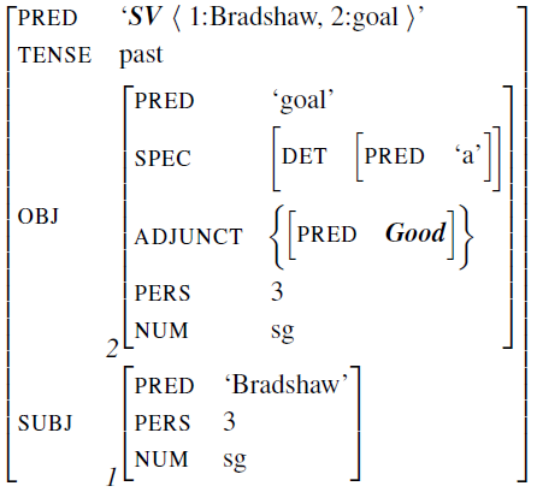



Figure 8. f-structure de l'énoncé « Bradshaw kicked a beautiful goal. »

Ils se servent alors des principes de la « glue semantics » (Andrews, 2010) pour intégrer les FL à la f-structure. Cette théorie se base sur la logique linéaire : les prémisses sont consommées lors de la déduction logique. Ce principe est avantageux en GAT, lors de la construction du sens de la phrase. Dans le cas présent, cela s'exprime dans le dictionnaire par la mention du constructeur sémantique dans chaque entrée lexicale. Le constructeur sémantique fonctionne comme des briques pour construire le sens complet de la phrase à partir de ses constituants. L'exemple de *Bradshaw* illustre le fonctionnement de base de ce principe.

(59) Bradshaw N ("PRED)='Bradshaw'

Bradshaw : $\uparrow_\sigma$

En (59), la représentation de la construction sémantique se fait comme suit: le sens est représenté à gauche du « : », ici « Bradshaw », et la formule de construction sémantique se trouve à droite du « : ». Le symbole « σ » représente la projection sémantique d'un nœud. Ainsi, la ligne «Bradshaw: $\uparrow_\sigma$ » informe que la projection sémantique du nœud mère porte le sens « Bradshaw ».

(60) goal N (↑PRED)='goal'

$$\lambda X.goal(X) : ((OBJ\uparrow)SUBJ)_{\sigma} \multimap \uparrow_{\sigma}$$

(61) @OPER1(L)=

(↑ PRED )= '%stem⟨(↑ SUBJ ),(↑ OBJ )⟩'

(↑ OBJ PRED )= $_{c}$ 'L'

$\lambda X.X : (\uparrow OBJ)_{\sigma} \multimap \uparrow_{\sigma}$

(62) kick V @(OPER1 goal)

Les exemples (60) à (62) représentent le fonctionnement de la FL « $Oper_1$(goal)= kick ». La base, « goal » est un prédicat sémantique, marqué par la ligne « $\lambda X.goal(X) : ((OBJ\uparrow)SUBJ)_{\sigma} \multimap \uparrow_{\sigma}$ ». L'exemple (61) définit un patron de fonctionnement de la FL $Oper_1$. La ligne 2 de cette entrée indique que le verbe support est contraint par l'entrée lexicale qui l'appelle. Dans le cas présent, il s'agit de 'kick', tel que montré en (60). Le patron de $Oper_1$ permet également de récupérer le sens de son objet syntaxique, 'goal', car il s'agit de la base de la collocation. Dans le cas d'un $Func_i$, il aurait fallu récupérer le sens du sujet syntaxique.

Ce type de patron est suffisamment abstrait pour être appliqué à n'importe quelle langue. Il s'agit d'une généralisation du fonctionnement de la FL $Oper_1$. De ce fait, toutes les instances d'$Oper_1$ sont traitées dans toutes les langues grâce à cet unique patron. Ce travail de généralisation fait donc écho aux travaux de (Heid & Raab, 1989) énoncés plus tôt.

## 2.3 Synthèse

Ce chapitre nous a permis de préciser le concept de « collocations ». Nous avons vu que ce terme possède plusieurs définitions, basées sur des critères différents, et que le meilleur moyen d'appréhender le comportement idiosyncrasique des collocations est de les percevoir comme des restrictions de sélection lexicale. Autrement dit, une collocation est la combinaison de trois éléments: (i) le choix libre d'un lexème X, (ii) la volonté d'exprimer une relation sémantique particulière **f** sur ce lexème et (iii) la sélection contrainte d'un lexème Y permettant d'exprimer cette relation.

La Théorie Sens-Texte a mis au point un mécanisme pour décrire ces collocations: les fonctions lexicales. Une fonction lexicale **f** s'exprime comme suit :

**f**(X)=Y

En plus de représenter les collocations de manière systématique, les FL permettent de corréler certaines collocations grâce à la relation sémantico-lexicale qu'elles dénotent. C'est le cas de la FL Magn qui instancie la relation d'intensification entre la base et son collocatif. Nous avons également vu qu'il existe beaucoup de FL et que ces dernières ont été classifiées selon trois grands axes :

- axe standard/non standard
- axe paradigmatique/syntagmatique
- axe simple/combinaison

Par ailleurs, plusieurs travaux relatent de leur expérience avec l'implémentation des FL en GAT. Ces travaux ont permis de mettre au jour plusieurs stratégies pour optimiser le traitement des FL :

- Le comportement des FL est généralisable;
- Il est possible d'intégrer les FL à différents stades du processus de réalisation linguistique;
- Les FL sont adaptées au traitement multilingue;
- Il est avantageux d'intégrer les valeurs des FL le plus tôt possible dans le processus de réalisation;
- La lexicalisation et l'arborisation sont des tâches qui ne devraient pas être séparées;
- Les FL peuvent être représentées aussi bien dans les dictionnaires que dans les règles de transition entre deux niveaux de représentation linguistique.

Le prochain chapitre présente notre réalisateur de texte, GÉCO ainsi que la méthodologie d'implémentation des FL.

# Chapitre 3 Le générateur de surface GÉCO

Dans ce chapitre, nous présentons notre méthodologie d'implémentation des fonctions lexicales (FL), que nous intégrons dans un générateur automatique de texte multilingue en cours de développement, GÉCO (GÉnération de COllocations). Nous commençons ce chapitre par la présentation de GÉCO : son origine, son architecture et son fonctionnement sont directement repris de MARQUIS (Lareau & Wanner, 2007; Wanner, Bohnet, et al., 2007; Wanner et al., 2010). La deuxième partie aborde la méthodologie d'implémentation des collocations comme telle. La première étape de notre démarche identifie les régularités parmi les collocations à l'aide des FL de la Théorie Sens-Texte (I. Mel′čuk, 1973, 1997; I. A. Mel′čuk, 1988). La deuxième étape consiste à décrire ces régularités en accord avec le fonctionnement de GÉCO. Enfin, la dernière étape intègre ces modélisations dans la plateforme de développement MATE.

## 3.1 GÉCO: successeur de MARQUIS

GÉCO est un générateur automatique de texte multilingue en cours de développement. Il est le successeur de MARQUIS (cf. 1.2.2.4). L'objectif de GÉCO est de rajouter à MARQUIS la description formelle des patrons récurrents de collocations que l'on retrouve à travers les langues. Leur architecture et leur fonctionnement sont donc pratiquement identiques.

MARQUIS (Multimodal Air QUality Information Service for General Public) est un générateur automatique de texte multilingue (Wanner, et al., 2007). Il génère des bulletins sur la qualité de l'air dans différentes régions d'Europe de manière compréhensible pour le grand public.

La création de MARQUIS répond à des besoins tant économiques (tourisme) que sanitaires (connaître le taux de pollution de l'air) ou encore légaux (obligation de communiquer de manière exhaustive et transparente l'état de la qualité de l'air). Comme l'indiquent (Wanner et al., 2010), les services de transmission d'information sur la qualité de l'air ne font en général que présenter les données brutes, peu ou non interprétées. MARQUIS a donc été développé dans une optique orientée utilisateur: chaque bulletin fournit à l'utilisateur un contenu approprié en termes d'interprétation des données et de présentation de ce contenu. Lors de sa création,

MARQUIS proposait donc un service innovant car sa technologie repose sur un ensemble de données non prises en compte par ses prédécesseurs: un ensemble de profils utilisateurs, la transmission de données s'étendant sur plusieurs régions européennes, des moyens de diffusion modernes (web, email, sms), la couverture des substances polluantes majeures, un interpréteur poussé de traitement de données et un moteur de génération de texte de pointe permettant de générer des bulletins de façon multilingue. Il prend en charge huit langues: le catalan, l'anglais, le français, le finnois, l'allemand, le polonais, l'espagnol et le portugais.

L'avantage principal de MARQUIS pour nous est que son architecture est basée sur les principes de la Théorie Sens-Texte. Par ailleurs, MARQUIS se base sur des dictionnaires riches. Enfin, il a été développé sur MATE, un transducteur de graphes. Nous allons aborder ces points maintenant.

### 3.1.1 Architecture

Le système MARQUIS présente une architecture à deux niveaux principaux : le premier effectue le traitement des données et le deuxième gère les requêtes utilisateurs (Wanner, Bohnet, et al., 2007; Wanner et al., 2010). Brièvement, le premier niveau se charge de récupérer les informations sur les concentrations de polluants, puis les analyse et les interprète. Le deuxième niveau créé et transmet le message contenant l'information interprétée à chaque requête utilisateur. La Figure 9 présente l'organisation de MARQUIS et été adpatée de (Wanner et al., 2010).

MARQUIS collecte les données périodiquement mais ne les interprète qu'à la réception d'une requête utilisateur afin de sélectionner au mieux le contenu du message à transmettre. Une fois les données interprétées en accord avec le profil utilisateur, celles-ci sont envoyées au module de planification de document. Ce dernier produit un plan de document intégrant les unités d'information composant le message. Ces unités sont liées entre elles en fonction de leur thématicité, de la structure discursive du message, etc. Ce plan de document est ensuite envoyé au module de génération de surface, sur lequel nous allons nous focaliser.

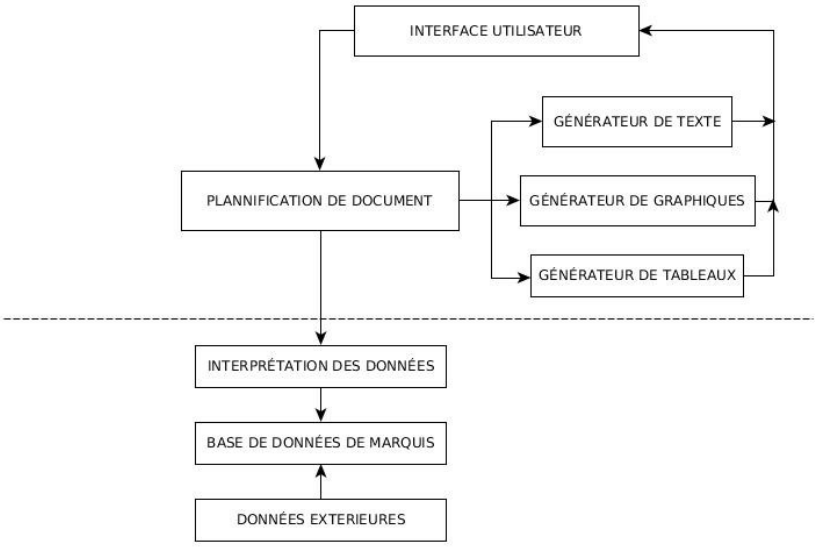



Figure 9. Architecture générale de MARQUIS en deux niveaux

La réalisation de texte opérée par MARQUIS est complète. Elle part d'une structure très abstraite et exploite les principes de la Théorie Sens-Texte pour structurer le processus de réalisation ainsi que les ressources employées (Lareau & Wanner, 2007). La Théorie Sens-Texte se prête bien à la génération automatique de texte car elle offre un modèle linguistique fonctionnel multistratal permettant à la fois de choisir le degré d'abstraction de départ tout en gardant les ressources modulaires et simples. (L. Iordanskaja et al., 1992; Lidija Iordanskaja et al., 1996) structurent également le processus de réalisation linguistique à partir des représentations linguistiques énoncées en Théorie Sens-Texte. Chaque phénomène linguistique peut être traité indépendamment dans le module le plus approprié.

Le module de génération de surface de MARQUIS s'articule autour de six niveaux : conceptuel, sémantique, syntaxique profond, syntaxique de surface, morphologique (profond et de surface) et textuel. Pour passer d'un niveau à l'autre, MARQUIS effectue une séquence de transductions permettant de fournir une correspondance entre une structure de niveau *i* et une ou plusieurs structure(s) du niveau adjacent *i+1*. Ces séquences de transductions sont organisées en règles qui constituent la grammaire de MARQUIS. Les structures linguistiques ne sont pas représentées de la même manière dans chaque niveau, voir la Figure 10 (Lareau & Wanner, 2007, p. 207). Les structures du niveau conceptuel forment des graphes conceptuels au sens de (Sowa, 2000). Au niveau sémantique, elles s'articulent en graphes orientés acycliques

représentant des prédicats sémantiques et leurs actants. En syntaxe profonde et de surface, il s'agit d'arbres de dépendance. Les niveaux morphologiques présentent des structures linéarisées. Enfin, le niveau textuel contient le texte complet à transmettre à l'utilisateur.

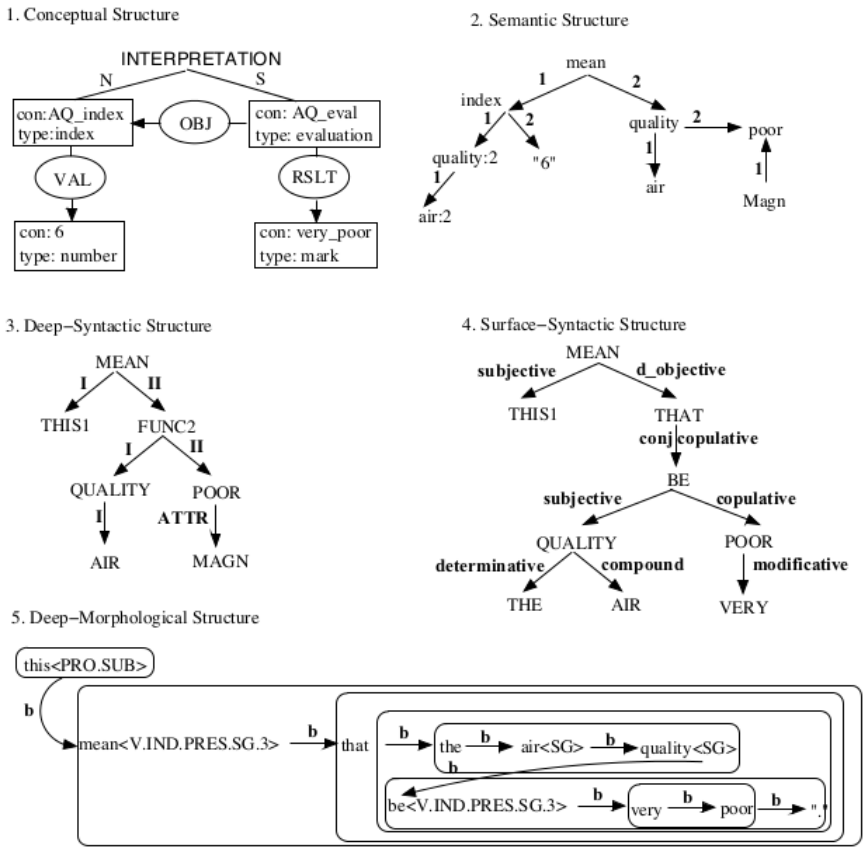



Figure 10. Les différents niveaux de représentation linguistique de MARQUIS

Afin d'effectuer les séquences de transduction entre chaque niveau, MARQUIS se base sur la plateforme MATE, un transducteur de graphes, ainsi que sur plusieurs dictionnaires.

### 3.1.2 MATE et l'encodage des informations linguistiques

Les transducteurs de graphes sont utilisés dans plusieurs applications en TAL, notamment en GAT (Bohnet & Wanner, 2010). MATE est un transducteur de graphes libre conçu par Bernd Bohnet (Bohnet, 2006; Bohnet, Langjahr, & Wanner, 2000; Bohnet, Lareau, Wanner, et al., 2007; Bohnet & Wanner, 2010) conçu à la base pour modéliser l'approche multistratale de la TST. Il manipule divers types de graphes et construit, à partir d'un graphe source, un graphe correspondant d'un niveau supérieur, sans modifier la structure de départ.

MATE fournit également une plateforme de développement et de maintien de générateurs de texte employant la TST (Bohnet et al., 2000). Pour passer d'un graphe à l'autre, MATE applique des règles et forme le graphe de sortie par un procédé d'unification. Nous ne rentrerons pas dans les détails techniques du fonctionnement de MATE et nous encourageons le lecteur intéressé à se référer à (Bohnet & Wanner, 2010).

En plus d'être un transducteur de graphe, MATE est aussi une plateforme de développement et de maintien de ressources à large couverture. (Bohnet et al., 2000) indiquent que MATE permet d'écrire, modifier et tester à la fois des dictionnaires et des grammaires. Il permet une structuration hiérarchique de ces ressources grâce à un système de classes. Chaque type de ressource possède son propre éditeur. MATE permet aussi de créer des structures à différents niveaux d'abstraction grâce à un éditeur dédié. MATE offre la possibilité de visualiser les différents graphes et fournit un inspecteur pour examiner le processus de correspondance lors de l'application des règles. L'inspecteur montre les correspondances établies entre les nœuds, l'instanciation des variables des règles appliquées et une trace de toutes les opérations effectuées pendant la compilation, ce qui favorise le développement et le maintien de larges grammaires. Nous allons maintenant nous attarder sur l'encodage des dictionnaires et des grammaires dans MATE.

#### 3.1.2.1 Dictionnaires

(Lareau & Wanner, 2007) indiquent que les dictionnaires de MARQUIS englobent deux types d'information: (i) l'information sur le lexique et (ii) l'information sur les correspondances entre les entrées du lexique et leur équivalent dans la strate adjacente. C'est pourquoi il existe quatre dictionnaires principaux dans ce système: un dictionnaire d'unités conceptuelles, un dictionnaire d'unités sémantiques (*semanticon* ci-après), un dictionnaire d'unités lexicales (*lexicon* ci-après) et un dictionnaire de fonctions lexicales (*lf* ci-après).

Les dictionnaires sont essentiels au bon fonctionnement du module de génération de surface de MARQUIS. Chaque séquence de transduction récupère l'information contenue dans les différents dictionnaires, comme le montre la Figure 11. Nous présentons maintenant chacun de ces dictionnaires.

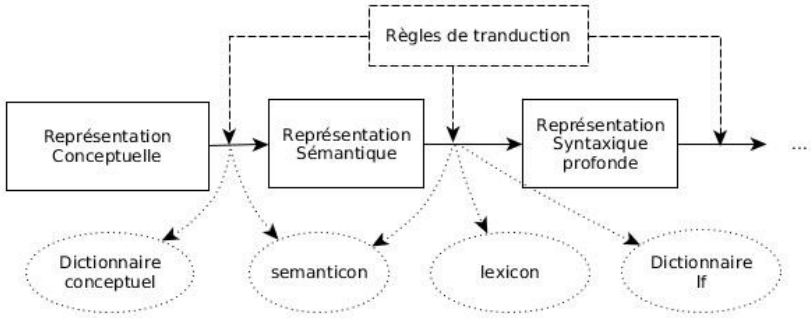



Figure 11. Interdépendance des règles et dictionnaires dans MARQUIS et GÉCO

Le dictionnaire conceptuel englobe l'ensemble des concepts et leurs correspondants sémantiques dans les différentes langues. En effet, le module de génération profonde de MARQUIS représente le contenu du message à l'aide de graphes conceptuels. Lors du passage entre la structure conceptuelle du message (RCon) vers son équivalent sémantique (RSem), MARQUIS récupère les correspondants sémantiques des nœuds de la RCon dans le dictionnaire conceptuel. Ce processus est aussi appelé « sémanticisation ». Voici un exemple d'une entrée du dictionnaire conceptuel, tiré de (Lareau & Wanner, 2007).

(63) Dictionnaire conceptuel

```
Concentration : property_attribute {
  sem = 'concentration'
}
```

MATE structure les dictionnaires par un système de classes. Ici, le concept *concentration* est une instance de la classe *property_attribute*. De même, les unités de chaque dictionnaire sont décrites à l'aide de traits sous forme attributs-valeur, ces traits pouvant être regroupés en matrices de traits (cf. attribut « sem »). Dans l'exemple (63), *concentration* possède un attribut « sem » dont la valeur renvoie au prédicat sémantique correspondant, 'concentration'. À partir de cette entrée, MATE est capable de sémanticiser le concept *concentration* en un prédicat sémantique. Ce mémoire porte sur l'aspect linguistique de la GAT. Or ce dictionnaire et l'interface RCon ⇔RSem sont en dehors du champ linguistique à proprement parler de la GAT. Nous n'avons pas inclus une étude approfondie de cette étape dans notre recherche.

Le *semanticon* regroupe tous les sémantèmes d'une langue donnée et indique l'ensemble de leur(s) correspondant(s) lexicaux possibles. Il existe donc un *semanticon* par langue dans

MARQUIS. En effet, il arrive que certains sens ne s'organisent pas de la même manière dans plusieurs langues (cf. les exemples (56) et (57) tirés du LFS à la section 2.2.3.2 du Chapitre 2). L'objectif du *semanticon* est de gérer la lexicalisation des sens d'un énoncé lors du passage entre la structure sémantique (RSem) vers la structure syntaxique profonde (RSyntP) de cet énoncé.

(64) Semanticon

```
concentration {
  lex = concentration
}
augmentation {
  lex = augmenter
  lex = augmentation
}
```

L'exemple (64) montre que le *semanticon* regroupe toutes les lexicalisations possibles pour chaque entrée. Le processus de génération dans MATE est donc non-déterministe, c'est-à-dire que pour chaque structure $S_i$ du niveau de représentation $R_i$, MATE fournit toutes les structures équivalentes possible $S_{j,..n}$ dans le niveau de représentation adjacent $R_{i+1}$. Cependant, un générateur ne transmet qu'un texte à l'utilisateur. Il existe plusieurs méthodes pour sélectionner une structure appropriée. Or, cet aspect de la GAT sort du champ de ce mémoire et ne sera donc pas abordé ici.

Le *lexicon* est un dictionnaire riche regroupant toutes les unités lexicales (UL) d'une langue donnée. Chaque entrée correspond à une UL et est structurée autour de trois types majeurs d'information, voir exemple (65) : (i) ses caractéristiques grammaticales, (ii) son régime et enfin (iii) l'inventaire de ses fonctions lexicales. Ces trois types d'information nous renseignent sur ses propriétés combinatoires.

(65) Lexicon

```
concentration : noun {
  // (i) grammatical characteristics :
  dpos = N
  spod = common_noun
  // (ii) government pattern (subcategorization):
```

```
  gp = {
    // Sem-Dsynt valency projection (1=>I, 2=>II):
    1 = I
    2 = II
    // First syntactic actant can be realized as « ozone
concentration »:
    I = {
        dpos = N
        rel = compound
        det = no
    }
    // First syntactic actant can be realized as « concentration of
ozone »:
    I = {
        dpos = N
        rel = noun_completive
        prep = to
        det = no
    }
    // Second syntactic actant can be realized as « concentration of
180ug/m3 »:
    II = {
        dpos = Num
        rel = noun_completive
        prep = of
    }
  }
  // (iii) Lexical functions:
  Magn = high
  AntiMagn = low
  Adv1 = in // « (we found) ozone in a concentration (of 180 ug/m3) »
  Func2 = be // « the concentration (of ozone) is 180 ug/m3 »
  Oper1 = have // « ozone has a concentration (of 180 ug/m³) »
  IncepFunc2 = reach // « the concentration (of ozone) reached 180
ug/m3 »
  IncepOper1 = reach //ozone will reach a concentration of 180 ug/m3 »
}
```

Les caractéristiques grammaticales indiquent la partie du discours profonde (catégories lexicalement pleines universelles telles que nom, verbe, adjectif, adverbe) et de surface de l'UL. Le régime informe sur les relations entre l'UL et ses arguments. Ainsi, on sait que les actants sémantiques 1 et 2 de l'exemple (65) se réalisent respectivement par les relations syntaxiques profondes I et II. Ces arguments présentent un certain nombre de caractéristiques. Par exemple, l'argument I de CONCENTRATION peut se réaliser de deux manières, ce qui a une incidence sur sa relation avec l'UL.

Il est important de noter que ce traitement de la lexicalisation diffère des travaux de (L. Iordanskaja et al., 1992; Lidija Iordanskaja et al., 1996). Ces derniers lexicalisent lors du passage entre RSem et RSemR (représentation sémantique réduite). Ensuite, l'interface RSem ⇔ SyntP arborise les relations entre lexèmes. Dans le cas de Géco et Marquis, la lexicalisation et l'arborisation sont traitées de concert, notamment grâce aux entrées riches du *lexicon*.

Enfin, la combinatoire restreinte (collocations) des UL sont intégrées à l'entrée à l'aide de son inventaire de fonctions lexicales. Cependant, indiquer que CONCENTRATION possède une fonction lexicale Magn dont la valeur est HIGH n'est pas suffisant pour construire la collocation complète *high concentration*. En effet, dans MARQUIS et GÉCO, chaque FL s'instancie par un patron spécifique, à la manière de (Lareau et al., 2011, 2012). Par exemple, tous les Magn se réalisent par un modificateur adjectival ou adverbial. Ces informations de régime sont indiquées dans le dictionnaire *lf*.

(66) Dictionnaire *lf*

```
verb {
  dpos = V
  gp = {
     I = { dpos = N }
     II = { dpos = N }
     II = { dpos = N  }
  }
}
Func: verb {
  gp = {
      L = I
```

```
  }
}
Mod {
  dpos = A
  gp = {
      L =ATTR
  }
}
Magn: Mod {}
Ver: Mod {}
```

Ce dictionnaire est commun à toutes les langues. Encore une fois, le dictionnaire est structuré en classes. En (66), Func et Labor est une fonction lexicale organisée sous la classe « verb ». La relation entre la base et le collocatif en RSyntP est marquée par l'attribut « L » dont la valeur correspond à la position syntaxique profonde de la base par rapport au verbe support. Cet exemple présente aussi la classe « Mod » regroupant plusieurs fonctions lexicales dont Magn et Ver. Cette classe décrit des collocatifs adjectivaux entretenant une relation attributive (ATTR) avec la base.

Il est possible de créer autant de dictionnaires que nécessaire afin de modéliser tout type de phénomène linguistique. Ces informations sont par la suite récupérées lors de l'application des règles de grammaire, comme nous allons le voir tout de suite.

#### 3.1.2.2 Grammaires

Dans MARQUIS, la réalisation linguistique part d'une structure conceptuelle constituée d'un graphe conceptuel au sens de (Sowa, 2000). MATE fournit la forme linéarisée textuelle correspondant à ce graphe en plusieurs étapes de transduction, comme cela a été décrit précédemment. L'ensemble de ces règles forme la grammaire de la langue.

Les règles sont tripartites et contiennent une partie gauche, une partie droite et des conditions d'application de la règle. La partie gauche représente le graphe de départ. Il ne s'agit pas nécessairement d'un graphe complet. Autrement dit, la partie gauche se focalise sur les nœuds et les arcs pertinents pour l'application de la règle.

La partie droite représente le graphe de sortie dans le niveau adjacent. Encore une fois, il ne s'agit pas nécessairement d'un graphe complet: seuls les nœuds, arcs et traits pertinents sont créés. Ils seront unifiés par la suite avec le reste du graphe construit par d'autres règles. Enfin, un ensemble de conditions limite le champ d'application des règles. Le Tableau IV présente la règle de lexicalisation simple incluse dans l'interface RSem ⇔ RSyntP de GÉCO.

| **Partie Gauche** | **Patrie Droite** |
|---|---|
| `l:?Xl{}`<br>`?L <- semanticon::(?Xl.sem).(lex)` | `rc:?Xr{ <=> ?Xl`<br>`dlex=?L`<br>`}` |
| **Conditions d'application** | |
| `semanticon::(?Xl.sem).lex;` | |

Tableau IV Règle de lexicalisation simple

En théorie, cette règle permet de récupérer n'importe quel nœud de la RSem et de créer son équivalent lexicalisé dans la RSyntP. Nous illustrons le fonctionnement de la procédure de lexicalisation simple du sémantème 'chat':

La partie gauche contient un nœud, `?Xl,` celui que l'on cherche à lexicaliser. Les arguments de ce nœud ne sont pas précisés, comme l'indique l'accolade vide `{}`. Ce nœud est une variable, qui dans notre exemple va s'instancier sur un nœud étiqueté 'chat'. La règle bloque ce nœud à l'aide de l'instruction `l:?Xl`. Autrement dit, la règle rend `?Xl` indisponible pour d'autres règles. En plus de ce nœud, la partie gauche crée une variable `?L` et fait un appel au dictionnaire *semanticon* par le biais de l'expression `semanticon::(?Xl.sem).(lex)`. Cette expression est un chemin permettant de fouiller le *semanticon* à la recherche de l'entrée pour le sémantème qui étiquette le nœud `?Xl` (retournée par `?Xl.sem`) et de récupérer sa lexicalisation (valeur de l'attribut `lex`). De ce fait, `?L` récupère CHAT, soit la valeur de l'attribut `lex` dans l'entrée `chat` du *semanticon*.

Dans la partie droite, la règle définit un contexte de droite `rc:`. Ce contexte va s'instancier sur un nœud existant dans le graphe de sortie, `?Xr`. Ensuite, la règle indique que `?Xr` est le

correspondant du nœud `?Xl`, ce qui est marqué par `<=>?Xl`. La règle créé un attribut `dlex` à `?Xr` dont la valeur est la lexicalisation de `?Xl`. Le nœud `?Xr` a donc `CHAT` comme valeur à pour son attribut `dlex` et est la correspondance lexicale au sémantème 'chat'.

Enfin, La règle de lexicalisation simple possède une condition d'application, exprimée par « `semanticon::(?Xl.sem).lex` ». Cela vérifie que le sémantème qui étiquette `?Xl` possède bien une lexicalisation dans le *semanticon*.

Les règles de transduction sont regroupées en modules. Ces modules représentent chaque interface pour passer d'un niveau de représentation à un autre. Au sein d'un module, les règles sont regroupées en paquets afin de faire hériter des conditions d'application à l'ensemble des règles de chaque paquet. Il existe six modules de règles sous MARQUIS, référencés dans la Figure 12 tirée de (Lareau & Wanner, 2007, p. 215). L'objectif des développeurs de MARQUIS était de généraliser le plus de règles possible afin de pouvoir les partager et les appliquer entre chaque langue. C'est le principe de « partage des ressources » tel que présenté à la partie 1.1.2. du Chapitre 1. Le nombre de règles génériques dépend du degré d'abstraction : plus le niveau est abstrait, plus il y a de règles génériques.

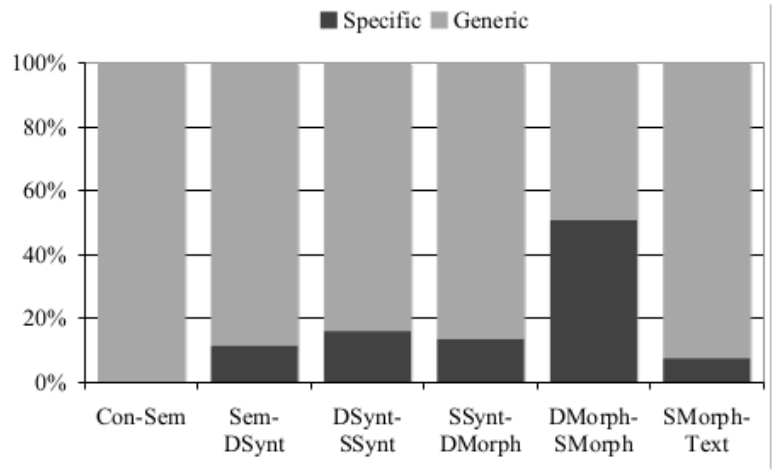



Figure 12. Ratio moyen de règles génériques et spécifiques pour chaque module (Lareau & Wanner, 2007, p. 215)

Grâce à cette architecture, il est possible de partager beaucoup de règles entre les langues traitées. L'architecture de la grammaire de MARQUIS est donc optimisée pour la génération

automatique de texte multilingue. De même, sa modularité facilite la création de nouvelles règles ainsi que l'intégration de nouvelles langues similaires dans le système.

La Figure 12 présente le ratio moyen de règles génériques et spécifiques par module. MARQUIS compte environ 200 règles. Il n'existe aucune règle spécifique dans l'interface entre le niveau conceptuel et sémantique. À part le module DMorph-SMorph, le pourcentage de règles spécifiques n'excède jamais 16 %. Autrement dit, plus de 85 % des règles de chaque module sont génériques et donc utilisées par toutes les langues du système. Le module DMorph-SMorph contient très peu de règles, entre 8 et 12 par langue, ces règles ne servant qu'à préparer la chaîne de caractère pour le traitement morphologique du niveau suivant.

Ce mémoire se focalise sur le module d'interface RSem ⇔ RSyntP dans GÉCO. Ce module est composé de deux grands paquets : *core* et *LexicalFunctions*. Dans sa version finale, GÉCO inclura plus de paquets en fonction des phénomènes linguistiques traités. Le paquet *core*, hérité de MARQUIS, possède des règles de construction d'arbres syntaxiques. Ces règles créent la racine de l'arbre syntaxique profond et instancient les relations de dépendance entre les prédicats et leurs dépendants. Ce paquet contient également les règles de lexicalisation de base (voir Tableau IV). Le paquet *LexicalFunctions* regroupe tous les patrons de fonctions lexicales. MARQUIS possède 16 règles de transduction de fonctions lexicales qui permettent de générer une trentaine de patrons de collocations. La plupart sont des verbes supports.

Maintenant que nous avons vu en détail l'architecture de MARQUIS, la plateforme MATE ainsi que les stratégies de développement de ressources, nous allons présenter la méthodologie d'implémentation des fonctions lexicales dans GÉCO.

## 3.2 Méthodologie d'implémentation des collocations

L'implémentation a suivi une méthodologie en trois étapes: l'identification des régularités parmi les diverses collocations qui existent dans les langues, la modélisation de ces régularités et leur intégration dans GÉCO via la plateforme MATE.

### 3.2.1 Description des régularités

Le précédent chapitre a mis en évidence l'intérêt tant pratique que théorique des fonctions lexicales (FL) de la Théorie Sens-Texte comme outil de description des diverses collocations qui existent dans les langues.

MARQUIS implémentait déjà une trentaine de FL. Elles sont intégrées à l'énoncé lors de la phase de lexicalisation, c'est-à-dire dans l'interface RSem ⇔ RsyntP. Les FL sont modélisées au moyen de 16 règles de transduction. La plupart de ces FL introduisent un verbe support ($Func_0$, $Oper_0$, $Labor_{ij}$, $IncepFunc_0$, etc.) en RSyntP. Le reste introduit Magn, un modificateur adjectival, et $Adv_i$, un modificateur adverbial de l'actant i de la base. Ces FL sont standard, syntagmatiques simples, mis à part $IncepFunc_i$ qui dénote une FL complexe. Enfin, MARQUIS intègre une configuration de FL, $Magn.Oper_i$ en l'occurrence. Les valeurs des FL sont intégrées dans l'interface RSyntP ⇔ RSyntS. Le Tableau V présente la liste des FL couvertes par MARQUIS :

| | | | |
|---|---|---|---|
| $Func_0$ | $IncepFunc_1$ | $Labor_{23}$ | $Oper_2$ |
| $Func_1$ | $IncepFunc_2$ | $Labor_{31}$ | $Oper_3$ |
| $Func_2$ | $IncepFunc_3$ | $Labor_{32}$ | $Magn.Oper_1$ |
| $Func_3$ | $Labor_{12}$ | Magn | $Oper_{12}$ |
| $Adv_1$ | $Labor_{13}$ | $Oper_0$ | $Oper_{13}$ |
| $Adv_2$ | $Labor_{21}$ | $Oper_1$ | $Oper_{21}$ |
| $Oper_{23}$ | $Oper_{31}$ | $Oper_{32}$ | $Magn.Oper_2$ |

Tableau V FL implémentées dans MARQUIS

Notre implémentation vise à continuer l'intégration de ces types de FL dans GÉCO de manière exhaustive, en nous focalisant sur les FL standard. Nous laissons de côté les FL non standard car elles dénotent des relations extrêmement spécifiques et non systématiques (I. Mel′čuk et al., 2015); leur comportement est par définition peu généralisable. Parmi les FL standard, nous

incorporons celles au caractère syntagmatique, qu'elles soient simples ou complexes. Nous ne traiterons pas non plus des configurations de FL car il s'agit de combinaisons de plusieurs FL non reliées syntaxiquement, rendant leur mécanisme de formation difficile à modéliser.

Nous ne pouvons pas intégrer les FL une par une dans GÉCO. En effet, même si la littérature estime à soixante le nombre de FL standard simples, il n'existe pas de décompte des FL standard syntagmatiques, simples et complexes; nous soupçonnions que ce nombre serait trop élevé pour les traiter individuellement. Nous devons donc regrouper les FL similaires sous des patrons, ou classes, génériques, de telle manière à généraliser leur comportement. Cette stratégie, proposée par (Heid & Raab, 1989), a été implémentée dans (Lareau et al., 2011) et MARQUIS. C'est ce que nous présentons maintenant.

### 3.2.2 Modélisation des fonctions lexicales

MARQUIS suit les principes du partage de grammaire (voir section 1.1.2 du chapitre 1). Il établit des règles de l'interface sémantique-syntaxe abstraites, génériques et capables de traiter plusieurs langues. Le nombre de règles spécifiques à une langue donnée est limité. C'est dans cette optique d'optimisation que nous avons généralisé le traitement des FL à l'aide de patrons de FL plutôt que de traiter chaque FL séparément. La création de patrons généraux, des classes de FL partageant des propriétés similaires, s'insère dans une perspective de développement continu de notre plateforme et vise à faciliter l'implémentation de nouvelles FL en plus du maintien des structures existantes (Lambrey & Lareau, 2015). L'intérêt de créer des patrons est qu'ils font abstraction de la nature exacte des FL traitées et fonctionnent comme un modèle de lexicalisation plus général encore.

Nous avons modélisé la représentation et le fonctionnement des FL sous forme de règles de correspondance entre le niveau sémantique et le niveau syntaxique profond. Comme nous l'avons mentionné dans le Chapitre 2, les noms des FL s'intègrent normalement dans la représentation syntaxique profonde de l'énoncé. La structure sémantique ne représente que les sens et relations exprimés par la FL. Par exemple, la Figure 13 décrit la réalisation en syntaxe profonde des collocations *peur bleue*, *brouillard dense*, *heavy smoker*, tous étant des exemples de la fonction lexicale Magn, telle qu'elle est prévue dans la Théorie Sens-Texte.

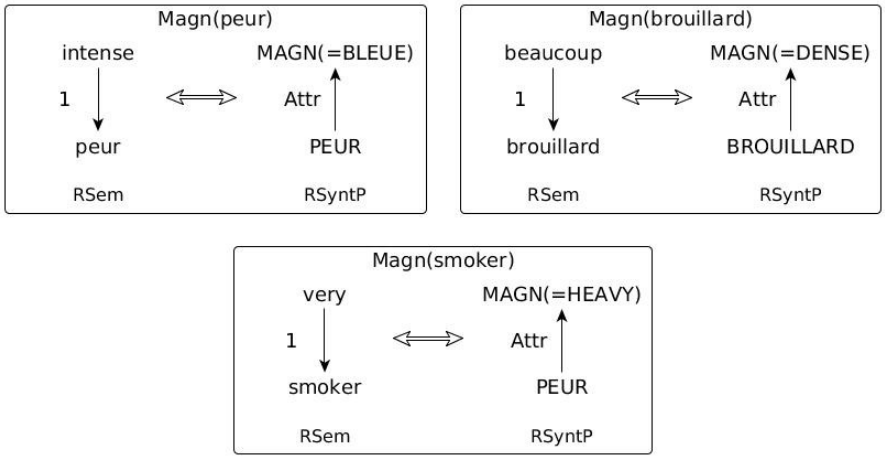



Figure 13. Exemples de Magn

Magn est introduite par un sens spécifique dans chaque structure, 'intense', 'beaucoup' et 'very'. Chaque sens se rapportant plus ou moins au sens plus général d'intensification. Malheureusement, il n'est pas possible de prévoir tous les sémantèmes possibles se lexicalisant par Magn, en particulier dans une perspective multilingue. Pour pallier à ce problème, nous avons employé directement le nom des FL dans la RSem, comme le montre la Figure 14. De même, nous introduisons directement les valeurs des FL en RSyntP afin de faciliter le traitement des spécificités de ces unités lexicales dans leur emploi en tant que collocatif. Le fait d'introduire les valeurs le plus tôt possible suit les recommandations de Iordanskaja et al. (1996).

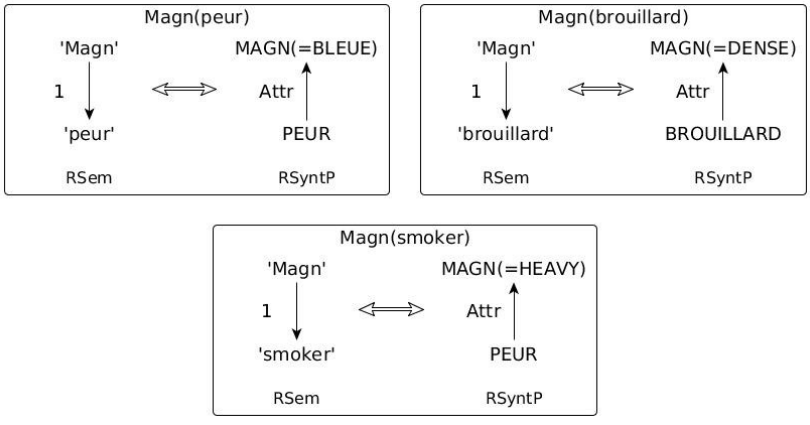



Figure 14. Généralisation du fonctionnement de Magn pour GÉCO

Les structures de la Figure 14 fonctionnent de la même manière. Cela nous permet de modéliser le comportement générique de Magn. La modélisation générique du comportement de Magn correspond donc à la Figure 15. La base de la collocation est représentée par la variable x en RSem et par X en RSyntP. Y est une variable correspondant à la valeur de la collocation Magn dans le cas de Magn(X)=Y.

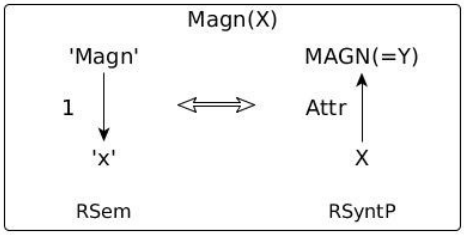



Figure 15. Modélisation générique de Magn dans GÉCO

Plusieurs FL ont un fonctionnement similaire à Magn. C'est le cas de la FL Ver, dont le comportement peut aussi être généralisé, voir Figure 16. Magn et Ver ont donc été regroupées sous un même patron, en l'occurrence le patron Modification.

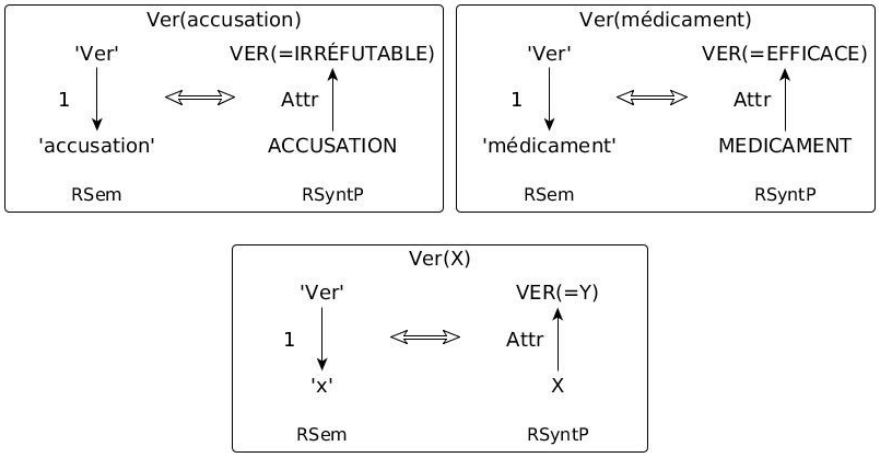



Figure 16. Généralisation du comportement de Ver

Nous avons commencé par modéliser et regrouper les fonctions lexicales standard syntagmatiques simples. Cependant, il n'existe pas de liste standard de FL. Plusieurs listes sont présentées dans divers travaux (Jousse, 2003, 2010; I. Mel′čuk et al., 1999). Nous nous basons principalement sur les listes de FL standard simples fournie dans (I. Mel′čuk, 1996; I. Mel′čuk et al., 1999, 1995). Nous avons retiré les FL paradigmatiques de cette liste, ce qui nous amène

à traiter 40 FL standard syntagmatiques simples, en incluant Gener dans son fonctionnement syntagmatique (voir exemple (67)).

(67) Gener(colère) = sentiment de ~

En ce qui concerne l'implémentation des FL complexes, Mel'cuk et al. (1995, 1996, 1999) fournissent des exemples de combinaisons de FL complexes. Il existe trois types de combinaisons possibles :

- La plupart des FL complexes sont constituées d'une FL introduisant un collocatif verbal, par exemple Real, à laquelle se rajoute un élément phasique, Incep par exemple, et/ou causatif;
- La plupart des FL se combinent également avec les FL Anti et Non;
- Enfin, $A_i$ peut s'ajouter à une FL existante, simple ou complexe, afin de faire une dérivation adjectivale du collocatif.

Nous avons modélisé les FL complexes à l'aide de la représentation sémantique fournie par (Kahane & Polguère, 2001) dans l'encodage explicite. Autrement dit, les FL complexes sont modélisées comme des compositions sémantiques. La Figure 17 présente la modélisation en RSem des FL complexes des exemples (68) et (69):

(68) AntiBon(choix) = mauvais ~

(69) $CausReal_1$(abîme) = projeter dans ART ~

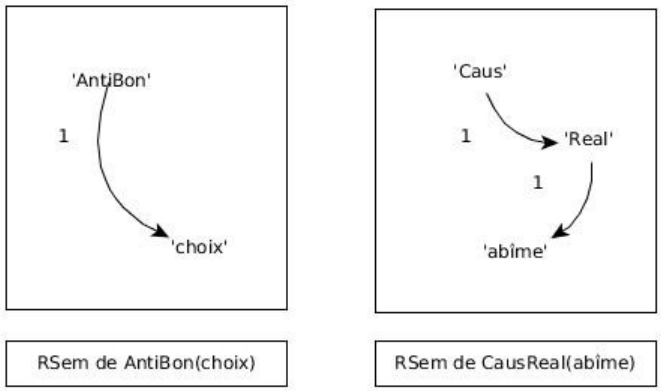



Figure 17. Représentation sémantiques de FL complexes dans GÉCO

Les principes de formation de FL complexes s'appliquent aux patrons et non à chaque FL individuellement. Autrement dit, toutes les FL du patron Modification présentent les mêmes

schémas de combinaison pour former des FL complexes. De ce fait, Magn, Ver et Pos se combinent de la même manière pour former AntiMagn, AntiVer et AntiPos. Il s'agit là d'un choix que nous avons fait pour faciliter le traitement des FL complexes. Nous verrons lors de l'évaluation s'il s'avère que les FL appartenant à un même patron se combinent effectivement de la même manière.

Nous ne nous sommes pas servis de ressources existantes pour monter notre inventaire de FL simples et complexes. En effet, les ressources comme DiCoInfo (L'Homme, 2009), etc., possèdent leur méthodologie propre d'annotation de FL et adoptent une perspective lexicographique, plutôt que computationnelle. Elles sont aussi développées dans le cadre d'un domaine de connaissance spécifique, ce qui restreint leur couverture. Nous avons employé une méthode « logique » et déductive basée sur les exemples de référence de (I. Mel′čuk, 1996; I. Mel′čuk et al., 1999, 1995) pour la modélisation des FL simples et création des patrons de combinaisons de FL. Enfin, nous n'avons pas modélisé les configurations de FL dans le cadre de ce mémoire. En effet, contrairement aux FL complexes, que l'on peut décrire à l'aide d'une composition sémantique, les configurations de FL se modélisent différemment en RSem. Les exemples (70) et (71) présentent deux cas de combinaisons de FL et la Figure 18 présente leur RSem.

(70) AntiBon.AntiMagn(bagage) = ~ maigre

(71) $\text{Magn.Oper}_1$(enthousiasme) = déborder de ~

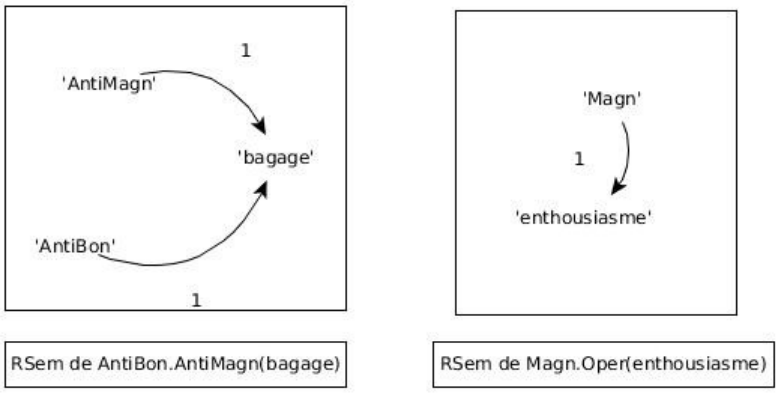



Figure 18. Configurations de FL dans GÉCO

Comme on peut le voir, il n'existe pas de relation entre les composantes des configurations de FL présentées à la Figure 18 pour l'exemple (70). Cependant, formellement, il n'est pas possible de différencier une FL complexe d'une configuration de FL lorsqu'il s'agit d'un verbe support car ils sont sémantiquement vides. Les configurations de FL représentent donc un défi pour la GAT car leur mode de combinaison semble aléatoire. Le projet GÉCO prévoit de rajouter ces FL par la suite, mais cet aspect du travail déborde du cadre de notre mémoire.

### 3.2.3 Intégration des FL dans MATE

Comme le mentionnent (Wanner & Lareau, 2009, p. 211-213) :

> Although SemS is langage-specific, it is generic enough to be isomorphic for many utterances in similar languages. (…) Compared to SemS, DsyntS is considerably more language specific, although it is abstract enough to level out surface-oriented syntactic idiosyncrasies.

Autrement dit, malgré que les RSem soient spécifiques à une langue donnée, elles demeurent structurellement très similaires, ce qui favorise le développement de règles génériques pouvant s'appliquer sur des RSem de langues différentes. Par contre, les spécificités de chaque langue apparaissent dans les RSyntP. Il faut donc intégrer les collocations lors du passage entre la RSem et la RSyntP. Comme cela a été mentionné dans la section précédente, MARQUIS (Lareau & Wanner, 2007; Wanner, Nicklaß, et al., 2007), intègre déjà plusieurs collocations à l'aide de règles de transduction dans l'interface RSem ⇔ RSyntP.

De même, dans un article portant sur le travail de ce mémoire, (Lambrey & Lareau, 2015) indiquent que lors du processus de lexicalisation, les règles de transduction de GÉCO activent l'information contenue dans les dictionnaires. Il est donc nécessaire de développer des dictionnaires détaillés fournissant une analyse fine des unités lexicales de chaque langue, notamment leurs propriétés combinatoires, et d'optimiser leur structuration pour accéder facilement à ces données. C'est dans cette optique que deux dictionnaires ont particulièrement été retravaillés: le dictionnaire de fonctions lexicales (*lf* ci-après) et le dictionnaire d'unités lexicales (*lexicon* ci-après). Nous présentons chacun de ces aspects maintenant.

#### 3.2.3.1 Dictionnaires

MARQUIS possédait déjà un dictionnaire *lf*, cependant nous avons évidemment décidé d'y rajouter beaucoup plus de FL, notamment des FL complexes. Cela amène à repenser l'organisation du dictionnaire en classes et à intégrer de l'information sémantique afin de traiter les FL explicitement (Kahane & Polguère, 2001). Nous avons donc intégré un attribut « sem » pour chaque FL, ou chaque patron de FL, comme l'exemple (72) le montre :

(72) Extrait du dictionnaire *lf*:

```
lf{
verb {
  dpos = V
  gp = {
     I = { dpos = N }
     II = { dpos = N }
     II = { dpos = N  }
  }
}

// Verbes supports
Oper : verb {
  gp = { L = II }
}
Oper0 : Oper {}
...
// Verbes de réalisation
Real0 : Oper0 { sem=Real }
...
}
```

Dans cet extrait, une classe « verb » de FL est définie. Cette classe contient différents attributs, dont une « dpos » (partie du discours profonde) et son « gp » (patron de régime). Cette classe décrit trois arguments en RSyntP (I, II et III) et impose des contraintes sur leur partie du discours.

Cet extrait présente également deux regroupements de FL : les verbes supports et les verbes de réalisation. La FL Oper est un verbe support et est une sous-classe de « verb ». Son patron de

régime « gp » inclut un élément L dont la valeur est II. Cela indique que la base (L) se réalise comme l'actant syntaxique profond II du collocatif introduit par une FL Oper. Il existe plusieurs types de Oper, notamment $Oper_0$. Le patron des verbes de réalisation regroupe plusieurs FL, dont $Real_0$. Le dictionnaire *lf* indique que Real est une sous-classe de $Oper_0$, c'est-à-dire que cette FL présente les mêmes propriétés que $Oper_0$. En théorie, Real n'est pas un sous-type de Oper. Cependant, cette architecture évite la redondance de description des FL. De ce fait, la seule différence existant entre $Oper_0$ et $Real_0$ est que $Real_0$ est sémantiquement pleine : elle possède un attribut « sem » représentant son sens, ici 'Real'.

Le dictionnaire *lf* regroupe à la fois les propriétés combinatoires des FL ainsi que leurs propriétés sémantiques. Cela nous permet d'encoder leur fonctionnement explicitement, à la (Kahane & Polguère, 2001).

En ce qui concerne le *lexicon*, la description des fonctions lexicales est différente de l'approche de MARQUIS. Pour chaque entrée d'unité lexicale, on insère un nœud enfant « lf », lui-même contenant deux attributs, « name » et « value ». Cette structuration, certes redondante, sert à faciliter la récupération des informations lors de l'application des règles de transduction, (cf. section suivante). Lorsque cela est nécessaire, le nœud « lf » peut également être parent d'un nœud « gp » contenant les spécificités d'emploi du collocatif, par exemple la contrainte de l'indéfinitude. Voici un extrait simplifié du *lexicon* :

(73) Extrait du *lexicon* :

```
fumée : noun {
  lf = {
    name = Figur
    value = rideau
  }
}
```

L'entrée du lexème FUMÉE est une instance de la classe « noun ». Elle contient une FL (« lf ») dont le nom (« name ») est Figur et la valeur (« value ») est RIDEAU.

### 3.2.3.2 Règles de transduction

Un patron, par exemple le patron Modification, se traduit par une règle de transduction dans MATE opérant dans l'interface RSem ⇔ RSyntP. Nous allons décrire le fonctionnement général d'un patron de FL dans GÉCO. Nous nous basons sur l'exemple suivant:

(74) Magn(fumeur) = gros ~

L'exemple (74) est un Magn faisant partie du patron Modification. La règle de base de ce patron est représentée dans la Figure 19. Voici comment fonctionne cette règle :

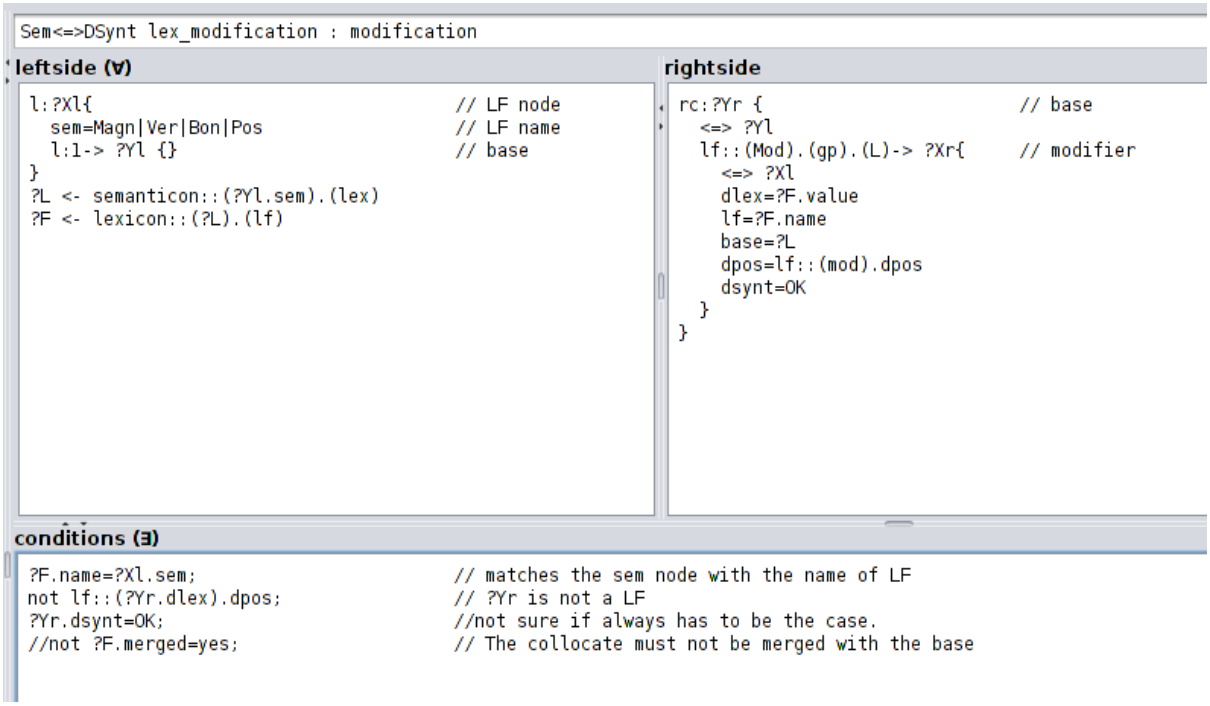



Figure 19. Capture d'écran de la règle « lex_modification » encodée dans MATE

La partie gauche spécifie deux nœuds, `?Xl` et `?Yl`. De même, `?Yl` est l'actant sémantique 1 de `?Xl`. `?Xl` représente le nœud qui est étiqueté par le sens d'une FL en RSem, et peut s'instancier sur un nœud Magn, Ver, Bon ou Pos dans cette règle. Dans le cas de l'exemple (74), il s'agit de Magn. `?Yl` s'instancie sur le nœud étiqueté 'fumeur'. De ce fait, la partie gauche de la règle correspond au graphe de la Figure 20.

La partie gauche déclare deux variables, `?L` et `?F`. Leurs valeurs sont récupérées à l'aide d'un appel aux dictionnaires. Dans le cas de `?L`, l'appel fouille dans le *semanticon* (`semanticon ::`) à la recherche de l'entrée désignée par `?Yl` (`?Yl.sem`). Par exemple, si

le nœud `?Y1` est étiqueté par le sémantème ‘fumeur’, alors la variable pointe vers l’entrée `fumeur` dans le *semanticon*. Dans cette entrée, la variable récupère la valeur de l’attribut « lex » (`.(lex)`). Dans le cas du sémantème ‘fumeur’, l’attribut « lex » contient la valeur FUMEUR. De ce fait, la valeur de `?L` est instanciée par le lexème FUMEUR. En ce qui concerne la variable `?F`, le chemin fouille le *lexicon* à la recherche de l’entrée correspondant à `?L`, c’est-à-dire l’entrée FUMEUR, et récupère son inventaire de FL (`.(lf)`). À ce stade, `?F` s’instancie par l’inventaire complet des fonctions lexicales de FUMEUR. Cependant, il faut encore lexicaliser la bonne fonction lexicale.

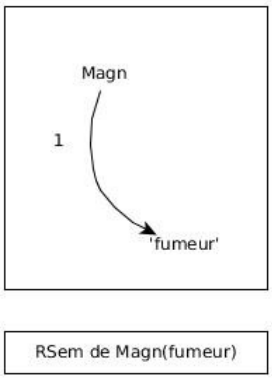



Figure 20. Représentation sémantique (RSem) de Magn(fumeur)

Pour s’assurer de lexicaliser la bonne FL, ici Magn, nous incluons la condition d’application suivante : `?F.name = X.sem`. Cette condition vérifie que le nom de la FL instanciée par la variable `?F` est identique à la valeur de l’attribut « sem » du nœud `?X1`. En d’autres termes, on force la variable `?F` à posséder la valeur Magn pour son attribut « name ». Dans le cas où FUMEUR possède plusieurs Magn dans son inventaire de FL, cette condition permet de générer tous les Magn possibles. Parmi les autres conditions, la règle s’assure que le nœud `?Y1` ne soit pas une FL et attend que ce nœud soit lexicalisé pour s’appliquer (`dsynt=ok`). Jusqu’ici, la règle a récupéré les informations suivantes :

- `?X1` s’instancie par le nœud étiqueté ‘Magn’ en RSem
- `?Y1` s’instancie par le nœud étiqueté ‘fumeur’ en RSem
- `?L` s’instancie par FUMEUR

- `?F` s'instancie par la FL Magn de FUMEUR

Maintenant que la règle a déterminé toutes les informations pertinentes à partir du graphe en entrée et des dictionnaires, la partie droite détermine le graphe de sortie, celui représentant la RSyntP, et son contenu.

La partie droite de la règle spécifie qu'il doit exister au préalable un nœud `?Yr`, équivalent au nœud `?Yl` de la partie gauche, ce qui est marqué par l'instruction `rc :?Yr ⇔?Yl` . Autrement dit, ce nœud est l'équivalent syntaxique profond de 'fumeur'. Il s'agit donc du nœud étiqueté par le lexème FUMEUR. Cette règle présuppose que la lexicalisation de 'fumeur' par FUMEUR a déjà été opérée par une autre règle.

Le nœud `?Yr` possède un actant en RSyntP, `?Xr`. `?Xr` correspond à la lexicalisation de l'étiquette du nœud instancié par `?Xl`. Sa lexicalisation est inscrite dans le trait `dlex` (litt. lexicalisation profonde) à l'aide de la valeur `?F.value`. Dans le cas présent, il s'agit du collocatif GROS.

La relation entre les nœuds `?Yr` et `?Xr` est spécifiée par le chemin suivant : `lf ::(Mod).(gp).(L).` Comme nous l'avons vu dans l'extrait du dictionnaire *lf* à l'exemple (72), Magn est une FL appartenant au patron des FL de Modification, ce qui se traduit dans le dictionnaire *lf* par la classe « Mod ». Cette classe regroupe toutes les FL regroupées sous le patron Modification. Ici, le chemin récupère la valeur de l'attribut « L » dans le patron de régime « gp » de la classe de FL « Mod » qui se trouve dans le dictionnaire *lf*. De ce fait, la relation entre `?Yr`, FUMEUR, et son Magn, GROS, s'instancie par la valeur ATTR.

Les appels aux dictionnaires nous permettent de lexicaliser les FL ainsi que leur relation vis-à-vis de la base à l'aide de la description des chemins d'accès aux informations pertinentes. Ce procédé s'applique à toutes les FL traitées dans GÉCO et met en avant l'interdépendance de nos ressources. L'Annexe contient des captures d'écran de chaque règle de transduction implémentée dans GÉCO.

Nous allons maintenant présenter chaque patron de FL.

## 3.3 Inventaire des patrons et leur combinaison

### 3.3.1 Modification

Plusieurs FL simples permettent d'intégrer en RSyntP un collocatif de type adjectival ou adverbial, ce dernier étant soumis à une relation attributive avec la base. Ces FL renvoient à un sens donné, tel que l'intensification (Magn), la louange (Pos), etc. Voici quelques exemples de ces fonctions lexicales.

(75) Magn(peur) = ~ bleue

(76) Bon(conseil) = ~ précieux

(77) Ver(peur) = ~ justifiée

(78) Pos(opinion) = ~ favorable

Dans chaque cas, la FL est sémantiquement pleine, c'est-à-dire qu'elle doit être représentée dans la structure sémantique de la phrase à l'aide de son nom, voir Figure 21.

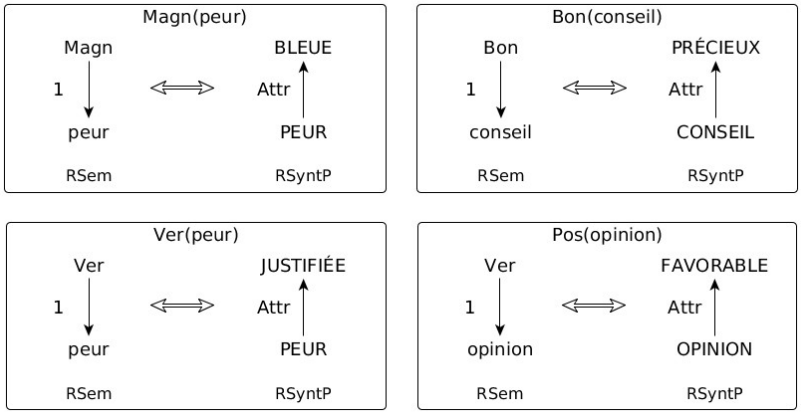



Figure 21. Fonctionnement de Bon, Magn, Ver et Pos

Par ailleurs, ces fonctions lexicales sont prédicatives, c'est-à-dire qu'elles possèdent toutes un actant en sémantique, la base. Lorsque l'on passe en RSyntP, la relation s'inverse entre la base et le collocatif, comme la Figure 21 le montre.

La position actancielle 1 de la FL sera toujours remplie par la base RSem, peu importe sa valeur. Cette relation s'instancie par la relation attributive entre la base et le collocatif dans la

représentation syntaxique profonde (RSyntP) de la phrase. La Figure 22 illustre ce propos et offre une première généralisation sur le fonctionnement de ces FL.

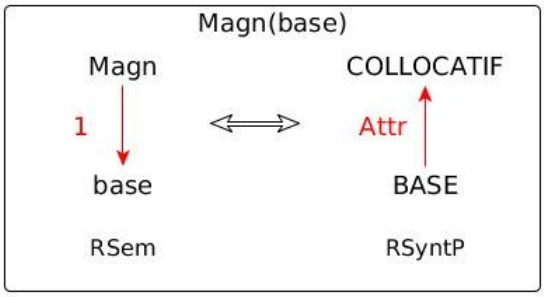



Figure 22. Première généralisation de Magn

Même si les fonctions lexicales renvoient à une relation sémantico-lexicale spécifique, elles introduisent toutes une relation attributive entre la base et le collocatif en RSyntP. Le collocatif agit donc comme un modificateur adjectival ou adverbial de la base. Ces FL sont donc regroupées au sein d'un unique patron, nommé Modification. Chacune se réalisera par une relation de dominance inversée attributive entre la base et le collocatif dans la RSyntP de la phrase.

Le patron Modification modélise une lexicalisation complexe puisqu'il lexicalise à la fois la base, son collocatif ainsi que la relation attributive inversée entre les deux. Le fonctionnement du patron Modification est synthétisé dans la Figure 23. Ce patron s'implémente dans MATE à l'aide d'une unique règle de transduction nommée « lex_modification ». Une capture d'écran est fournie dans la Figure 19.

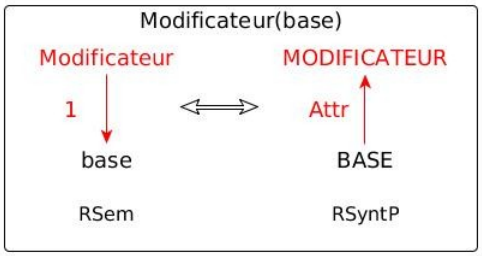



Figure 23. Fonctionnement du patron Modification

Les FL du patron Modification se combinent uniquement pour former des FL complexes des exemples (79) et (80). La modélisation de ces combinaisons dans l'interface RSem ⇔ RSyntP

est présentée dans la Figure 24. Ces combinaisons ont été introduites à l'aide du même patron. En effet, nous avons intégré l'antonymie directement au nœud de la FL de Modification. La section 3.3.7.2 développe notre traitement de la FL 'Anti'.

(79) AntiBon(choix) = mauvais

(80) AntiVer(accusation) = ~ infondée

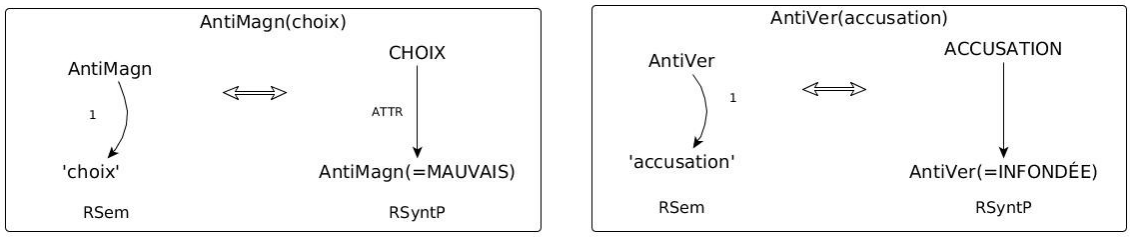



Figure 24. Fonctionnement des FL complexes de Modification

## 3.3.2 Préposition

Certaines FL introduisent une préposition dans la RSyntP de la phrase.

(81) $Loc_{in}$(terrasse) = en ~

(82) $Loc_{ad}$(filet)= dans ART ~

(83) $Loc_{ab}$(balcon)= depuis ART ~

(84) Instr(corde)= avec ART ~

La famille des FL Loc a été décrite plusieurs fois en Sens-Texte, notamment par (I. Mel′čuk, 1996). Elles servent à introduire une « préposition ou expression prépositive exprimant le sens 'se trouvant dans' [spatialement ou temporellement] (=$Loc_{in}$), 'se déplaçant à partir de (=$Loc_{ab}$), 'se déplaçant pour se trouver dans' (=$Loc_{ad}$) ». Instr, quant à elle, insère une préposition régissant le mot-clé et exprime le sens « au moyen de », selon les mêmes auteurs. Nous avons également incorporé la FL Propt dans ce patron. En effet, elle permet également d'introduire une préposition régissant la base. Son sens est 'à cause de' (I. Mel′čuk et al., 1999).

L'usage de ces FL est à différencier de l'insertion d'une préposition par les verbes transitifs indirects comme dans l'énoncé *Pierre parle à Jean*. Ici, la préposition est vide de sens et ne sert qu'à introduire un argument du verbe. Il s'agit donc d'une information relative au régime

du verbe. Dans le cas des Loc, Propt et Instr, la préposition insérée exprime un sens à part entière. La Figure 25 présente leur mode de fonctionnement.

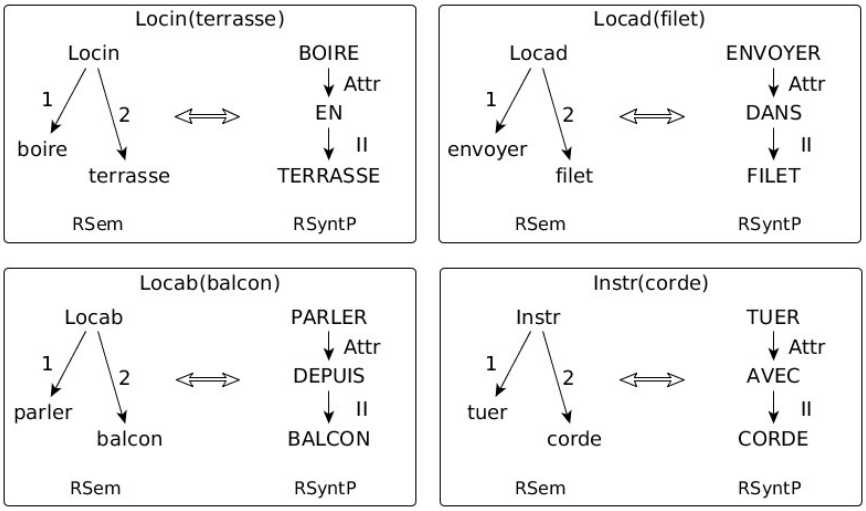



Figure 25. Fonctionnement de $Loc_{in}$, $Loc_{ab}$, $Loc_{ad}$ et Instr

La Figure 25 met en avant la similarité de fonctionnement de ces FL. Dans chaque cas, la FL est prédicative et possède deux actants en RSem. L'actant 1 concerne une action, ou une personne. L'actant 2 représente la base de la collocation et est donc soit l'entité dans laquelle se passe l'action ($Loc_{in}$), l'entité de départ de l'action ($Loc_{ab}$), l'entité cible de l'action ($Loc_{ad}$), l'entité utile à l'action (Instr) ou la source de l'action (Propt).

Une fois la lexicalisation effectuée, on observe que la relation est inversée entre le collocatif et l'actant 1 de la FL en RSyntP. Autrement dit, la lexicalisation de l'actant 1 gouverne à présent la FL grâce à une relation attributive (ATTR) en RSyntP. Ensuite, la relation entre le collocatif et la base, l'actant 2 en RSem, s'instancie en RSyntP par la relation II. De même, la valeur de la FL a été intégrée à la représentation de la phrase. Il est donc possible de regrouper ces FL sous le patron intitulé Préposition dont le fonctionnement est décrit dans la Figure 26. Il est important de noter que la base d'une FL Préposition doit nécessairement se lexicaliser par un lexème nominal. Il faut donc lexicaliser la base en même temps que la FL.

Ce patron s'implémente dans le générateur à l'aide d'une unique règle de transduction dans MATE nommée « lex_preposition ». Elle lexicalise la FL en même temps que la base et la

relation de dépendance II entre les deux. De même, la relation attributive entre la Préposition et l'élément X est lexicalisée. Il s'agit donc d'une lexicalisation complexe.

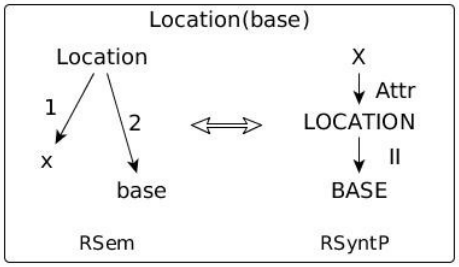



Figure 26. Fonctionnement du patron Préposition

### 3.3.3 Nom Gouverneur Sémantique

Plusieurs FL sémantiquement pleines insèrent un collocatif nominal régissant la base en syntaxe profonde.

(85) Culm(enthousiasme) = déchainement de ART ~

(86) Sing(chocolat) = carré de ~

(87) Mult(loup)= meute de ~

(88) Germ(colère) = levain de ART ~

(89) Centr(forêt) = cœur de ART ~

(90) Cap(famille) = chef de ~

(91) Equip(bateau) = équipage de ~

Comme pour les FL décrites précédemment, celles-ci ont été mentionnées à plusieurs reprises dans la littérature et nous allons nous baser essentiellement sur Mel'cuk et al. (1995) pour les définir. Culm renvoie au sens de 'culmination de ... '. De même, Sing renvoie à l'unité minimale régulière de... ', Mult exprime un 'ensemble régulier de…', Germ signifie 'gerne/origine de…', Centr dénote 'le centre de …' et enfin Cap 'reflète l'idée du 'chef de …' alors que Equip signifie 'équipe …'. Chacune est exprimée en RsyntP par un nom gouvernant la base, comme le montre la Figure 27.

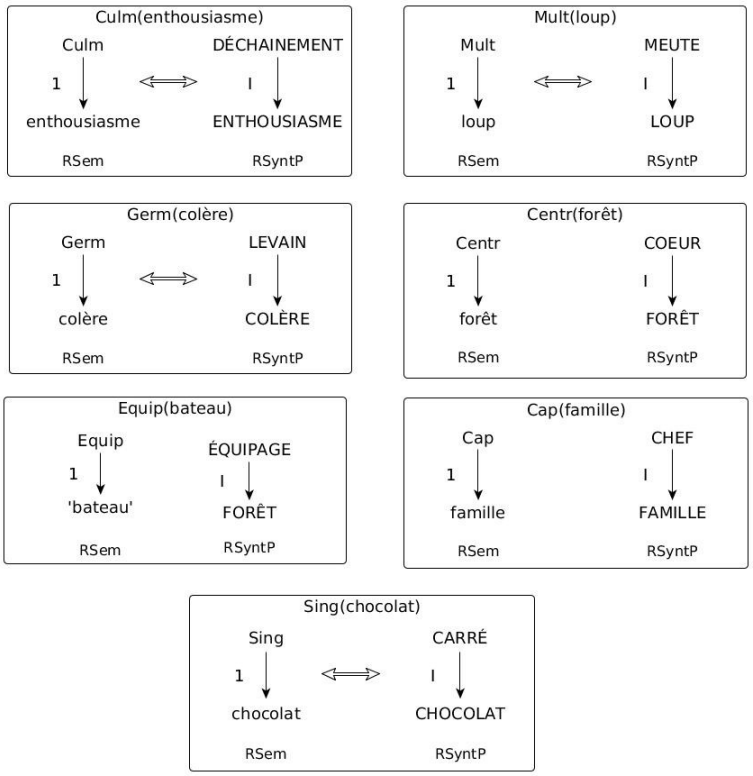



Figure 27. Fonctionnement de Culm, Mult, Centr, Cap, Equip, Germ et Sing

Il est clair que sémantiquement, ces FL divergent. Pourtant leur représentation en RSem et leur correspondance en RSyntP sont identiques. Dans chaque cas, la RSem intègre un nœud prédicatif ayant la base comme actant 1. Ces FL introduisent ensuite le collocatif nominal en RSyntP, ce dernier régissant la lexicalisation de la base avec la relation I (complément du nom en syntaxe de surface). Ces FL ont été regroupées sous le patron Nom Gouverneur Sémantique (NgovSem) dont le fonctionnement est représenté par la Figure 28, encodé par la règle de lexicalisation « lex_ngovsem » dans MATE. Le patron lexicalise la FL, la relation I entretenue avec la base et la base. Il s'agit donc d'une lexicalisation complexe.

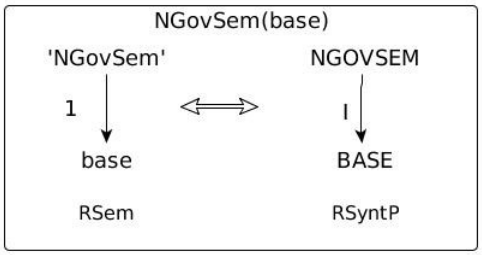



Figure 28. Fonctionnement du patron NGovSem

Les Noms Gouverneurs Sémantiques se combinent avec les Prépositions pour former des FL complexes du type :

(92) $Loc_{in}Centr$(forêt) = au cœur de ART ~

Nous avons donc créé un patron « lex_loc_ngov_sem » pour gérer ces cas particuliers, voir Annexe. Ce patron lexicalise $Loc_{in}Centr$ à l'aide de la préposition complexe : AU_COEUR_DE. Autrement dit, cette préposition est considérée comme une locution, et non comme une combinaison libre de la préposition À suivie du nom CŒUR.

### 3.3.4 Verbes supports

#### 3.3.4.1 Fonctionnement général

L'analyse et le traitement de verbes supports est un phénomène qui a largement été traité, en particulier dans la Théorie Sens-Texte, par (Alonso Ramos, 2007; I. Mel′čuk, 2004) pour n'en citer que quelques-uns. Un verbe support est sémantiquement vide dans son usage en tant que collocatif. Autrement dit, les verbes supports ne sont pas présents en RSem mais sont des coquilles syntaxiques verbales présentent en RSyntP dont le but est d'assurer la grammaticalité d'un énoncé. L'utilisation de verbes supports est très fréquente en français, comme dans plusieurs autres langues européennes, et a un impact important sur la structure syntaxique d'un énoncé. Les FL dénotant un verbe support ont été regroupées sous le patron général Verbes Supports (vsupp).

En Théorie Sens-Texte, les verbes support sont répartis en trois grands types : Func, Oper et Labor.

(93) $Func_0$(pluie)= ART ~ tomber

(94) $Oper_1$(accusation)= émettre ART ~

(95) $Labor_{21}$(cadeau)= offrir PREP ~

Ils sont définis en fonction du rôle syntaxique de la base vis-à-vis du verbe support en RSyntP. Lors de l'emploi de Func, la base devient l'argument syntaxique profond I (sujet syntaxique) du verbe support. La base devient l'argument II (objet direct) lorsque la fonction lexicale est Oper, puis incarne l'argument III (objet indirect) lors de l'emploi de Labor, voir Figure 29.

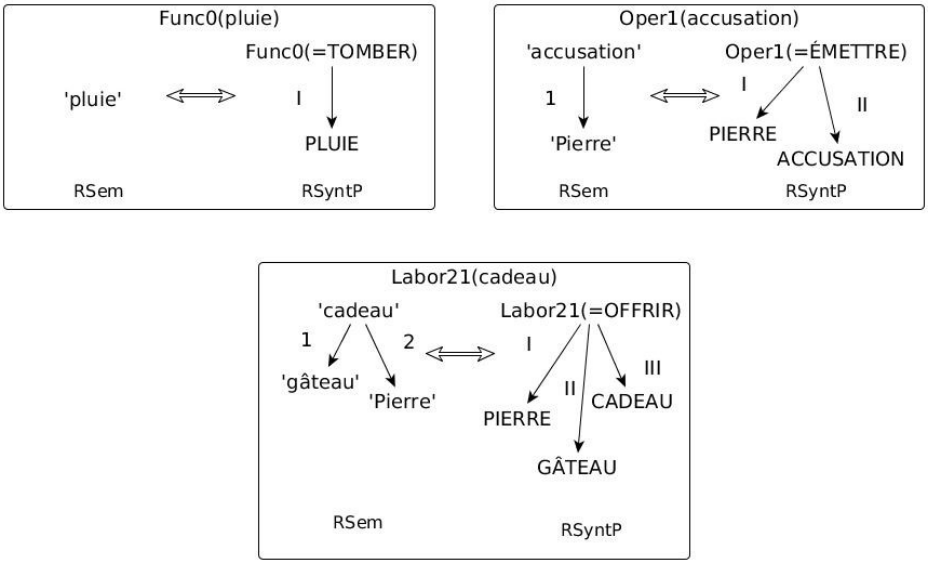



Figure 29. Fonctionnement de Func, Oper et Labor

Ces FL présentent un indice qui fournit la position syntaxique des actants de la base par rapport au verbe support. Traditionnellement, cet indice fait référence aux actants syntaxiques profonds du prédicat. Dans notre implémentation, les indices font référence aux actants sémantiques de la base. La Figure 29 présente le fonctionnement de $Oper_1$ sur le mot-clé ACCUSATION.

La structure en RSem contient deux sémantèmes, ‘accusation’ et son actant 1, ‘Pierre’. La correspondance en RSyntP de cette structure contient trois nœuds: le verbe support, ÉMETTRE, son actant II, la lexicalisation de la base, ACCUSATION, et son actant I, la lexicalisation de l’actant 1 de la base, PIERRE. Ici, l’indice 1 fait référence à la lexicalisation de l’actant sémantique 1 de la base. Dans la Figure 30, $Oper_2$ est également appliqué sur le mot-clé ACCUSATION.

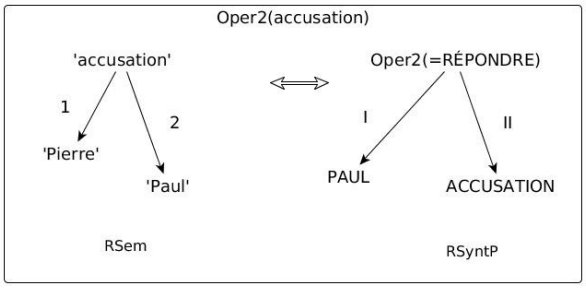



Figure 30. Fonctionnement $Oper_2$(accusation)

Dans cet exemple, le sémantème ‘accusation’ possède deux arguments en RSem. La différence avec $Oper_1$ est que l’actant I en RSyntP est lexicalisé par l’actant 2 de la base, ici ‘Paul’. L’indice ‘2’ de $Oper_2$ fait donc référence à la lexicalisation de l’actant sémantique 2 de la base, ici ‘Paul’. Le collocatif fourni par la FL est donc différent de celui fourni par $Oper_1$. Ce mécanisme d’indice permet de contraindre la lexicalisation des actants des mots-clés et générer des textes de meilleure qualité. Le fait de récupérer l’information en RSem permet de généraliser l’application d’indices, comme la Figure 31 le montre. En RSem, la base ‘x’ possède un i-ème actant ‘a’. $Oper_i$ lexicalise cet actant en RSyntP. Ce traitement des indices se généralise pour toutes les FL introduisant un collocatif verbal. Dans les faits, la lexicalisation des actants se fait grâce à une variable et un appel au dictionnaire. Le Tableau VI illustre ce processus.

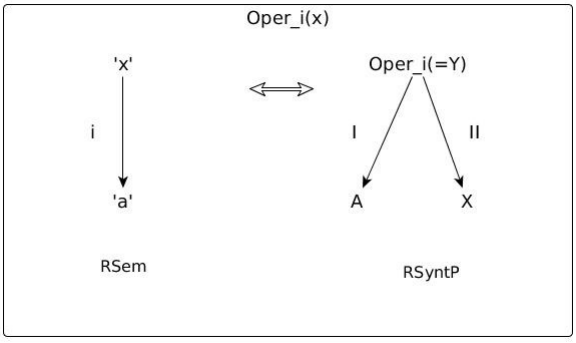



Figure 31. Fonctionnement général des indices pour les FL verbales

Voici le fonctionnement de cette règle :

- La partie gauche contient un nœud `?Xl` et un nœud `?Yl`, régi par la relation `?i` avec `?Xl`.
- Les conditions imposent à la FL de porter le nom « Oper » auquel on concatène la valeur de la variable `?i`.
- La partie droite lexicalise la base à l’aide du verbe support, étiqueté sur le nœud `?Zr`, et de sa correspondance, étiquetée sur le nœud `?Xr`. Comme le collocatif ne porte pas de sens et est une extension de la base, les traits « split=top » et « split=bottom » marquent que le nœud `?Xl` en RSem se lexicalise par deux

nœuds, `?Zr` et `?Xr` en RSyntP. La valeur de la FL est ensuite récupérée à l'aide de l'attribut « dlex ».

Enfin, comme nous l'avons dit plus haut, dans notre approche, les indices des FL renvoient aux actants sémantiques de la base. Cependant, la Théorie Sens-Texte indique que les indices renvoient normalement aux actants syntaxiques profonds de la base.

| **Contexte gauche** | **Contexte droit** |
|---|---|
| ?Xl {<br>?i-> ?Yl {}<br>}<br>?L <- semanticon ::(X.sem).(lex)<br>?F<- lexicon ::(?L).(lf) | ?Zr {<br>⇔ ?Xr<br>split=top<br>dlex=?F.value<br>lf ::(?.name).(gp).(L) -> ?Xr<br>{<br>⇔ ?Xl<br>split=bottom<br>}<br>lf ::(?.name).(gp).(?i) -> ?Yr<br>{⇔ ?Yl}<br>} |
| **Conditions** | |
| ?F.name= "Oper"+?i; | |

Tableau VI verbes supports et indices dans GÉCO

#### 3.3.4.2 Intégration dans MATE

Il existe trois règles de transduction correspondant au patron Verbes Supports dans MATE. La différence entre ces trois patrons porte sur la description du nombre d'actants syntaxiques

profonds du verbe support. En effet, MATE nous contraint à expliciter de manière statique le nombre de nœuds et d'arcs créés dans la structure RSyntP de sortie.

De ce fait, la règle « lex_vsupp_0 » lexicalise la base avec son verbe support. Elle modélise donc le fonctionnement de $Func_0$ et $Oper_0$. La deuxième règle, « lex_vsupp_i » lexicalise la base, le verbe support ainsi qu'un actant de la base. Elle prend donc en charge $Oper_1$, $Oper_2$, $Func_1$, $Func_2$, $Labor_1$, $Labor_2$, etc. Enfin, le troisième patron, « lex_vsupp_ij » lexicalise la base, le verbe support et deux actants de la base. Les FL $Oper_{12}$, $Oper_{21}$, $Func_{12}$, etc. sont donc traitées par ce patron. Le Tableau  présente la procédure d'attribution des étiquettes des relations en RSyntP. Pour ce faire, deux appels au dictionnaire *lf* sont effectués :

- `lf ::(?F.name).(gp).(L) -> ?Xr` fouille le dictionnaire *lf* pour récupérer l'entrée de la FL correspondant à la valeur de `?F.name`. Ce chemin fouille son patron de régime (`gp`) et récupère la valeur assignée à l'attribut « L », qui désigne le mot-clé. L'exemple (96) présente l'entrée de dictionnaire pour la FL $Oper_1$, sous-type de Oper. L'attribut « L » possède la valeur « II ». Cette valeur est inscrite en RSyntP.
- `lf ::(?F.name).(gp).(?i) -> Yr` est un chemin similaire mais récupère la valeur de l'attribut « `?i` » et non « L ». Dans le cas de $Oper_1$(accusation), « `?i` » est instancié par la valeur « 1 ». Ce chemin récupère donc la valeur « I » et l'inscrit en RSyntP.

(96) $Oper_1$ dans dictionnaire *lf* :

```
Oper : verb {
  gp = { L = II }

}
Oper1 :Oper {
  gp = { 1 = I }
}
```

La valeur de l'attribut « 1 » étant « I », la règle « lex_vsupp_i » inscrit la relation « I » entre le verbe support et la lexicalisation de l'actant 1 de la base lorsque la FL est $Oper_1$.

Nous verrons dans la section 3.3.7 les différentes FL complexes qu'il est possible de former avec toutes les FL verbales, c'est-à-dire les FL introduisant un collocatif verbal en RSyntP. Les verbes supports sont un sous-ensemble des FL verbales. Nous présentons maintenant les deux autres types de FL verbales : les verbes de réalisation et les autres verbes sémantiquement pleins.

### 3.3.5 Verbe de réalisation

Il est possible d'insérer des collocatifs verbaux autres que des verbes supports. Les verbes ayant un sens de réalisation, c'est-à-dire faire de quelque chose ce qu'on est censé en faire, sont en sont un exemple. Ces verbes sont dénotés par les FL Fact, Real et Labreal. À l'instar des verbes supports, les étiquettes des verbes de réalisation indiquent le positionnement de la lexicalisation de la base vis-à-vis du verbe introduit.

(97) $Fact_1$(couteau)= ART ~ couper

(98) $Real_1$(voiture)= conduire ART ~

(99) $Labrea_{12}$(armoire)= ranger N dans ART ~

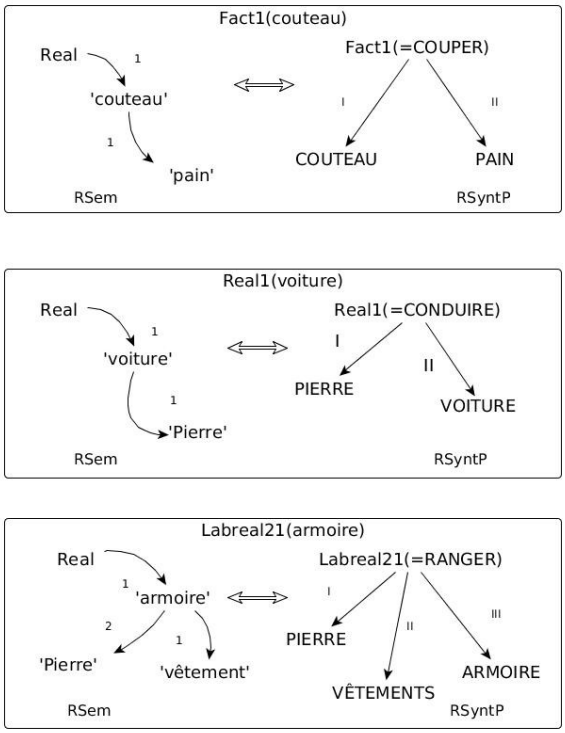



Figure 32. Représentation des verbes de réalisation

Contrairement aux verbes supports, les verbes de réalisation sont sémantiquement pleins. Fact, Real et Labreal possèdent le sen de réalisation de la base. De ce fait, ils sont représentés en RSem de la même manière, comme le montre la Figure 32. Comme on peut le voir, les FL de réalisation sont intégrées en RSem à l'aide d'un prédicat unaire dont l'unique argument est la base. Il s'agit d'une simplification sémantique. Prenons l'exemple (97). Si l'on fait la décomposition sémantique de 'couteau', on obtient la Figure 33 :

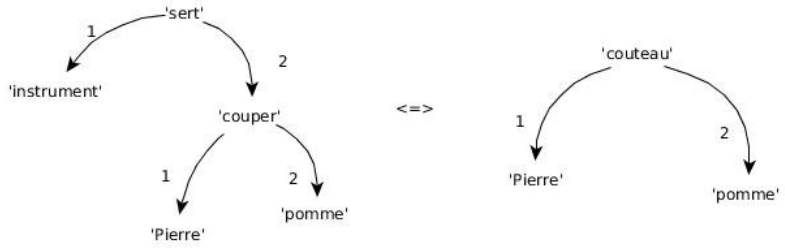



Figure 33. Décomposition sémantique de 'couteau'

Cette structure permet de mettre en évidence que le fait de réaliser sémantiquement un prédicat comme 'couteau' introduit une redondance dans la RSem :

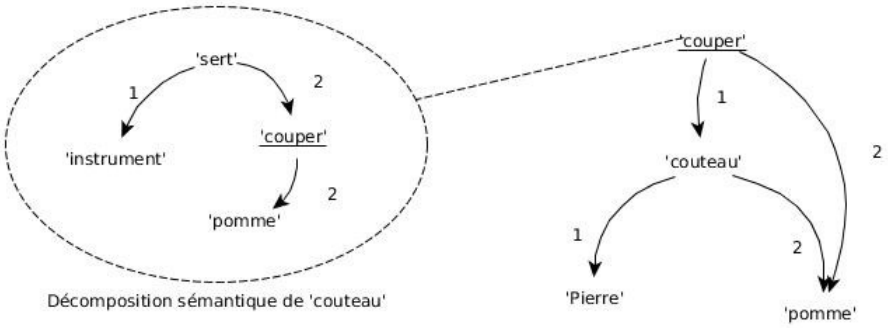



Figure 34. Réalisation du sens 'couteau'

En effet, 'couteau' possède déjà le sens de 'couper' dans sa décomposition sémantique. La réalisation de ce sémantème met au jour cette composante sémantique de 'couteau'. Cependant, il est difficile de retrouver la composante sémantique qui se réalise lorsque l'on applique une FL verbale de réalisation. Pour ce faire, il faudrait accéder à un niveau encore plus abstrait, une représentation conceptuelle, où l'information du concept 'couteau' contiendrait un attribut 'fonction'. La valeur de cet attribut représenterait la composante sur

laquelle porte le verbe de réalisation. Dans le cas de ‘couteau’, l’attribut ‘fonction’ aurait pour valeur ‘couper’. Il s’agit ici de connaissances conceptuelles, et non linguistiques, sur les objets manipulés par le générateur.

D’un point de vue syntaxique, les verbes de réalisation fonctionnent de la même manière que les verbes supports. Il existe trois règles de transductions : « lex_vreal_0 », « lex_vreal_i » et « lex_vreal_ij ». Ces règles permettent de lexicaliser la base, le verbe de réalisation et un ou deux actants de la base en fonction de l’indice porté par la FL de réalisation.

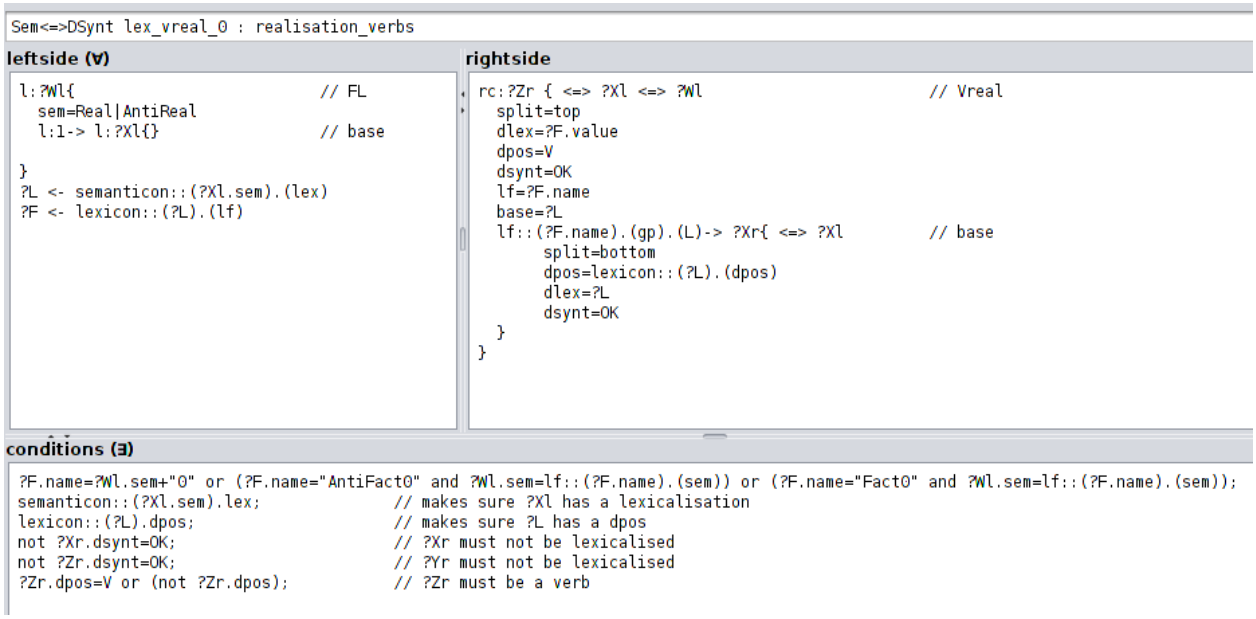



Figure 35. Règle « lex_vreal_0 »

La Figure 35 est une capture d’écran de la règle « lex_vreal_0 ». Elle met en évidence la dernière distinction existante entre les verbes supports et les verbes de réalisation. La partie gauche contient un nœud `?Wl` dont l’étiquette sémantique « sem » est nécessairement ‘Real’, ou ‘AntiReal’. Lors de l’application de la règle, la variable `?F.name` permet de récupérer le nom de la FL. Dans le cas présent, la règle spécifie que la valeur de la variable `?F.name` est soit $Real_0$ (valeur du trait « sem » du nœud `?Wl`), $AntiReal_0$, $Fact_0$ ou $AntiFact_0$. Cette information est encodée à l’aide de la première condition d’application. Elle spécifie que `?F.name` doit être égale à une des propositions suivantes :

- à la valeur du trait « sem » du nœud `?Wl`

- à $AntiFact_0$ si la valeur du trait « sem » de $AntiFact_0$ dans le dictionnaire *lf* est égale à celle du nœud `?W1` (AntiReal)
- à $Fact_0$ si la valeur du trait « sem » de $AntiFact_0$ dans le dictionnaire *lf* est égale à celle du nœud `?W1` (Real)

Cette indication est nécessaire puisque Real et Fact possède le même sens, la réalisation, que nous avons décidé d'indiquer en RSem par le nom 'Real'. Cette mention permet donc de générer un $Fact_0$ le cas échéant. De même, si la RSem contient le nom 'AntiReal' comme étiquette de `?W1`, la règle pourra générer un $AntiReal_0$ ou un $AntiFact_0$. Nous sommes conscients qu'il n'est pas judicieux de représenter cette information en dur dans la règle. Pourtant, il nous semble plus important de respecter le fait que Fact et Real sont deux équivalents de la même FL en RSem.

### 3.3.6 Autres verbes sémantiquement pleins

En plus des verbes de réalisation, d'autres FL introduisent des collocatifs verbaux non vides de sens.

(100) Manif(gaieté) = rayonner de ~ ['se manifester']

(101) Obstr(vue) = ~ se brouiller ['fonctionner difficilement']

(102) Excess(sang) = ~ bouillir ['fonctionner d'une façon excessive']

(103) Son(cloche) = ~sonner ['produire le son typique']

(104) Sympt1(effroi) = pâlir de ~ ['montrer un symptôme physique d'un état psychologique, ce symptôme étant un état d'une partie du corps ou d'un organe']

(105) Degrad(lait) = ~ tourner ['se dégrader, devenir pire']

(106) Stop(coeur) = ~ flancher ['arrêter de fonctionner']

Ces FL sont décrites dans (I. Mel′čuk, 1996). Syntaxiquement parlant, elles fonctionnent comme un Fact ou un Func, c'est-à-dire que la base est l'argument I du collocatif verbal en RSyntP. La Figure 36 est représentative de leur fonctionnement. Chacune de ces FL possède un sens distinct, repris entre crochets dans les exemples (100) à (106).

En général ces FL ne possèdent pas d'indices ou ne forment pas de triplets pour représenter leur comportement syntaxique, comme Real/Fact/Labreal ou Func/Oper/Labor. Elles sont

moins décrites que les verbes supports et les verbes de réalisation. Cependant, il pourrait être utile d'intégrer ces informations dans le réalisateur car certains cas peuvent être problématiques. Par exemple :

(107) Manif(gaieté)= ~ se lire ; ex. la gaieté se lit sur le visage de Paul

(108) Manif(gaieté) = rayonner de ~ ; ex. le visage de Paul rayonne de gaieté / la gaieté rayonne sur le visage de Paul

Dans l'exemple (108), la base semble se lexicaliser en tant qu'argument II du collocatif verbal. La FL agit donc plutôt comme un Oper. Pourtant, dans l'exemple (107), la base se lexicalise soit comme actant I ou II par rapport au collocatif. Malheureusement, ces FL n'ont pas été beaucoup décrites dans la littérature. Nous avons choisi d'intégrer des indices lorsqu'un ou plusieurs actants se réalisent en même que la base. Malgré tout, la base garde sa position actancielle I en syntaxe profonde. Toutes ces FL ont été regroupées sous le patron 'Autres Verbes Sémantiquement Pleins'. Malgré leurs sens différents, ces FL fonctionnent de manière similaire. La Figure 36 présente le fonctionnement général de ce patron.

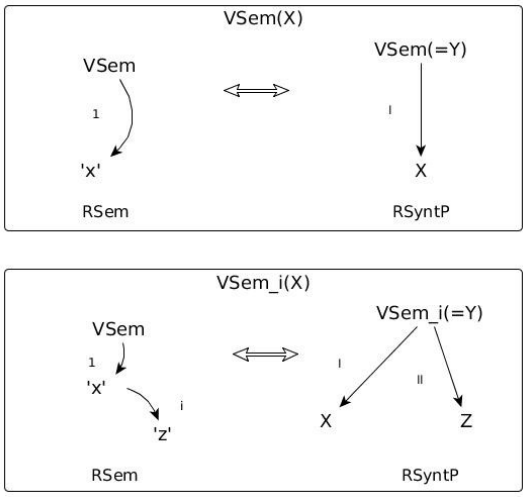



Figure 36. Fonctionnement des autres verbes sémantiquesment pleins

La sémantique de chacune de ces FL est spécifiée dans leur entrée dans le dictionnaire *lf*. Cette spécification permet de simplifier grandement les règles de transduction de ces verbes sémantiquement pleins. Encore une fois, il est nécessaire de préciser trois patrons de constructions de ces FL simples en fonction du nombre d'actants que l'on souhaite lexicaliser.

### 3.3.7 Combinaisons des FL verbales

Maintenant que nous connaissons le fonctionnement des verbes supports, de réalisation et sémantiques, nous pouvons aborder la question des FL complexes créées à partir de ces FL verbales. Comme nous allons le voir, ces trois types de FL se combinent de la même manière pour former des FL complexes.

#### 3.3.7.1 Phases, Prepar, Prox, Non et FL verbales

(I. Mel′čuk, 1996; I. Mel′čuk et al., 1999) précisent qu'il est possible de combiner les verbes supports avec des FL exprimant « trois phases différentes d'un état ou événement ». Il existe trois phases, voir (109) à (111). L'exemple (112) illustre un cas de FL complexe avec une FL phasique et un verbe support.

(109) Incep(X) = 'commencer à X'

(110) Fin(P) = 'cesser de X'

(111) Cont(P) = 'ne pas cesser de X'

(112) $IncepOper_2$(accusation) = tomber en ~ [litt. Commencer à être accusé]

Comme nous suivons l'approche de (Kahane & Polguère, 2001) et encodons les FL complexes à l'aide de compositions sémantiques, il suffit d'intégrer un nœud représentant la FL phasique dans la RSem de la phrase. Dans le cas de (112), cela donne la Figure 37.

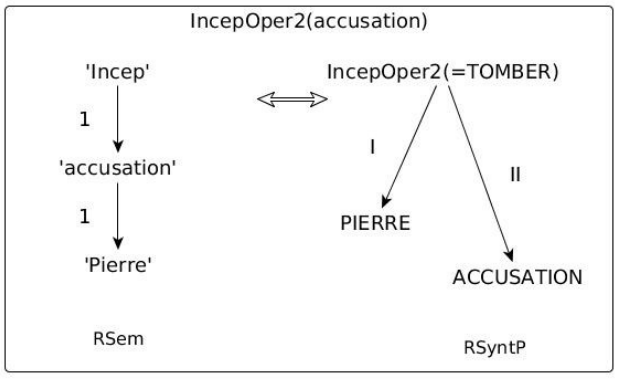



Figure 37. Fonctionnement de $IncepOper_2$(accusation)

Comme les verbes supports sont sémantiquement vides, le nœud phasique régit directement la base en RSem. Par contre, lorsque la FL verbale est sémantiquement pleine, le nœud phasique régit directement la FL verbale, voir Figure 38.

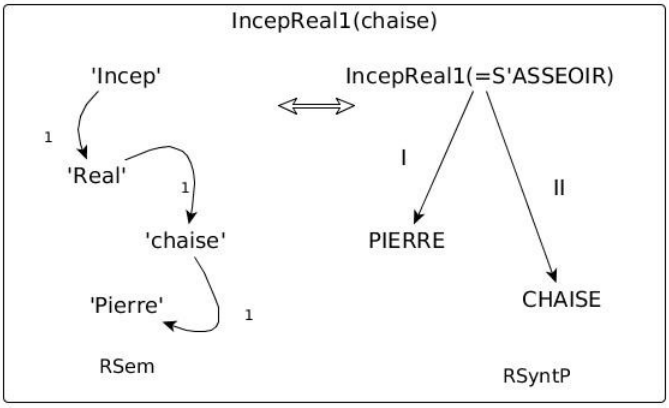



Figure 38. Fonctionnement de $IncepReal_1$(chaise)

(I. Mel′čuk et al., 1995) indiquent que « les trois FL phasiques n'ont pas de structure actancielle propre et ont donc besoin de s'appuyer sur les FL actancielles, telles que Oper, Func et Labor, ou Real, Fact et Labreal ». En nous appuyant sur ce passage, nous avons décidé de ne pas inclure les FL phasiques seules. De même, comme elles qualifient un « état ou événement », le seul actant qu'elles possèdent dénote la FL sur laquelle elles s'appliquent.

Ces FL se réalisent en RSyntP par un nœud verbal qui correspond à la fois au nœud phasique et au nœud de la FL verbale (quand il s'agit d'une FL sémantiquement pleine). Encore une fois, il est nécessaire de créer trois règles de transduction séparées en fonction du nombre d'actants que l'on souhaite lexicaliser.

Nous avons intégré d'autres FL en plus des FL phasiques dans ce type de combinaisons : Prox ('être sur le point de') et Prepar ('préparer pour'). Ce choix a été fait en accord avec les exemples de ces FL fournis dans (I. Mel′čuk, 1996; I. Mel′čuk et al., 1999, 1995). Alors que ces FL ne possèdent pas de sens phasique à proprement parler, les exemples fournis ne montrent Prepar et Prox qu'en combinaison avec d'autres FL verbales. De même, sémantiquement, ces FL représentent l'anticipation (volontaire ou non) d'une action ou d'un état. Leur sens est donc proche de celui d'une FL phasique. En raison de leur fonctionnement

syntaxique similaire, nous avons estimé qu'il était judicieux de les intégrer sous les mêmes règles de transduction plutôt que d'en créer de nouvelles.

Enfin, nous avons également inclus la négation à l'aide du label 'Non'. Non n'est pas une FL prévue par la Théorie Sens-Texte à la base. Cependant, (I. Mel′čuk, 1996) ont recours à cette notation pour définir les FL phasiques :

(113) Incep(P)

(114) Fin(P)=Incep(NonP)

(115) Cont(P)= NonFin(P)

Nous pensons que la négation est un élément intéressant à intégrer dans notre implémentation car elle permet de décrire des relations du type (116) et (117). Nous avons donc inséré Non dans les patrons de FL phasiques.

(116) $Real_2$(chance) = prendre A-POSS ~

(117) $NonReal_2$(chance) = laisser passer A-POSS ~

3.3.7.2 Anti, Non et les FL verbales

Les verbes de réalisation et les FL de Modification peuvent également être combinés avec la FL paradigmatique Anti (antonymie).

(118) $Real_1$(bataille) = gagner la ~

(119) $AntiReal_1$(bataille) = perdre la ~

(120) Magn(angoisse) = grande ~

(121) AntiMagn(angoisse)= ~ légère

(I. Mel′čuk et al., 1995., p. 110) définissent la FL Anti comme suit :

> Antonyme [Anti] : Comme cette notion est également fort connue, nous nous limiterons à préciser que la lexie L1 est une Antonyme de la lexie L2 si et seulement si leurs signifiés sont identiques sauf pour la négation se trouvant « au sein » d'un des deux signifiés. Ainsi, CONSTRUIRE [une maison] signifie 'causer que [la maison] commence à exister'; son antonyme, DÉTRUIRE, a le sens 'causer que [la maison] commencer à NE PAS exister'.

Dans les exemples (119) et (121), la FL Anti porte sur un élément de sens introduit par la FL, que ce soit Real ou Magn. Autrement dit, il est possible de paraphraser le sens de chacune

respectivement par ‘ne pas réaliser les « objectifs » inhérents de [la bataille]’ (autrement dit [la bataille] a lieu mais son objectif inhérent ne s’est pas réalisé) et ‘[angoisse] qui n’est pas intense’. Dans le cas de AntiMagn, la définition de l’antonymie ne permet pas de rendre compte de la valeur retournée: il semble exister une relation d’antonymie en raison du caractère gradable du modificateur adjectival, c’est-à-dire l’inverse de ‘intense’ est ‘peu intense’ et non ‘qui n’est pas intense’ (Gagné, 2015; Gagné & L’Homme, 2016). Malgré tout, l’antonymie porte sur une composante sémantique, voire conceptuelle, de la FL. Nous avons décidé de représenter l’antonymie dans le même nœud que la base en RSem, comme le montre la Figure 39 présentant les RSem des exemples (119) et (121).

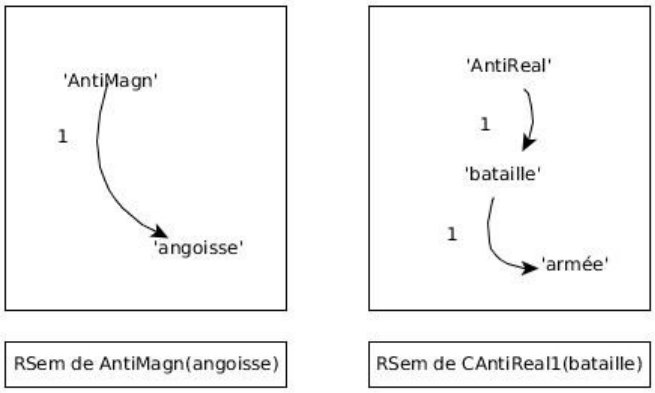



Figure 39. Fonctionnement de Anti en RSem

(I. Mel′čuk et al., 1995) ajoutent que « Anti se combine facilement avec d’autres FL (surtout Magn, Ver, Bon, Real) pour former des FL complexes ». Nous avons donc intégré Anti pour former des FL complexes avec les patrons Modification et Verbes de Réalisation.

#### 3.3.7.3 Causatifs et FL verbales

La TST définit un autre triplet formé des FL causatives. Elles expriment trois types de causation d’un état ou d’un événement. Elles sont annotées de la manière suivante :

(122) $CausFact_0$(fusée) = lancer ~

(123) $LiquFunc_0$(appréhension) = calmer ~

(124) $Liqu_1Func_0$(appréhension) = vaincre ~

(125) $Perm_1Manif$(aversion) = afficher [A-POSS] ~

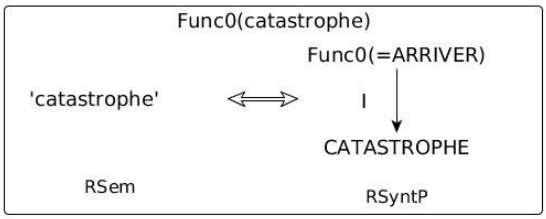
Encore une fois, il existe une relation logique entre ces trois FL. Caus signifie ‘causer que X’, Liqu veut dire ‘causer que non X’ (Caus(nonX)) alors que Perm porte le sens de ‘ne pas causer que non X’ (nonLiquX). (I. Mel′čuk, 1996) ne précisent pas si les FL causatives peuvent apparaître seules ou nécessairement en combinaison avec d’autres FL verbales. Cependant, tous les exemples fournis de ces FL les intègrent dans une FL complexe. Nous avons donc choisi de les implémenter uniquement en combinaison avec d’autres FL.

Contrairement aux FL phasiques, les FL causatives introduisent un nouvel élément dans la structure actancielle. Cet élément renvoie au causateur. Il est toujours marqué par la relation I avec le collocatif verbal. Cela a pour conséquence de décaler la réalisation de la structure actancielle du collocatif verbal. Prenons les exemples suivants :

(126) $Func_0$(catastrophe) = ~ arriver (ex. *une catastrophe arrive*)

(127) $CausFunc_0$(catastrophe) = provoquer une ~ (ex. *François provoque une catastrophe*)

Ici, CATASTROPHE est réalisée comme actant syntaxique profond I dans l’exemple (126) et comme actant syntaxique profond II dans l’exemple (127). Cela se traduit par le schéma de la Figure 40.



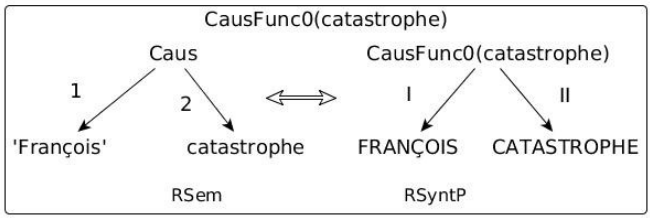




Figure 40. « Décalage » des actants en RSyntP à cause d'une FL causative

Nous distinguons deux types de causation : la causation involontaire et la causation volontaire. Dans le premier cas, l’élément causateur est un élément qui n’a rien à voir avec la situation dénotée par la base. Autrement dit, il ne s’agit pas d’un actant du mot-clé. Par opposition,

quand la causation est interne, cela signifie que cela émane d'un des actants du mot-clé. Les exemples (123) et (124) illustrent tout à fait ce propos et leur fonctionnement est schématisé dans la Figure 41.

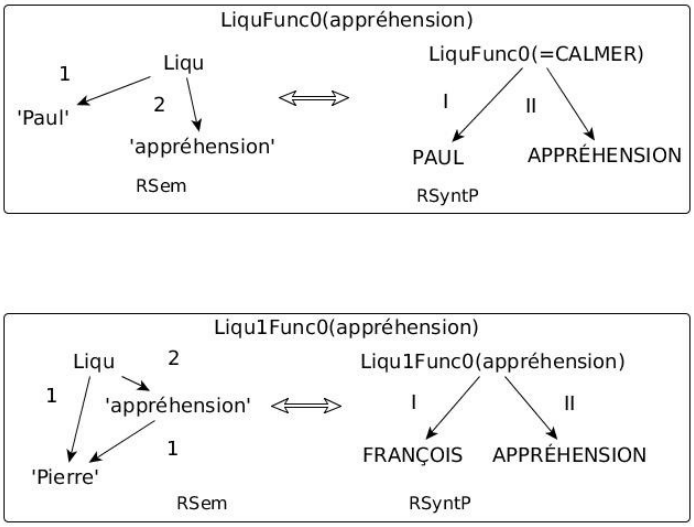



Figure 41. Divergences de RSem entre cause interne et cause externe

Enfin, (I. Mel′čuk et al., 1995, p. 145) mentionnent que la causation est intimement liée à la phase. Les deux se combinent pour former une FL complexe.

> Comme la causation est intimement liée à la phase du fait causé (on cause soit le commencement, soit la continuation, soit la cessation d'un procès, d'un événement, etc.), nous devrions, pour être rigoureux, toujours indiquer la FL phasique correspondante après une FL causative.

Cependant, (I. Mel′čuk, 1996) fournit également des exceptions. Par exemple, CausIncep étant le cas le plus courant de causation, il est possible de simplement écrire Caus. De même, CausFin et LiquCont sont remplacés par Liqu, PermCont par Perm. De ce fait, les combinaisons restantes sont CausCont, PermIncep, PermFin, LiquIncep et LiquFin. La Figure 42 illustre le fonctionnement des FL complexes des exemples (128) et (129).

(128) CausContFact0(curiosité) = entretenir la ~

(129) Caus1ContOper12(espoir) = bercer N d' ~

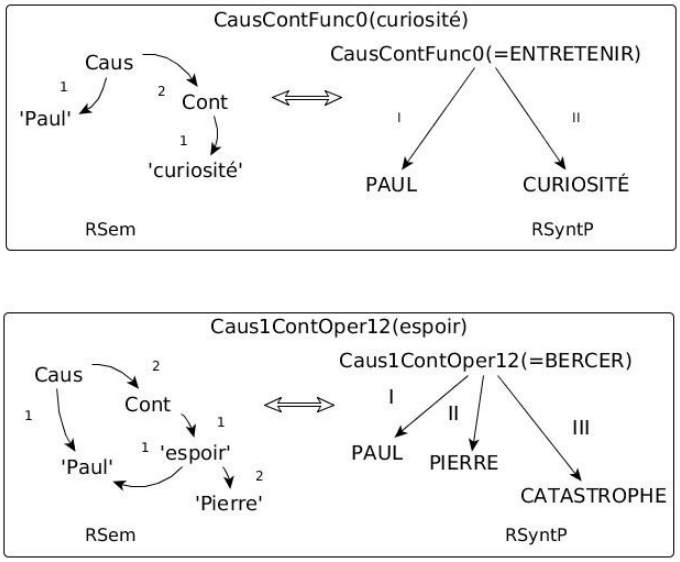



Figure 42. Fonctionnement de FL complexes du type Causation + Phase + Verbe support

Nous avons implémenté des patrons génériques pour rendre compte des combinaisons de FL causatives et phasiques. La Figure 43 est une capture d'écran du patron « lex_causExt_phase_vsupp_0 ».

```
Sem<=>DSynt lex_causExt_phase_vsupp_0 : support_verbs
leftside (∀)
l:?Cl{                                  // FL Caus
  sem=Caus|Liqu|Perm
  l:2-> l:?Zl {}                        // FL phase
  l:1->?Yl {}                           // source
}
l:?Zl {                                 //FL phase
  sem=Incep|Fin|Cont|Prepar|Prox
  l:1-> l:?Xl {}                        //base
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Zr { <=> ?Xl   <=> ?Zl  <=> ?Cl     // FL Caus + Phase + Vsupp
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(X1)-> ?Yr{ <=> ?Yl        // source
  }
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
}
conditions (∃)
?F.name=?Cl.sem+?Zl.sem+"Oper0" or ?F.name=?Cl.sem+?Zl.sem+"Func0";
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Figure 43. Règle générique de combinaison FL causatives + phasiques + verbe support

Cette règle de transduction prévoit toutes les combinaisons possibles: CausIncep, CausFin, CausCont, CausPrepar, etc. Afin d'éviter de sur-générer, le dictionnaire *lf* ne contient que les

combinaisons valides fournies par la TST. De ce fait, lors de l'application des règles, GÉCO ne construit pas d'arbres correspondants aux combinaisons CausIncep, CausFin, etc. en RSyntP. Cette méthode évite de créer des règles de transduction pour chaque cas de figure.

#### 3.3.7.4 Explosion combinatoire

Pour synthétiser, les trois patrons de FL verbales, verbes supports (Vsupp), de réalisation (Vreal) et autres verbes sémantiquement pleins (Vsem), se combinent de manière similaire. Voici les schémas de combinaisons exprimés sous forme d'expression générique :

- (Incep|Fin|Cont|Prepar|Prox|Non) (Vsupp|Vreal|Vsem)
- (Caus|Liqu|Perm) (Vsupp|Vreal|Vsem)
- $(Caus|Liqu|Perm)_i$ (Vsupp|Vreal|Vsem)
- (Caus|Liqu|Perm) (Incep|Fin|Cont|Prepar|Prox|Non) (Vsupp|Vreal|Vsem)
- $(Caus|Liqu|Perm)_i$ (Incep|Fin|Cont|Prepar|Prox|Non) (Vsupp|Vreal|Vsem)

La présence de la causation volontaire, marquée par un indice, génère une réelle explosion combinatoire. Cela explique pourquoi, comme nous le verrons par la suite, nous avons réussi à implémenter plus de 26 000 fonctions lexicales.

### 3.3.8 Dérivation adjectivale

Le dernier patron intéressant est celui orienté autour de la dérivation adjectivale $A_i$.

#### 3.3.8.1 $A_i$ simplement

$A_i$ est défini par (I. Mel′čuk et al., 1995) comme un « modificateur adjectival typique de la lexie $L_2$, en tant qu'argument syntaxique profond I/II/III de la lexie L1 – $A_1$, $A_2$, $A_3$, … ['tel qu'il est …'] ». Voici quelques exemples :

(130) $A_1$(mépris) = rempli de ~

(131) $A_2$(mépris) = couvert de ~

Ces exemples s'illustrent par la Figure 44.

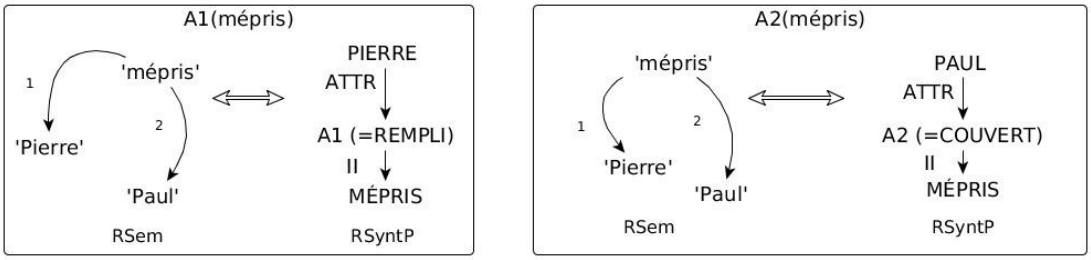



Figure 44. Modélisation du fonctionnement de $A_i$

De ce fait, le fonctionnement de $A_i$ peut être synthétisé par le schéma représenté à la Figure 45.

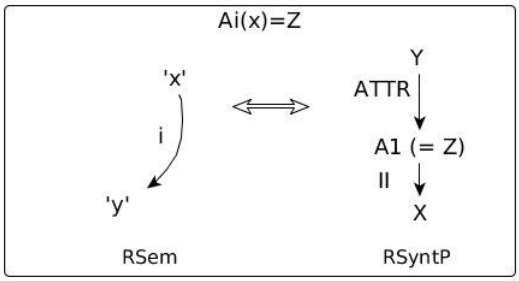



Figure 45. Fonctionnement général de $A_i$

La FL $A_i$ ne possède pas de sens, car elle permet de dériver la base pour former un modificateur adjectival de son i-ème actant. Cette FL est particulièrement intéressante en combinaison avec d'autres FL.

### 3.3.8.2 $A_i$ et FL verbales

La FL $A_i$ peut se combiner avec plusieurs FL différentes, en particulier celles introduisant un collocatif verbal. $A_i$ est décrite dans la littérature comme une fonction lexicale dérivationnelle. Lorsqu'on l'emploie pour former une FL complexe, ou pour la rajouter à une FL complexe existante, $A_i$ impose son patron syntaxique comme l'indiquent (I. Mel′čuk et al., 1995, p. 149) :

> Avec les FL dérivationnelles, les indices renvoient aux AsyntP de la FL précédente. Ainsi, dans Adv1Real2(invitation), l'indice 2 de Real réfère à l'AsyntP II de la lexie INVITATION, c'est-à-dire à l'invité; cependant, l'indice 1 de Adv réfère à l'AsyntP I de la FLReal, et non pas à l'AsyntP I d'INVITATION – c'est-à-dire, toujours à l'invité.

Commençons par traiter les cas de combinaison de $A_i$ avec une FL simple introduisant un collocatif verbal,

(132) $A_1Real_2$(surcre) = [Actant 2] au ~

(133) $A_2Real_1$(fauteuil) = ~ occupé par [Actant 1]

(134) $A_1Fact_2$(charrette) = ~ chargée de [Actant 2]

(135) $A_2Fact_1$(drap) = [Actant 1] recouvert de ~

Comme le montrent les exemples (132) à (135), deux comportements syntaxiques sont à l'œuvre ici représentés par les schémas de la Figure 46.

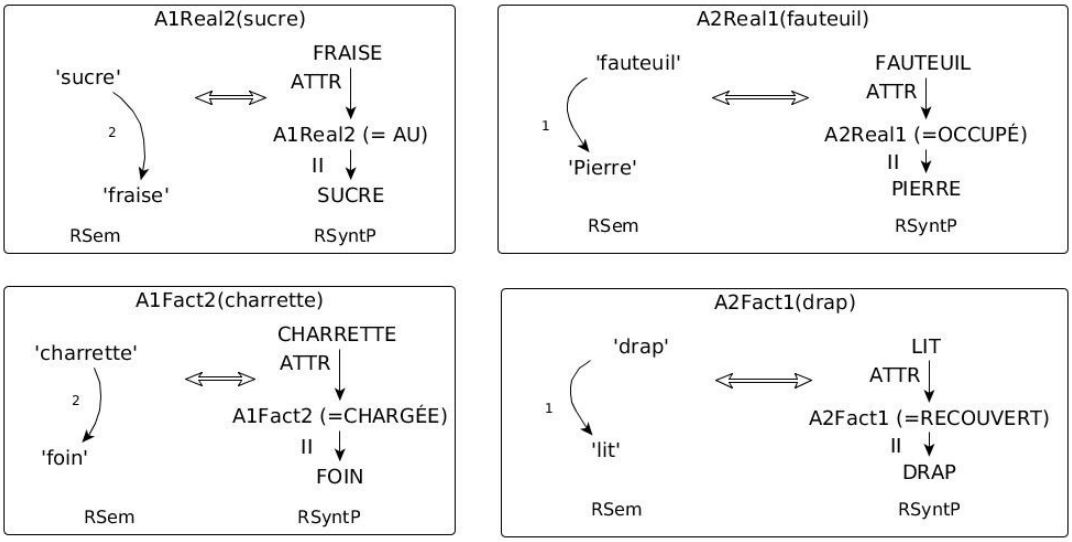



Figure 46. Différences de fonctionnement entre $A_iReal_j$ et $A_iFact_j$

Dans le premier cas, $A_1Fact_2$ insère un nœud gouverné par une relation attributive par la base. Ce nœud lui-même possède un argument II, la lexicalisation de l'argument de la base. Dans le second cas, $A_2Fact_1$ insère un nœud régi par une relation attributive avec la lexicalisation de l'actant 1 de la base. Ce nœud possède un argument II, la base. Dans le cas de $A_1Real_2$ et $A_2Real_1$, il en va de même : dans un cas c'est l'actant qui régit le nœud introduit par la FL en RSyntP et dans l'autre, la base domine le nœud introduit par la FL.

Pour en arriver à ce résultat, nous sommes obligés de découper la lexicalisation en deux étapes, comme le montre la Figure 47 : la lexicalisation de la fonction lexicale Fact, dans le cas présent, puis l'application de la fonction lexicale $A_i$.

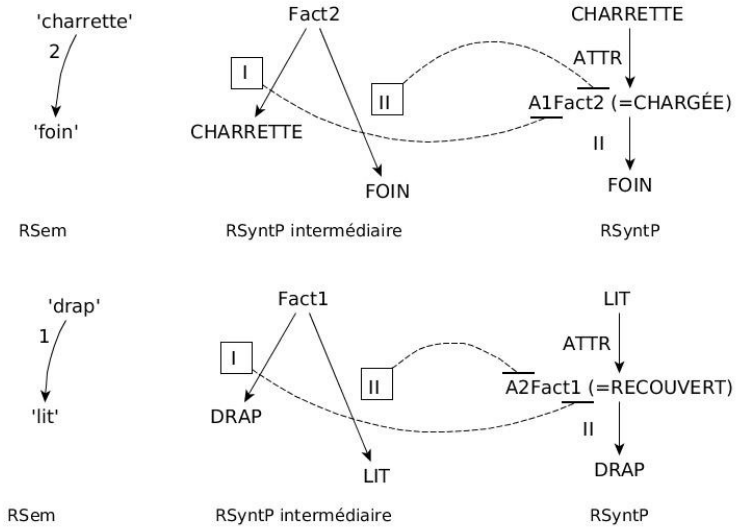



Figure 47. $A_1fact_2$ et $A_2Fact_1$

Cela vient illustrer le propos de (I. Mel′čuk et al., 1995) mentionné plus haut. Dans le cas de $A_1Fact_2$, comme l'argument I de Fact est la base, alors la valeur de $A_1Fact_2$ sera un modificateur de la base. Dans le cas de $A_2Fact_1$, comme l'argument II de Fact est la lexicalisation d'un actant de la base, alors la valeur de $A_2Fact_1$ sera un modificateur de cette lexie. Dans le cas de Real, puisque la base en RSyntP est Lactant II du collocatif verbal, alors le fonctionnement est inversé, voir Figure 48.

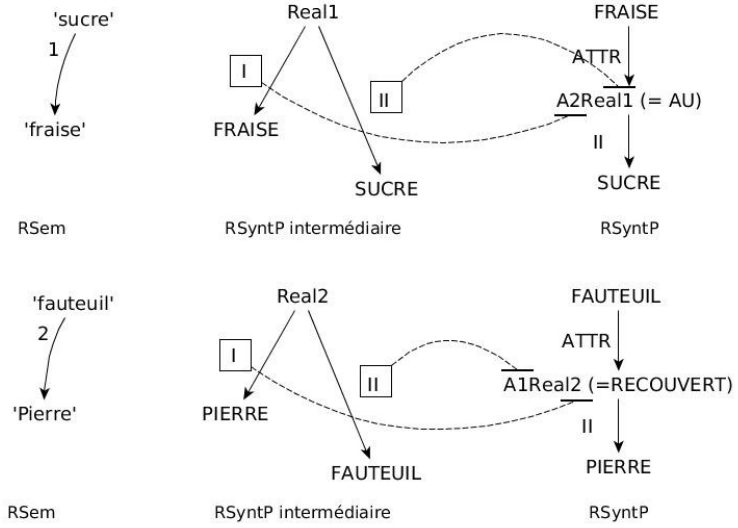



Figure 48. $A_1Real_2$ et $A_2Real_1$

Quand la valeur de $A_i$ s'applique sur la lexicalisation de la base, il est possible que le nœud modificateur inséré ne gouverne pas de lexie, comme c'est le cas par exemple de $A_1Fact_0$(cigarette) dont la valeur est ALLUMÉE.

Nous n'avons pas intégré d'étape de lexicalisation supplémentaire dans GÉCO. Nous avons généralisé le fonctionnement de $A_i$ en combinaison avec une FL verbale de la manière suivante, voir Figure 49. Il existe deux patrons représentant $A_i$ en combinaison avec une FL verbale : celui où la FL complexe modifie la base et celui où la FL complexe modifie l'actant de la base.

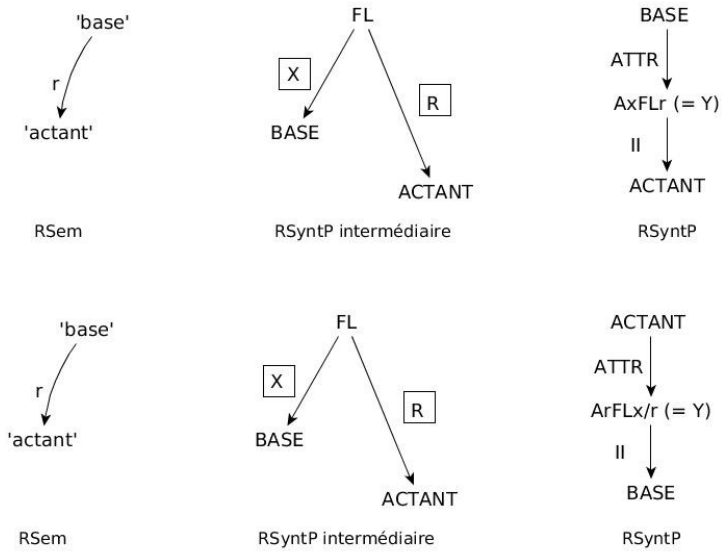



Figure 49. Double fonctionnement de $A_i$ en combinaison avec une FL verbale

Il est aussi possible d'appliquer $A_i$ sur une FL verbale complexe. Le processus est le même. Cependant, il faut noter que, comme les causations intègrent un nouvel actant dans la RSyntP, cela modifie le comportement de $A_i$. Encore une fois, la structure est décalée. Dans ce cas, une FL complexe de dérivation adjectivale comportant un élément causatif et ayant comme indice la valeur « 1 » ($A_1CausReal_2$, $A_1LiquFact_3$, etc.) modifiera toujours l'élément causateur.

Chacun de ces cas possède sa règle de transduction et ce pour chaque type de FL verbale. Il en résulte que le patron $A_i$ est celui contenant le plus de règles de transduction, 68 au total.

### 3.3.9 Fonctions Lexicales non classées

Plusieurs FL n'ont pas pu être classées car leur fonctionnement est unique. Nous les présentons une à une.

#### 3.3.9.1 Epit

Cette FL introduit un adjectif épithète en relation attributive avec la base dans la RSyntP. C'est une FL sémantiquement vide, c'est-à-dire qu'elle n'est pas présente en RSem. Voici quelques exemples d'emploi :

(136) Epit(océan) = ~ immense

(137) Epit(défier) = ~ ouvertement

(138) Epit(gagnant) = ~ heureux

La Figure 50 schématise son fonctionnement. Epit a été introduite à l'aide de la règle « lex_Epit ».

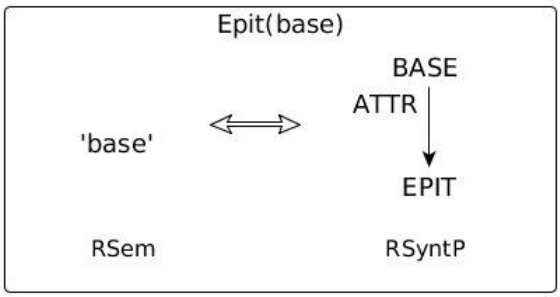



Figure 50. Fonctionnement de Epit dans GÉCO

#### 3.3.9.2 Figur

Figur introduit un nom métaphorique régissant la base. Il s'agit également d'une FL vide de sens et n'est également pas présente en RSem. Voici quelques exemples ainsi que la représentation de son fonctionnement dans Géco, Figure 51. Le patron « lex_Figur » encode cette FL dans Mate.

(139) Figur(fumée) = rideau de ~

(140) Figur(jalousie) = démon de la ~

(141) Figur(haine) = écran de ~

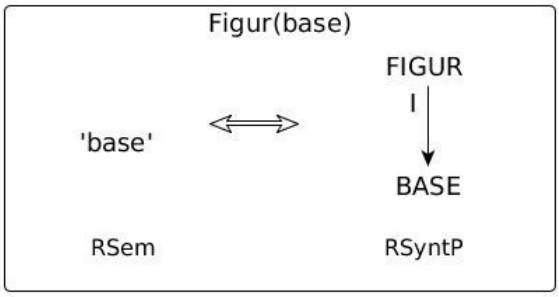



Figure 51. Fonctionnement de Figur dans Géco

#### 3.3.9.3 Involv

Involv est une FL particulière. (I. Mel′čuk et al., 1995, p. 115) la définisse comme suit :

> Verbe d'implication Involv : verbe ayant le sens 'impliquer, affecter' qui a le mot-clé L comme sujet grammatical [=SG] et dont le complément d'objet central est le nom de l'entité impliquée dans la situation 'L' sans en être un actant « légitime »; autrement dit, c'est une action de L qui n'est pas prévue par sa définition mais qui est pourtant assez typique de L pour avoir une expression particulière.

Voici quelques exemples d'emploi, tirés du même ouvrage. La Figure 52 présente la modélisation du fonctionnement de Involv dans Géco tel qu'il est encodé dans « lex_Involv ».

(142) Involv(soleil) = ~ inonder [la pièce]

(143) Involv(odeur) = ~ remplir [la pièce]

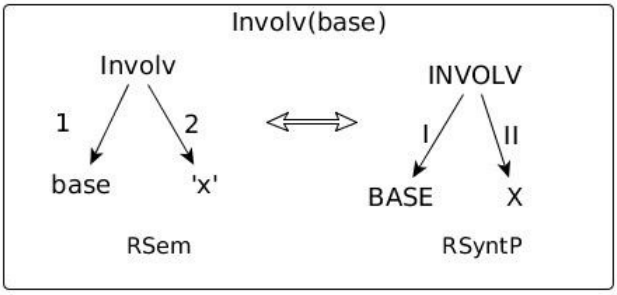



Figure 52. Modélisation de la FL Involv

### 3.3.10 Fonctions lexicales non traitées

Plusieurs fonctions lexicales n'ont pas été implémentées. Nous les présentons maintenant.

Copul et Pred sont des FL permettant d'introduire un verbe ayant le sens 'être'. De même, pred n'apparait qu'en combinaison avec d'autres FL. Nous ne les avons pas intégrés car nous pensons que les verbes supports permettent déjà d'intégrer une coquille syntaxique verbale dans la phrase, voire une copule.

De même, nous n'avons pas intégré la dérivation adverbiale $Adv_i$. $Adv_i$ est semblable à $A_i$ dans le sens où cette FL introduit un modificateur de l'actant « i » de la base en RSyntP. Cependant, $Adv_i$ introduit un modificateur adverbial, comme ci-dessous :

(144) $Adv_1$(mépris) = avec ~

(145) $Adv_1$(chercher) = en quête de [Actant I]

(146) $Adv_1$(joie) = avec ~

(147) $Av_2$(joie) = à la ~

Nous avons eu des problèmes pour modéliser le fonctionnement de $Adv_i$. Premièrement, il est difficile de voir sur quel élément se rattache le modificateur introduit par $Adv_i$. C'est le cas de l'exemple (145). Nous avons donc décidé de laisser cette fonction lexicale de côté et d'attendre que son fonctionnement soit mieux décrit dans les ouvrages de référence.

## 3.4 Conclusion

Dans ce chapitre, nous avons présenté GÉCO, le réalisateur de texte multilingue basé sur l'architecture de MARQUIS. Son architecture se base sur les principes de la Théorie Sens-Texte à la fois pour représenter les connaissances linguistiques, mais aussi dans la modélisation des collocations. La réalisation profonde part d'une représentation conceptuelle du texte à produire et dérive cette représentation sur plusieurs niveaux jusqu'à obtenir le texte final. Le processus de réalisation s'appuie sur plusieurs dictionnaires riches. Ces dictionnaires englobent différents types d'information linguistique.

Les collocations sont introduites dans GÉCO à l'aide des fonctions lexicales. Les fonctions lexicales sont introduites dans l'énoncé à produire lors du passage entre la représentation sémantique du texte et sa représentation syntaxique profonde. Ce passage s'effectue à l'aide des règles de transduction. Les règles de transduction forment la grammaire de GÉCO. Afin de lexicaliser les fonctions lexicales appropriées, GÉCO puise l'information pertinente dans trois

dictionnaires : (i) le *semanticon* liste les lexicalisations possibles de chaque sémantème, (ii) le *lexicon* recense toutes les entrées lexicales d'une langue et (iii) le dictionnaire *lf* englobe toutes les informations sur les fonctions lexicales et leur fonctionnement en RSyntP.

Nous avons également présenté notre méthodologie d'implémentation des fonctions lexicales dans ce réalisateur. Chaque fonction lexicale standard syntagmatique simple et complexe a été modélisée. De ce fait, nous avons découvert que plusieurs présentaient un comportement similaire en ce qui concerne leur comportement en RSyntP mais aussi en fonction de leurs capacité de composition de fonctions lexicales complexes. Cela nous a poussé à regrouper les fonctions lexicales dans des patrons génériques. Chaque patron s'instancie par une règle de transduction dans GÉCO.

Nous allons maintenant présenter les résultats de cette implémentation.

# Chapitre 4 Implémentation des collocations

Ce dernier chapitre présente une analyse de l'implémentation des fonctions lexicales (FL) dans GÉCO. Nous commençons par présenter l'ensemble des règles de transduction créées dans Géco et indiquons le degré de productivité de chacune. Par la suite, nous revenons sur les difficultés liées à l'implémentation des FL. Dans la deuxième section, nous procédons à l'évaluation de l'implémentation. Cette évaluation se fait en deux temps. Nous évaluons dans un premier temps la couverture de notre implémentation. Dans un second temps, nous évaluons la justesse de nos règles de transduction. Enfin, la dernière section analyse ces résultats et conclut sur les possibilités d'amélioration.

## 4.1 Implémentation

Cette section présente les résultats de l'implémentation. Chaque règle de transduction est fournie en Annexe.

### 4.1.1 Résultats principaux

Nous avons regroupé les FL sous 10 patrons génériques. À ces patrons correspondent 129 règles de transduction prenant en charge les FL simples et complexes. Cette organisation des FL couplée à la généricité des règles de transduction nous permet de générer 26259 fonctions lexicales en tout. Il s'agit d'une avancée majeure sur le traitement des fonctions lexicales dans un système de génération. Le Tableau VII présente le nombre de FL générées pour chaque patron de fonction lexicale.

| Patron | règles de transduction | Ratio en règles | FL générées | Ratio en FL |
|---|---|---|---|---|
| Modification | 1 | 0.77 | 8 | 0,03 |
| Préposition | 1 | 0.77 | 5 | 0,02 |

| | | | | |
|---|---|---|---|---|
| Nom gouverneur sémantique | 2 | 1.55 | 32 | 0,12 |
| Figur | 1 | 0.77 | 1 | 0,003 |
| Epit | 1 | 0.77 | 1 | 0,003 |
| Involv | 1 | 0.77 | 1 | 0,003 |
| Verbes supports | 18 | 13.95 | 1827 | 6,9 |
| Verbes de réalisation | 18 | 13.95 | 3625 | 13,8 |
| Autres verbes sémantiquement pleins | 18 | 13.95 | 8211 | 31,2 |
| Dérivation Adjectivale | 68 | 52.71 | 12548 | 47,7 |
| Total | 129 | ≈ 100 | 26259 | ≈ 100 |

Tableau VII Résultats généraux

Ces résultats montrent que certains patrons sont plus productifs que d'autres. Par exemple, alors que les patrons Verbes de réalisation et Autres verbes sémantiquement pleins possèdent le même nombre de règles, le deuxième génère beaucoup plus de fonctions lexicales.

On remarque que la moitié des règles de transduction et des FL générées sont des dérivations adjectivales ($A_i$). Cela s'explique par le fait que $A_i$ peut se combiner avec presque n'importe quelle autre FL. Cependant, les fonctions lexicales formant des dérivations adjectivales ne sont probablement pas les FL les plus usitées empiriquement parlant. C'est pourquoi nous avons évalué l'exhaustivité de notre implémentation à la fois par rapport au nombre de FL *per se*, mais aussi en fonction de la représentativité de chaque FL. Avant d'aborder l'évaluation en tant que telle, nous revenons sur les difficultés liées à l'implémentation.

### 4.1.2 Difficultés techniques

La difficulté principale liée à cette implémentation est liée à la récupération de valeurs d'attributs dans les règles de transduction. Autrement dit, nous avons été obligés d'indiquer directement dans les patrons les valeurs de certaines FL. La Figure 53 est une capture d'écran de la règle de transduction « lex_vsupp_0 » permettant de générer les FL $Oper_0$ et $Func_0$.

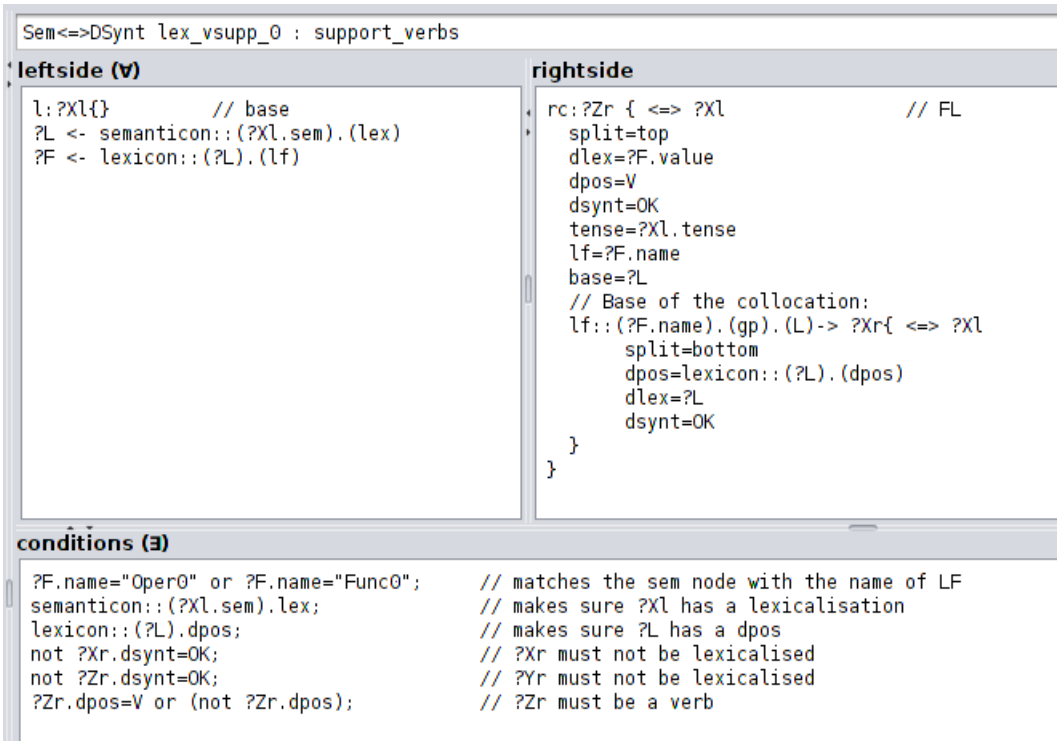



Figure 53. Règle de transduction « lex_vsupp_0 »

La partie gauche de cette règle spécifie un nœud `?Xl`. Ce nœud peut s'instancier par n'importe quel sémantème. La partie gauche spécifie également deux variables, `?L` et `?F`. La variable `?L` permet de récupérer la lexicalisation du sémantème instancié au nœud `?Xl`. Cette information est indiquée dans le *semanticon*. Par la suite, la variable `?F` récupère la valeur instanciée de `?L` et pointe vers l'inventaire de fonctions lexicales contenues dans l'entrée du *lexicon* correspondant à `?L`. Par exemple, si le nœud `?Xl` est étiqueté par le sémantème 'augmentation', la variable `?L` s'instanciera par AUGMENTATION et la variable `?F` contiendra l'inventaire des FL de l'entrée `augmentation` dans le *lexicon*. À la base, `?F` incluait le chemin suivant : `lexicon ::(?L).(lf).(name)` . Autrement dit, `?F` devait

récupérer directement le nom des FL. Cependant, MATE ne parvient pas à récupérer la valeur de l'attribut « name » dans le *lexicon* empêchant ainsi d'instancier dynamiquement les noms des FL. C'est pourquoi nous avons dû inclure des références directes aux noms des FL dans les conditions d'application. Dans le cas de la règle en Figure 53, l'instruction `?F.name=''Oper0'' or ?F.name=''Func0'';` permet de forcer la variable `?F` à pointer vers $Oper_0$ ou $Func_0$. Cela implique de rendre nos règles de transduction très dépendantes des noms des FL. Par conséquent, nous ne pouvons pas créer de règles de transduction plus génériques.

Le deuxième enjeu technique consiste à attribuer les bonnes correspondances en RSyntP. La Figure 54 présente la règle de transduction « lex_causExt_phase_vreal_0 » permettant de générer des FL telles que $CausContReal_0$, $PermIncepFact_0$, etc.

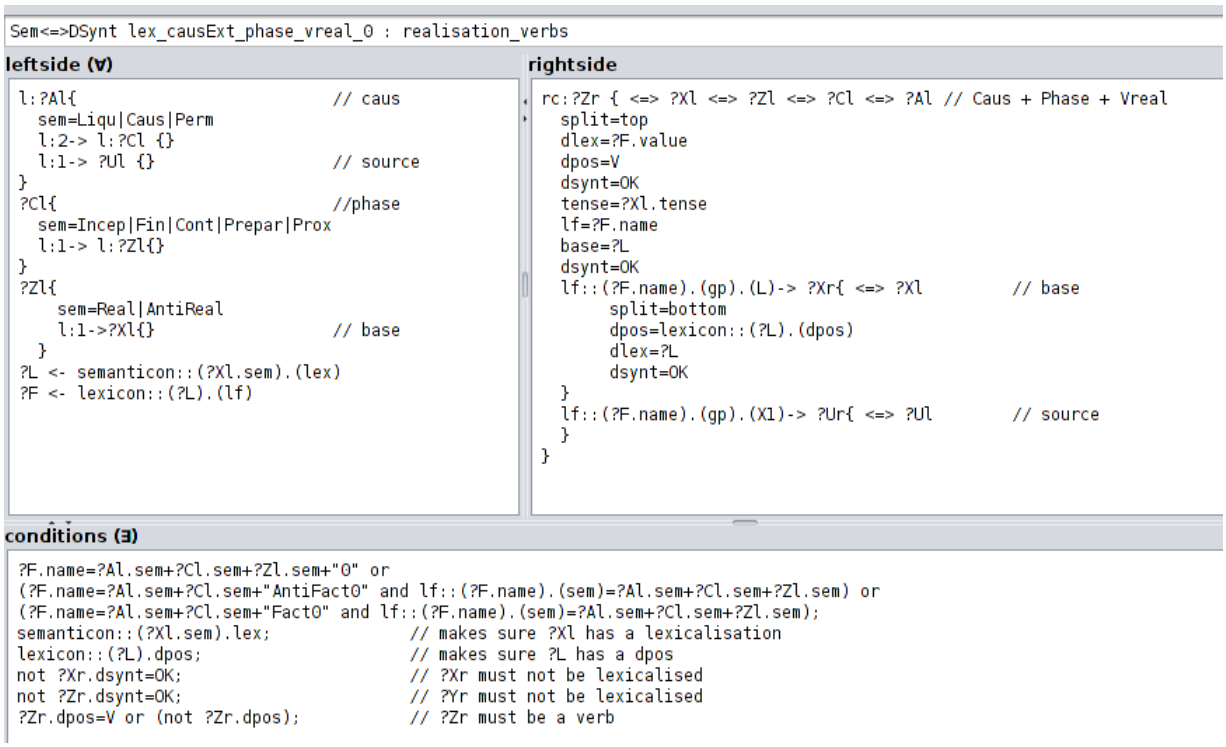



Figure 54. Règle de transduction « lex_causExt_phase_vreal_0 »

La partie gauche de cette règle est assez complexe. Comme on peut le voir, le sommet syntaxique de l'arbre en RSyntP, le nœud `?Zr` possède plusieurs correspondances en RSem, ce qui est marqué par l'instruction `<=> ?Xl <=> ?Zl <=> ?Cl <=> ?Al`. Il s'agit des nœuds suivants : (i) le nœud `?Al` étiqueté par le nom de la FL de causation (Caus|Liqu|Perm),

(ii) le nœud `?Cl` étiqueté par le nom de la FL phasique (Incep|Cont|Fin|Prox|Prepar|Non), (iii) le nœud `?Zl` étiqueté par le nom de la FL de réalisation (Real|AntiReal) et (iv) le nœud `?Xl` étiqueté par la base.

Lors de l'élaboration de ce patron, nous nous sommes aperçu que Mate ne permet pas de faire correspondre un nœud en RSyntP à plusieurs nœuds en RSem à moins de spécifier explicitement chaque point d'entrée du graphe. Autrement dit, la partie gauche de la règle de transduction doit non seulement présenter les nœuds du graphe d'entrée de manière enchâssée, comme dans l'exemple (148), mais il faut en plus spécifier à nouveau ces nœuds comme point d'entrée du graphe, comme le montre l'exemple (149) :

(148) 3 nœuds enchâssés :

```
?Xl {
  1-> ?Yl {
      1-> ?Zl {}
  }
}
```

(149) 3 nœuds comme points d'entrée :

```
?Xl {
  1-> ?Yl {}
}
?Yl {
  1-> ?Zl {}
}
?Zl{}
```

Ces deux exemples représentent la même information : un graphe est formé de trois nœuds, `?Xl`, `?Yl` et `?Zl` et ces nœuds sont enchassés. Cependant, MATE ne traite pas ces représentations de la même manière. Seule la notation de l'exemple (149), plus lourde, permet d'effectuer les correspondances multiples en RSyntP.

Ces difficultés n'ont pas posé de problèmes majeurs à l'implémentation. Nous allons maintenant procéder à l'évaluation.

## 4.2 Évaluation

L'objectif de ce mémoire était de faire le traitement le plus exhaustif possible des collocations dans un générateur automatique de texte multilingue à l'aide des fonctions lexicales. L'évaluation de ce travail est faite en deux temps : l'évaluation (i) du rappel et (ii) de la justesse de notre implémentation (précision).

### 4.2.1 Rappel

Afin de tester la couverture de notre implémentation, nous avons jugé utile de comparer notre liste de fonctions lexicales, contenant 26 259 entrées, avec les fonctions lexicales standard syntagmatiques, simples et complexes, attestées dans plusieurs ressources : le DicoInfo (L'Homme, 2002; L'Homme, 2009), le DicoEnviro (L'Homme & Lanneville, s. d.) et DicoLilex (Marengo & Robichaud, 2016).

La première étape de cette évaluation a été de récupérer la liste des FL standard décrites dans ces ressources. Benoit Robichaud, chercheur à l'OLST, a fourni ces listes dans un tableur. La liste des trois ressources combinées, que nous appellerons maintenant DicoFusion, présentent 4 848 FL standard pour un total de 46 663 occurrences de ces FL. Avant d'entamer l'évaluation, nous avons trié cette liste manuellement. DiCoInfo et DiCoEnviro présentent un mécanisme de vérification des fonctions lexicales à l'aide d'un statut : les FL vérifiées et correctes sont annotées '0', alors que celles non vérifiées ont un statu de '1' et plus. L'extraction des FL de ces ressources ne prend pas en compte ces status, ce qui explique une grande partie des erreurs.

En effet, comme nous nous intéressons à l'implémentation des collocations comme phénomène de combinatoire lexicale, nous n'avons modélisé que des FL syntagmatiques. Or la liste DicoFusion contient des FL standard à la fois syntagmatiques et paradigmatiques. Nous avons donc retiré ces dernières de DicoFusion, soit plus de 1300 FL. De même, Certaines FL complexes ont dû être retirées à cause de leur composante de dérivation, voir (150) et (151), les rendant paradigmatiques.

(150) $S_0Real_1$(balai) = balayage

(151) $A_0CausFunc_0$(sucre) = sucrier [adjectif]

Comme nous avons décidé de n'implémenter que les FL simples et complexes, les configurations de FL ont également été retirées de DicoFusion, représentant près de 550 FL.

Ce premier tri, a mis en avant l'abondance de FL non-standard, semi-standard et localement-standard. Ces FL représentent près de 1 100 entrées dans DicoFusion. Voici quelques exemples attestés :

(152) Jetable et faite de carton (assiette) = papier

(153) De_nouveau$Oper_{12}$(avertissement.2) = réitérer

(154) en échange de qqch. $CausOper_1$(silence) = acheter [le de qqn.]

Comme cela a été mentionné dans le Chapitre 3, ces FL n'ont pas fait partie de l'implémentation à cause de leur caractère non récurrent d'une part et de la question de leur modélisation en représentation sémantique d'autre part. Nous les avons donc retirées de DicoFusion. Cependant, ce tri a posé quelques difficultés. En effet, la Théorie Sens-Texte intègre un certain nombre d'exposants pouvant s'ajouter à certaines fonctions lexicales afin de spécifier la composante de sens sur laquelle porte la fonction lexicale, comme le montre l'exemple (155). Cependant, l'exemple (156) montre que certaines fonctions lexicales de DicoFusion présentent aussi des spécifications sur leur sémantisme.

(155) Magn-actual(faimI.1a)= de loup // fringale

(156) Mult-vente(clou)= boîte

(I. Mel′čuk et al., 1999) ne précisent pas la nature des exposants répertoriés : représentent-ils des unités de sens universelles ? De même, les auteurs ne semblent pas remettre en question le caractère standard de ces fonctions lexicales. Nous avons décidé de les garder dans la liste DicoFusion sur la base qu'ils ne font qu'apporter une précision sur la composante sémantique sur laquelle porte la fonction lexicale, et que cette composante ne rajoute pas de sens à la fonction lexicale en tant que tel, par opposition aux exemples (152) - (154).

Enfin, la dernière étape du nettoyage a consisté à retirer les coquilles d'annotations de DicoFusion. Le Tableau VIII présente les résultats de l’évaluation.

| | DicoFusion | Trouvées par Géco | Couverture |
|---|---|---|---|
| Nombre de FL | 1 563 | 331 | 21 % |
| Nombre d'occurrences | 18 939 | 13 118 | 69 % |

Tableau VIII Couverture de l'implémentation

Ces résultats sont sujets à plusieurs interprétations. Tout d'abord, ces deux listes n'ont pas le même ordre de grandeur, ce qui fausse les résultats. La liste DicoFusion représente des fonctions lexicales standard syntagmatiques, simples et complexes, alors que la liste de GÉCO fait état de toutes les fonctions lexicales standard syntagmatiques, simples et complexes, possibles d'un point de vue multilingue. Il est donc normal que la liste de FL implémentées dans GÉCO contienne beaucoup de FL non présentes dans DicoFusion.

Notre implémentation ne couvre que 21 % des FL de DicoFusion pour un total de 69 % d'occurrences. La faible couverture des FL s'explique par le fait que notre implémentation n'intègre pas les exposants et autres spécifications de sémantisme. Les exemples du Tableau IX sont un échantillon des FL présentes uniquement dans DicoFusion :

| | | | | |
|---|---|---|---|---|
| $A_1Real_1$--ii | Magn--ancienneté | Magn--temp | Magn-activité | $Prepar_1Fact_0$[man:en le reliant à 2] |
| $A_1Real_2$--i | Magn--diamètre | Magn-[impact sur y] | Magn-alcool | $Prepar_1Fact_0$ [phas:ii] |
| $A_1Real_2$--i/ $A_1Real_2$--ii | Magn--dimension | Magn-[résolution de x] | Magn bruit | $Prepar_1Fact_0$ [man:en le reliant à 3] |

| | | | | |
|---|---|---|---|---|
| $A_1Real_{23}$—i | Magn--quant | Magn-['conséquences'] | Magn-comportement | $Prepar_1Fact_0$ [man:en le reliant à l'ordinateur] |
| Magn speed | Magn--robuste | Magn-action | Magn-conséquence | temporairement $LiquFunc_0$ |
| Magn vitesse | Magn--usual | Magn-actions | NonOper1--actual | [mult] |

Tableau IX Quelques FL non traitées par GÉCO

Comme on peut le voir, beaucoup de silence est dû à la présence des exposants ou marques de sémantisme. En effet, nous avons préféré garder un certain degré de généralisation dans notre implémentation. Pourtant, la présence des exposants n'altère en rien le sens de la FL.

Par ailleurs, malgré la faible précision de la couverture des FL, notre implémentation couvre deux tiers des occurrences des FL de DicoFusion. En effet, certaines fonctions lexicales sont très représentées. Par exemple, Magn apparait 1 649 fois et $Oper_1$ apparait 1 305 fois. De même, 855 FL, soit la moitié de la liste, ne sont représentées qu'une fois dans DicoFusion. Ceci est en partie dû à l'utilisation des exposants.

Nous avons normalisé la liste DicoFusion pour retirer les exposants afin d'obtenir une meilleure estimation de la couverture de notre implémentation. Voici des exemples de normalisation :

(157) Locin-temp devient $Loc_{in}$

(158) Real1-I devient $Real_1$

(159) Real@ devient $Real_0$

(160) $Magn_3$ devient Magn

D'un point de vue appliqué, il ne semble pas utile d'intégrer les exposants dans l'implémentation car les textes à produire ne représenteront probablement pas un tel degré de spécification sémantique. Si l'on se focalise sur l'implémentation comme objet théorique, intégrer ces spécifications sémantiques reviendrait à repenser la modélisation des sémantèmes

et des lexèmes dans GÉCO. Il faudrait inclure les décompositions sémantiques pour chaque unité lexicale dans les représentations sémantiques des énoncés, voire même de créer un *semanticon* constitué d'entrées sous forme de graphes. Cette question fort intéressante ne trouve malheureusement pas de réponse ici mais peut être la base pour un futur travail de recherche. Le Tableau X présente les nouveaux scores.

| | DicoFusion | Géco | Couverture |
|---|---|---|---|
| Nombre de FL | 732 | 361 | 49 % |
| Nombre d'occurrences | 18 939 | 16 001 | 84 % |

Tableau X Couverture de l'implémentation après normalisation

D'après le Tableau X, cette normalisation réduit le taux de silence. En effet, le nombre de FL différentes a chuté de moitié, augmentant ainsi la proportion de FL communes à DicoFusion et GÉCO. De même, cela a eu un impact sur le taux de rappel : notre implémentation couvre à présent près de 85 % des occurrences de FL de DicoFusion. Cependant, certaines fonctions lexicales de DicoFusion demeurent absentes de la liste de GÉCO. En voici quelques-unes :

| | | | | |
|---|---|---|---|---|
| Copul | Pred | $Caus_1Oper_4$ | Result | $A_4$ |
| IncepPredPlus | $Adv_1real_1$ | $Fact_{123}$ | LiquPredAntiVer | AntMagn |
| MultFigur | $Caus_1$ | $LocFunc_0$ | $Func_4$ | Cont |
| $Adv_1$ | $AntiOper_2$ | PredMagn | CausInvolv | $IncepFinFunc_0$ |

Tableau XI Quelques FL manquantes après la normalisation

Parmi ces FL, plusieurs n'ont pas été implémentées dans Géco, comme Pred, Copul ou $Adv_1$. Cependant, ces FL présentent tout de même un taux d'occurrence assez élevé : 151 occurrences pour $Adv_1$, 64 pour Pred seule mais 111 pour IncepPredMinus et 153 pour IncepPredPlus. Ces FL, ainsi que leurs combinaisons, seront intégrées à GÉCO.

Par ailleurs, certaines FL n'ont pas été couvertes en raison de leur indice. C'est le cas de $A_4$, $Func_4$, $Caus_1Oper_4$ et $Fact_{123}$ dans le tableau ci-dessus. L'ajout de ces indices peut se faire facilement, sauf dans le cas de $Fact_{123,}$ car la présence d'un nouvel indice nous oblige à créer une nouvelle règle de transduction permettant d'instancier trois actants en RSyntP en plus de la base.

Les FL Cont, $Caus_1$ et $AntiOper_2$ représentent un autre cas de silence. Ces FL n'ont pas été intégrées dans Géco en raison d'une divergence d'interprétation. En effet, nous nous sommes basés sur (I. Mel′čuk, 1996; I. Mel′čuk et al., 1999, 1995) pour bâtir notre inventaire de FL. Cependant, dans ces ouvrages, les FL phasiques apparaissent uniquement en combinaison avec d'autres FL verbales. C'est pourquoi nous n'avons pas intégré Cont et $Caus_1$ seules. De même, nous n'avons pas intégré de FL complexes formées de Anti et d'un verbe support car les verbes supports étant sémantiquement vide, il ne nous semble pas possible de leur attribuer un antonyme.

Enfin, les FL $IncepFinFunc_{0,}$ CausInvolv et MultFigur présentent des combinaisons de FL que nous n'avons pas intégrées dans GÉCO. Les exemples suivants présentent la généralisation possible de leur principe de combinaison :

(161) (Incep|Cont|Fin) (Incep|Cont|Fin) (Func|Oper|Labor)

(162) (Caus|Liqu|Perm) Involv

(163) (Mult|Cap|Gener|Sing|Equip|Germ) Figur

Avant d'intégrer ces schémas de formation de FL complexes dans Géco, nous allons vérifier leur attestation et leur degré de productivité dans DicoInfo, DicoEnviro et DicoLiLex. Enfin, notre méthodologie, énoncée au Chapitre 3, devrait permettre de rajouter rapidement et facilement les FL manquantes à GÉCO.

Cette évaluation a donc permis de mettre en évidence deux points sur la couverture de notre implémentation. Premièrement, nous avons implémenté plus de fonctions lexicales qu'il y en a d'attestées. En soi, ce n'est pas un problème. L'objectif de cette implémentation était de modéliser et généraliser des comportements récurrents de manière suffisamment abstraite dans une perspective appliquée, celle de la génération automatique de texte multilingue. L'inventaire de fonctions lexicales de GÉCO doit donc rendre compte du plus grand nombre de

phénomènes possible en vue de son utilisation. De ce point de vue-là, notre implémentation est un succès car elle couvre plus de 26 200 fonctions lexicales. Deuxièmement, un certain taux de silence a été attesté en dépit de l'uniformisation des données de test. Notre implémentation ne possède donc pas une assez grande couverture. Cependant, la méthodologie présentée dans ce mémoire permet de modéliser simplement et explicitement le comportement des collocations rendant l'ajout de nouvelles fonctions lexicales dans le générateur facile et rapide. Afin d'améliorer la procédure d'évaluation, plusieurs ressources pourraient être ajoutées à la liste DicoFusion.

Enfin, il est important de mentionner que cette évaluation ne traite que des fonctions lexicales standard syntagmatiques, simples et complexes, et exclut les configurations de fonctions lexicales et les exposants aux fonctions lexicales. Une fois ces éléments intégrés, le décompte des fonctions lexicales traitées pourrait facilement être 10, 20 voire 50 fois supérieur à notre décompte actuel. Ces résultats ont un impact non négligeable sur la notion même de fonction lexicale. Nous aborderons ce point en détail lors de la discussion des résultats (section 4.3).

### 4.2.2 Précision

Tester la justesse de l'implémentation revient à vérifier que les règles de transduction, le *lexicon*, le *semanticon* et le dictionnaire *lf* fonctionnent à l'unisson. Le meilleur moyen de vérifier cela est en évaluant la production de chaque règle de transduction. (Lareau & Wanner, 2007) spécifient qu'il faut à la fois tester le bon fonctionnement d'une règle seule (microtest) et en interaction avec les autres (macrotest). Les tests suivants ont été effectués manuellement. Nous abordons chaque test successivement.

#### 4.2.2.1 Microtest

La procédure de microtest est la suivante : pour chaque règle de transduction, un ensemble de RSem de référence a été créé avec leurs correspondants dictionnairiques. Ces structures doivent être les plus simples possibles afin de cibler le phénomène précis modélisé par la règle. De même, ces structures ne contiennent pas de sens spécifiques à une langue et ne présentent que des étiquettes abstraites. La Figure 55 représente la structure « modification.sem.str » créée pour tester la règle « lex_modification ». Nous avons créé un

total de 129 structures sémantiques, soit une structure par patron. Un ensemble représentatif des structures développées dans le cadre de cette évaluation sont fournies dans une version de démo de GÉCO.

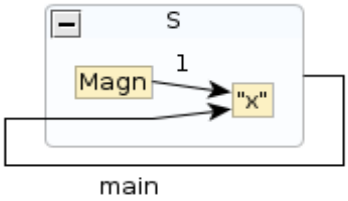



Figure 55. RSem de test « modification.sem.str »

Chaque règle de transduction a été appliquée à l'ensemble de RSem correspondantes. Les résultats ont été enregistrés puis analysés. Comme l'application des règles de fonctions lexicales requiert l'application d'autres règles, notamment celles du package *core*, nous avons suivi la procédure de création de la structure syntaxique profonde de sortie à l'aide de l'outil Inspector de MATE. Par exemple, afin de tester les règles dépendant du patron Dérivation Adjectivale, telle que « lex_anti_non_ai », nous avons intégré dans la RSem un nœud étiqueté par un sémantème se lexicalisant par un verbe, 'v'. En effet, les FL correspondant à ce patron ne s'appliquent que sur des unités lexicales nominales. Or, seul un nœud étiqueté par une unité lexicale verbale peut servir de racine de l'arbre de dépendance introduit en RSyntP. De ce fait, lorsque 'v' se lexicalisera, cette contrainte sera respectée. La Figure 56 représente la structure de syntaxe profonde de test générée par « lex_anti_non_ai ».

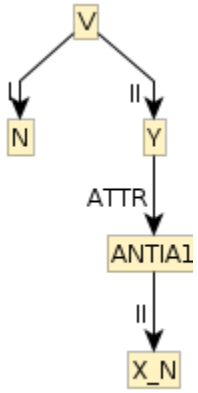



Figure 56. RSyntP produite par « lex_anti_non_ai »

Dans cette RSyntP, le collocatif de la FL introduit par AntiA1 est indiqué en majuscule par l'étiquette ANTIA1. La base est lexicalisée par une unité lexicale nominale, X_N. Le nœud étiqueté par V représente un élément verbal quelconque.

Il est important de mentionner que nous avons testé les règles de transduction directement, et non les FL en tant que tel. Nous partons du principe que lorsqu'une règle de transduction fournit le résultat escompté, toutes les fonctions lexicales rattachées à cette règle sont bien implémentées. Par exemple, si la règle « lex_support_verbs_i » créé un arbre de dépendance correct pour $Oper_1$, alors cette règle est censée créer un arbre de dépendance valide pour les FL $Oper_2$, $Func_1$, $Func_2$, $Labor_1$ et $Labor_2$ qui lui sont attachées. Dans le cas contraire, il est raisonnable de penser que le bogue provient d'une coquille dans le dictionnaire *lf*. Chaque résultat a été évalué manuellement.

Cette évaluation s'est faite de manière *ad hoc* dans le sens où elle a fait partie intégrante du processus d'élaboration des règles de transduction. En effet, lorsqu'une règle ne fournissait pas le résultat attendu, elle a été remaniée jusqu'à présenter une RSyntP valide. Cette évaluation a surtout mis en avant les problèmes techniques énoncés dans la section précédente, en particulier le problème de la réentrance du graphe en partie gauche des règles.

Nous avons employé une terminologie commune d'étiquetage des nœuds en RSem et RSyntP afin de rendre le processus d'évaluation plus facile à suivre :

- 'x' représente toujours le sémantème dénotant la base. Il peut se lexicaliser à l'aide d'un lexème nominal, X_N, ou d'un lexème verbal X_V;
- 'y' représente le sémantème dénotant l'actant sémantique 1 (`?r`) de 'x', lexicalisé par un lexème nominal Y;
- 'z' représente le sémantème dénotant l'actant sémantique 2 (`?l`) de 'x', lexicalisé par un lexème nominal Z;
- 's' représente le sémantème dénotant l'actant sémantique 3 (`?s`) de 'x', lexicalisé par un lexème nominal S;
- 'v' représente un sémantème quelconque se lexicalisant par un lexème verbal V;
- 'n' représente un sémantème quelconque se lexicalisant par un lexème nominal N;

- ‘m’ représente un sémantème quelconque se lexicalisant par un lexème nominal M;

Nous avons employé le système de classes complet du *lexicon* de GÉCO pour les tests.

#### 4.2.2.2 Macrotest

Il existe une certaine interaction entre les patrons de FL. La procédure de macrotest permet de vérifier le bon fonctionnement de l'implémentation comme un tout. Reprenons l’exemple de la RSem faite pour tester la règle de transduction « lex_modification » fournie en Figure 55.

Le sémantème ‘x’ peut se réaliser par un verbe, X_V ou un nom, étiqueté par X. dans la Figure 57. S'il se réalise comme un nom, un verbe support est introduit. Il faut donc que GÉCO prenne en compte l'insertion du verbe support au besoin tout en réalisant correctement le Magn. Trois structures sont alors possibles, voir Figure 57.

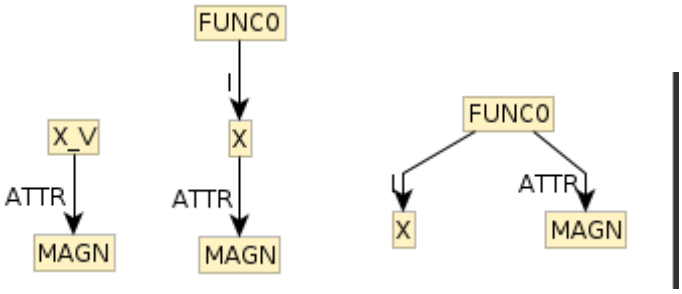



Figure 57. RSyntP fournies par la règle « lex_modification »

D'une certaine manière, cette interaction s'apparente aux configurations de fonctions lexicales à une différence près : les valeurs des configurations de fonctions lexicales sont fusionnées alors que dans les figures ci-dessus, les deux FL se réalisent simultanément.

De même, il arrive qu’une même RSem permette d’intégrer différentes FL en RSyntP. Prenons comme exemple la RSem fournie en Figure 58. Dans cette figure, la base, ‘x’, possède deux actants, ‘y’ et ‘z’. De même, la composante sémantique de la FL ‘Real’ régit la base, cette dernière étant elle-même régie par une FL phasique, ‘Incep’.

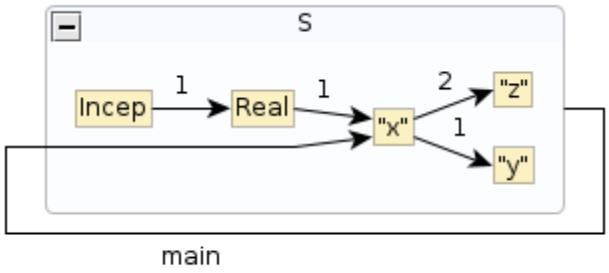



Figure 58. RSem de test

Lorsque l'on applique toutes les règles sur cette RSem, elle renvoie les résultats suivants (Figure 59) :

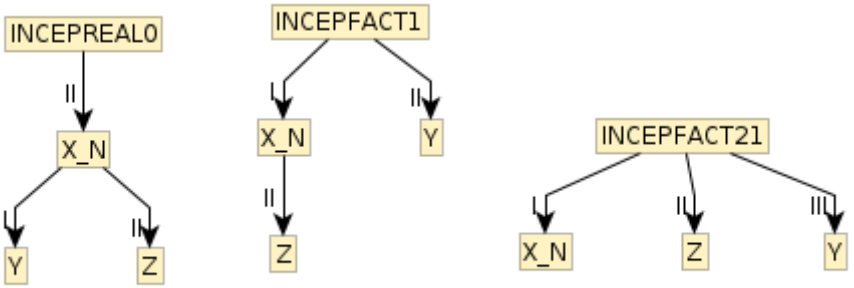



Figure 59. Trois RSyntP possibles

Ces structures sont valides grammaticalement parlant et font intervenir trois règles de transduction différentes : (i) « lex_phase_vreal_0 », (ii) « lex_phase_vreal_i » et (iii) « lex_phase_vreal_ij ». Ces structures diffèrent dans leur traitement des actants de la base. Dans le premier cas, le collocatif ne régit que la lexicalisation de la base en RSyntP. Dans le second, le collocatif introduit par Incepfact1 régit deux éléments : la base et la lexicalisation de son actant '1'. Enfin, dans le dernier cas, le collocatif régit à la fois la base, mais aussi ses deux actants. Il s'agit d'une bonne interaction des règles de transduction car chacun de ces arbres est valide.

Cependant, beaucoup de RSyntP fautives sont également générées. Dans le cas de la Figure 60, une même relation est instanciée deux fois, créant ainsi un arbre de dépendance non valide. Lorsque l'on regarde l'ensemble des règles appliquées pour générer cette structure, on

remarque que deux règles s'appliquent en même temps : « lex_phase_vreal_ij » et « lex_vsupp_0 » créant ainsi la double relation.

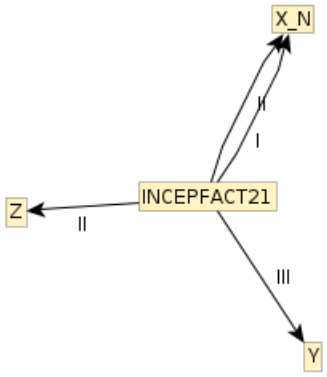



Figure 60. RSyntP fautive 1

En ce qui concerne la Figure 61, il s'agit d'un réel problème de fonctionnement de la part de MATE. Lors du passage entre RSem et RSyntP, MATE a appliqué la règle « lex_ai_vsupp_base_i » sur la lexicalisation nominale de la base, X_N, qui est d'ailleurs le sommet syntaxique de l'arbre de dépendance. Pourtant, les règles du paquet *core* spécifient que seul un nœud étiqueté par un lexème verbal peut servir de racine à l'arbre de dépendance. X_N ne devrait donc pas se retrouver comme sommet syntaxique.

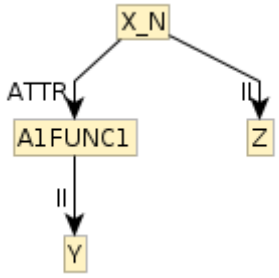



Figure 61. Arbre syntaxique non valide

Enfin, le dernier type de mauvaise interaction entre les règles de transduction est illustré par la Figure 62. Dans cette RSyntP, seul le nœud correpsondant à 'Real' est lexicalisé. De ce fait, le nœud étiqueté par 'Incep' n'est pas du tout traité. Il faudrait probablement revoir le fonctionnement de MATE pour, d'une part, forcer l'application de règles tant que tous les nœuds du graphe en entrée ne sont pas consommés et, d'autre part, inclure une instruction

empêchant MATE de fournir un arbre dont les nœuds ne sont pas équivalents à tous les nœuds contenus dans le graphe en entrée.

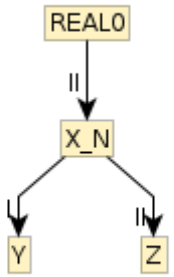



Figure 62. RSyntP incomplète

Malheureusement, nous n'avons pas fait une étude exhaustive de l'interaction de nos règles de transduction. Nous espérons approfondir ce point dans de futurs travaux.

## 4.3 Analyse des résultats

### 4.3.1 Exhaustivité de l'implémentation

Le premier objectif de ce mémoire était de faire une implémentation exhaustive des collocations pour la GATM à l'aide des fonctions lexicales. Nous avons implémenté 26 259 fonctions lexicales différentes à l'aide de 129 règles de transduction. Dans l'absolu, ce résultat est très concluant. Comparativement au travail de (Steinlin, 2003) qui implémentait 3 fonctions lexicales, et à MARQUIS (Bohnet & Wanner, 2010; Lareau & Wanner, 2007; Wanner, Nicklaß, et al., 2007) qui en répertoriait une trentaine, ou à (Lareau et al., 2012) qui en répertoriaient 222, nos résultats sont excellents. Cependant, une fois notre liste comparée aux fonctions lexicales présentes dans différentes ressources (DiCoInfo, DiCoEnviro et DICoLiLex), nous nous sommes rendu compte que notre implémentation ne couvrait que la moitié des FL standard syntagmatiques simples et complexes de ces ressources, après normalisation. De ce fait, nous ne traitons en réalité qu'une petite partie de l'inventaire complet des fonctions lexicales. Toutefois, ces fonctions lexicales couvrent près de 85 % des occurrences de ces ressources. On peut donc en déduire que ce n'est pas le nombre de FL différentes qui importe mais la représentation de ces FL. Autrement dit, un petit ensemble FL,

comme Magn ou $Oper_i$, constitue la majorité des occurrences de ces FL. Cette observation vaut également pour les règles de transduction. « lex_modification » permet de générer huit FL différentes, dont Magn, cependant, en termes d'occurrences, cette règle génère 3256 occurrences, soit environ 17% des occurrences de FL de DicoFusion. De ce fait, l'implémentation doit se focaliser sur l'intégration de ces FL.

GÉCO intègre correctement ces FL. Cependant, quelques-unes doivent être rajoutées. C'est le cas notamment de la FL Pred, lorsqu'elle apparait en combinaison avec d'autres FL, et $Adv_i$. Nous pouvons nous baser sur les résultats de notre évaluation pour déterminer les prochaines FL à implémenter.

### 4.3.2 Les FL comme outil de modélisation des collocations

Le deuxième objectif de ce mémoire concernait la méthodologie de modélisation des collocations à l'aide des fonctions lexicales. Comme nous l'avons mentionné, nous avons regroupé les fonctions lexicales dans des patrons génériques afin de regrouper celles ayant un comportement similaire.

Nous avons mis au jour 10 patrons de FL. Les FL d'un même patron ont un fonctionnement similaire vis-à-vis de leur représentation en RSem et et RSyntP. Autrement dit, elles fonctionnent comme des patrons similaires de lexicalisation. Nous avons également généralisé et appliqué les mêmes règles de formation de FL complexes pour chaque patron. De ce fait, toutes les FL du patron Verbe Support se combinent de la même manière.

Cette généricité de traitement a contribué à l'implémentation de plus de 26 000 FL. L'utilisation des indices, en particulier lors du traitement des FL de causation volontaire, contribue grandement à l'explosion combinatoire des FL. Nous ne nous nous attendions pas à modéliser autant de FL avec si peu de règles. De même, ce nombre risque facilement d'augmenter de manière exponentielle au fur et à mesure que de nouvelles FL sont rajoutées dans GÉCO.

Il existe donc un risque de modéliser plus de fonctions lexicales que d'unités lexicales dans GÉCO, malgré l'aspect multilingue. Cela confirme les intuitions de plusieurs auteurs quant à la proportion de phrasèmes dans les langues. Autrement dit, non seulement existe-il beaucoup

d'occurrences de collocations, mais ces collocations se modélisent à l'aide d'un nombre particulièrement élevé de fonctions lexicales. En effet, chaque fonction lexicale représente une relation sémantico-lexicale spécifique, et il existe un nombre incalculable de ces relations.

Nous pouvons donc nous demander : est-ce que le nombre important de fonctions lexicales dilue leur intérêt pour la GAT? S'il existe autant de fonctions lexicales que de lexèmes dans la langue, chacune représentant une relation spécifique, est-ce que modéliser toutes les FL présentent un intérêt concret pour la GAT? Est-ce même faisable ? La puissance combinatoire des FL est donc un atout à double tranchant : cela permet de modéliser beaucoup de comportements idiosyncrasiques mais présente peut-être un degré de granularité trop fin pour une perspective appliquée.

De ce fait, nous pensons qu'il est utile d'implémenter les FL qui ont un taux d'occurrence élevé dans les langues. Le Tableau XII recense les vingt FL standard syntagmatiques simples et complexes les plus représentées dans DiCoFusion, pour un total de 10 358 occurrences. Parmi ces FL, seule $Prepar_1Fact_0$ n'est pas couverte par GÉCO.

| Fonctions Lexicales | Occurrences | Traitées par GÉCO |
|---|---:|---|
| Magn | 1 922 | Oui |
| $Oper_1$ | 1 346 | Oui |
| Gener | 919 | Oui |
| $Real_1$ | 820 | Oui |
| $Caus_1Func_0$ | 552 | Oui |
| $Labreal_{12}$ | 526 | Oui |
| $Fact_2$ | 474 | Oui |
| AntiMagn | 453 | Oui |

| | | |
|---|---|---|
| $Loc_{in}$ | 417 | Oui |
| $CausFunc_0$ | 344 | Oui |
| $A_1$ | 343 | Oui |
| $Fact_0$ | 290 | Oui |
| $Real_2$ | 289 | Oui |
| **$Prepar_1Fact_0$** | **268** | **Non** |
| $Oper_2$ | 264 | Oui |
| Ver | 241 | Oui |
| $IncepOper_1$ | 239 | Oui |
| AntiBon | 232 | Oui |
| $IncepReal_1$ | 222 | Oui |
| $A_2$ | 197 | Oui |

Tableau XII Les FL standard syntagmatiques simples et complexes les plus représentées dans DicoFusion

La représentation empirique des FL est un aspect qui n'a pas du tout été pris en compte dans notre méthodologie d'implémentation. En effet, nous avons choisi de nous baser sur les descriptions théoriques des FL dans les ouvrages de référence. Notre objectif était à la fois de modéliser le comportement des FL de manière logique et déductive, mais aussi de ne pas se baser sur les ressources existantes afin d'éviter les divergences d'interprétation et d'emplois des FL. Par exemple, les FL phasiques (Incep, Cont et Fin) et causatives (Caus, Liqu et Perm) ne s'emploient qu'en combinaison avec d'autres FL dans les ouvrages de référence. Pourtant, elles s'emploient seules dans DiCoInfo, DiCoEnviro et DiCoLiLex.

La suite de ce travail doit donc être moins théorique et se focaliser sur l'implémentation des FL les plus fréquentes telles qu'attestées dans les ressources existantes. Cette perspective empirique pourrait par la suite alimenter la description théorique des FL.

Pour conclure, ce mémoire a mis en avant le fait qu'il est possible de modéliser un grand nombre de patrons de collocations au moyen des fonctions lexicales pour la GATM. Notre implémentation est optimisée dans le sens où nous avons encodé un nombre élevé de fonctions lexicales à l'aide de peux de règles de transduction. De même, ces règles sont valables théoriquement pour plusieurs langues. Une évaluation quantitative de cette implémentation a démontré que l'objectif n'était pas d'implémenter le plus grand nombre de FL mais celles ayant le plus haut taux d'occurrence. Nous proposons donc de poursuivre cette implémentation conjointement avec une étude empirique de la représentativité des FL pour déterminer les patrons de collocations les plus utilisés dans les langues.

# Conclusion

La génération automatique de texte (GAT) se répartit en deux modules : la génération profonde et la réalisation linguistique. Un des enjeux de la réalisation linguistique multilingue est la modélisation de phénomènes linguistiques récurrents dans les langues de manière générique afin de pouvoir s'adapter facilement à de nouvelles langues. Les collocations sont un parfait exemple de phénomènes linguistiques récurrents. Cependant, les collocations ont toujours été problématiques en TAL et n'ont pas été beaucoup traitées en GAT. C'est pourquoi ce mémoire avait pour objectif d'implémenter les collocations de manière systématique dans GÉCO, un réalisateur de texte multilingue. Pour ce faire, nous avons fait face à deux questions :

- Comment intégrer les collocations dans le processus de réalisation ?
- Comment modéliser les collocations ?

Le chapitre 1 a fait l'introduction du domaine de la GAT. Nous avons présenté les différentes étapes constitutives du processus de génération et avons présenté deux méthodes de gestion de ressources linguistiques pour la GAT multilingue : le partage de grammaire et l'adaptation de grammaire. Par la suite, nous avons comparé plusieurs réalisateurs. SimpleNLG et JSreal sont des réalisateurs de surface dans le sens où ils partent d'une structure syntaxique. Par opposition, les réalisateurs comme FUF, GENI, RTGen, ou encore MARQUIS font de la réalisation profonde, puisqu'ils partent de structures plus abstraites. Cela offre un plus grand contrôle sur les énoncés générés, notamment grâce à l'étape de lexicalisation. Nous avons décidé de nous baser sur le réalisateur de MARQUIS pour notre implémentation des collocations.

Nous avons ensuite étudié le concept de collocation. Plusieurs définitions existent et se basent tant sur des critères fréquentiels que combinatoires pour déterminer si une unité polylexicale est une collocation ou non. Malheureusement, ces définitions restent trop floues pour être implémentées telles quelles en GAT. Nous nous sommes dirigés vers une approche

fonctionnaliste des collocations : l'approche Sens-Texte. Cette théorie décrit les collocations comme des restrictions de sélection lexicale. Autrement dit, une collocation est une paire de lexèmes, le choix de l'un étant déterminé par le choix de l'autre en fonction de la relation sémantico-lexicale qu'ils entretiennent. Cette relation est exprimée au moyen d'une fonction lexicale qui prend un lexème en argument, la base, et retourne le collocatif approprié. Les fonctions lexicales représentent donc des patrons de collocations. Il existe plusieurs types de fonctions lexicales. Nous avons implémenté celles dont le comportement est systématique, généralisable et universel. Il s'agit des fonctions lexicales standard syntagmatiques, simples et complexes.

L'implémentation des collocations au moyen des fonctions lexicales a été faite dans GÉCO, un réalisateur de texte multilingue basé sur le générateur multilingue MARQUIS. Ces deux réalisateurs organisent le processus de réalisation selon les principes de la Théorie Sens-Texte : ils prennent en entrée une structure abstraite conceptuelle, et dérive cette structure sur plusieurs niveaux de représentation linguistique jusqu'à obtenir la forme finale de l'énoncé au format textuel. GÉCO prévoit six niveaux de représentation linguistique : conceptuel, sémantique, syntaxique profond et de surface, morphologique profond et de surface et textuel.

L'intégration des collocations s'effectue lors du passage entre les niveaux sémantique (RSem) et syntaxique profond (RSyntP). Pour passer d'un niveau à l'autre, GÉCO décrit plusieurs règles de transduction. Ces règles forment la grammaire de la langue. En plus de ces règles, GÉCO utilise trois dictionnaires : un dictionnaire d'unités sémantiques (*semanticon*), un dictionnaire d'unités lexicales (*lexicon*) et un dictionnaire de fonctions lexicales (*lf*). Les règles de transduction récupèrent les informations stockées dans ces trois dictionnaires pour guider le processus de lexicalisation.

Nous avons commencé par implémenter les fonctions lexicales standard syntagmatiques simples. Plutôt que de créer une règle de transduction pour chaque fonction lexicale, nous les avons regroupées en 10 patrons génériques. Chaque patron représente des fonctions lexicales dont le fonctionnement est similaire, c'est-à-dire qu'elles se représentent de la même manière en RSem et en RSyntP. À chaque patron correspond une règle de transduction. Par la suite, nous avons intégré les fonctions lexicales standard syntagmatiques complexes.

Quelques ajustements ont dû être faits par rapport à la théorie. Premièrement, la Théorie Sens-Texte spécifie que les fonctions lexicales n'apparaissent qu'en RSyntP. Cependant, nous avons intégré directement en RSem leur composante sémantique afin de généraliser encore plus leur fonctionnement. De même, nous avons inclus les valeurs des fonctions lexicales en RSyntP alors que la théorie prévoit de ne les introduire qu'en RSyntS.

Notre méthodologie a permis d'implémenter 26 259 fonctions lexicales à l'aide de 129 règles de transduction réparties dans 10 patrons de fonctions lexicales. Ce résultat est impressionant en comparaison aux travaux précédents sur l'implémentation des fonctions lexicales en GAT. Cela s'explique par l'utilisation des indices, en particulier dans le cas de la causation volontaire. Chaque règle de transduction a été testée séparément et en combinaison avec les autres.

Nous avons évalué le rappel de notre implémentation en comparant notre inventaire de fonctions lexicales avec ceux de trois ressources : DiCoInfo, DiCoEnviro et DiCoLiLex. GÉCO couvre la moitié des fonctions lexicales standard syntagmatiques simples et complexes de ces ressources, ce qui se traduit par une couverture de 85 % des occurrences de ces fonctions lexicales. Autrement dit, nous couvrons les fonctions lexicales qui sont les plus représentées dans ces ressources, telles que Magn ou $\text{Oper}_1$. De ce fait, notre inventaire de fonctions lexicales est exhaustif dans le sens où il couvre la grande majorité des occurrences de fonctions lexicales.

Cette implémentation montre à quel point les fonctions lexicales existent en grand nombre, remettant ainsi en question leur utilité pour la GAT. Par ailleurs, notre implémentation étant incomplète, il est possible que l'inventaire complet des fonctions lexicales standard syntagmatiques soit largement supérieur à ce que nous avons établi.

Le travail de ce mémoire apporte donc deux contributions. D'une part, nous avons fait une modélisation à grande échelle des collocations au moyen des fonctions lexicales pour la GAT. Alors que les travaux précédents ne modélisaient que quelques dizaines de fonctions lexicales, nous en modélisons plusieurs milliers. La deuxième contribution est d'ordre plus théorique : il n'existait pas d'inventaire théorique des fonctions lexicales. Notre travail pourra donc servir d'outil pour alimenter les études menées sur les fonctions lexicales.

# Bibliographie


Abeillé, A. (1993). *Les nouvelles syntaxes: grammaires d'unification et analyse du français*. Paris: A. Colin.

Aït-Kaci, H., & Nasr, R. (1986). Login: a logic programming language with built-in inheritance. *The Journal of Logic Programming*, *3*(3), 185-215. https://doi.org/10.1016/0743-1066(86)90013-0

Alonso Ramos, M. (2007). Towards the Synthesis of Support Verb Constructions. In L. Wanner (Éd.), *Selected lexical and grammatical issues in the Meaning-Text Theory: In honour of Igor Mel'cuk* (p. 97-138). John Benjamins Publishing.

Alonso Ramos, M., & Tutin, A. (1996). A Classification and Description of Lexical Functions for the Analysis of their Combinations. In L. Wanner (Éd.), *Lexical Functions in Lexicography and Natural Language Processing* (p. 147-168). Amsterdam/Philadelphia: John Benjamins Publishing.

Andrews, A. D. (2010). Propositional glue and the projection architecture of LFG. *Linguistics and Philosophy*, *33*(3), 141-170. https://doi.org/10.1007/s10988-010-9079-9

Apresjan, J. D., Boguslavsky, I., & Tsinman, L. L. (2007). Lexical Functions in Actual NLP-Applications. In L. Wanner (Éd.), *Selected Lexical and Grammatical Issues in Meaning-Text Theory. In honour of Igor Mel'cuk* (Vol. 84, p. 199-230). John Benjamins.

Avgustinova, T., & Uszkoreit, H. (2000). An ontology of systematic relations for a shared grammar of Slavic. In *Proceedings of the ... International Conference on Computational Linguistics* (Vol. 1, p. 28-34). Saarbrücken, Germany.

Bally, C. (1909). *Traité de stylistique française*. Paris: Klincksieck.

Bateman, J. A. (1997). Enabling technology for multilingual natural language generation: the KPML development environment. *Natural Language Engineering*, *3*(1), 15–55.

Bateman, J. A., Kruijff-Korbayová, I., & Kruijff, G.-J. (2005). Multilingual Resource Sharing Across Both Related and Unrelated Languages: An Implemented, Open-Source Framework for Practical Natural Language Generation. *Res Lang Comput Research on Language and Computation*, *3*(2-3), 191-219.

Bateman, J., Matthiessen, C., Nanri, K., & Zeng, L. (1991). The Re-use of Linguistic Resources Across Languages in Multilingual Generation Components. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 2* (p. 966–971). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Benson, M. (1990). Collocations and General-purpose Dictionaries. *Int J Lexicography International Journal of Lexicography*, *3*(1), 23-34.

Boguslavsky, I., Iomdin, L., & Sizov, V. (2004). Multilinguality in ETAP-3: Reuse of Lexical Resources. In *Proceedings of the Workshop on Multilingual Linguistic Ressources* (p. 7–14). Stroudsburg, PA, USA: Association for Computational Linguistics.

Bohnet, B. (2006). Development Environment forGraph/Tree Transducer Grammar and other linguistic Ressources.

Bohnet, B., Langjahr, A., & Wanner, L. (2000). A Development Environment for MTT-Based Sentence Generators. In *Proceedings of the XVI SEPLN Conference*. Vigo, Spain.

Bohnet, B., Lareau, F., Wanner, L., et al.. (2007). Automatic production of multilingual environmental information. In *Proceedings of the 21st Conference on Informatics for Environmental Protection (EnviroInfo-07), Warsaw, Poland*.

Bohnet, B., & Wanner, L. (2010). Open source graph transducer interpreter and grammar development envrionment. In *Proceedings of the Seventh conference on International Language Resources and Evaluation*. Valletta, Malta: European Language Resources Association (ELRA).

Bollmann, M. (2011). Adapting SimpleNLG to German. In *Proceedings of the 13th European Workshop on Natural Language Generation* (p. 133–138). Stroudsburg, PA, USA: Association for Computational Linguistics.

Bonami, O., & Boyé, G. (2010). La morphologie flexionnelle est-elle une fonction ? In I. Choi-Jonin, M. Duval, & O. Soutet (Éd.), *Typologie et comparatisme. Hommages offerts à Alain Lemaréchal.* (p. 21-35). Louvain, Belgique: Peeters.

Bresnan, J. (2001). *Lexical-Functional Syntax*. Oxford: Blackwell.

Butt, M. (1999). *A grammar writer's cookbook*. Stanford, Calif.: CSLI Publications.

Butt, M., Dyvik, H., King, T. H., Masuichi, H., & Rohrer, C. (2002). The Parallel Grammar Project. In *Proceedings of the 2002 Workshop on Grammar Engineering and Evaluation* (Vol. 15, p. 1–7). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.3115/1118783.1118786

Cruse, D. A. (1986). *Lexical semantics*. Cambridge [Cambridgeshire]; New York: Cambridge University Press.

Dale, R., Moisl, H. L., & Somers, H. L. (2000). *Handbook of natural language processing*. New York: Marcel Dekker.

Dalrymple, M. (2006). Lexical-Functional Grammar. In *Encyclopedia of Cognitive Science*. John Wiley & Sons, Ltd.

Danlos, L. (1987a). The linguistic basis of text generation. In *Proceedings of the third conference on European chapter of the Association for Computational Linguistics* (p. 1-1). Association for Computational Linguistics. https://doi.org/10.3115/976858.976859

Danlos, L. (1987b). *The Linguistic Basis of Text Generation*. Cambridge University Press.

Danlos, L. (1998). G-TAG : un formalisme lexicalisé pour la génération de textes inspiré de TAG. *Traitement Automatique des Langues*, *39*(2), 28 p.

Danlos, L., & Roussarie, L. (2000). La génération automatique de textes. In J.-M. Pierrel (Éd.), *Ingénierie des langues*. Paris: Hermès.

Daoust, N. (2014). *JSreal : un réalisateur de texte pour la programmation web*. Université de Montréal.

Daoust, N., & Lapalme, G. (2015). JSREAL: A Text Realizer for Web Programming. In N. Gala, R. Rapp, & G. Bel-Enguix (Éd.), *Language Production, Cognition, and the Lexicon* (p. 361-376). Springer International Publishing. https://doi.org/10.1007/978-3-319-08043-7_21

Elhadad, M. (1993). FUF: The Universal Unifier. The user Manual Version 5.2. Depratment of Computer Science.

Elhadad, M., & Robin, J. (1996). An overview of SURGE: a reusable comprehensive syntactic realization component. In *Demonstrations and Posters of the 8th International Workshop on Natural Language Generation* (p. 1-4). Herstmonceux, England.

Essers, V. R., & Dale, R. (1998). Choosing a surface realiser: Exploring the differences in using KPML/Nigel and FUF/Surge. In *Proceedings of PRICAI98*. Singapore.

Fellbaum, C. (1998). *WordNet an electronic lexical database*. Cambridge, Mass: Massachusetts Institute of Technology.

Firth, J. R. (1968). A synopsis of linguistic theory, 1930-1955. In F. R. Palmer (Éd.), *Selected papers of J.R. Firth, 1952-59.* Bloomington: Indiana University Press.

Fontenelle, T. (1992). Collocation acquisition from a corpus or from a dictionary : a comparison. In *Proceedings I-II. Papers submitted to the 5th EURALEX International Congress on Lexicography in Tampere* (p. 221-228). Tampere.

Gagné, A.-M. (2015). Guide d'encodage des contraires, DiCoInfo et DiCoEnviro Observatoire de Linguistique Sens-texte (OLST).

Gagné, A.-M., & L'Homme, M.-C. (2016). Opposite relationships in terminology. *Terminology*, *22*(1), 30-51. https://doi.org/10.1075/term.22.1.02gag

Gardent, C., & Kow, E. (2007). GenI, un réalisateur basé sur une grammaire réversible. In *14e conférence pour le Traitement Automatique des Langues Naturelles - TALN 2007* (p. 10 p.). Toulouse, France.

Gardent, C., & Perez-Beltrachini, L. (2010). RTG based surface realisation for TAG. In *23rd International Conference on Computational Linguistics - Coling 2010* (p. 367–375). Beijing, China.

Gatt, A., & Reiter, E. (2009). SimpleNLG: A Realisation Engine for Practical Applications. In *Proceedings of the 12th European Workshop on Natural Language Generation* (p. 90–93). Stroudsburg, PA, USA: Association for Computational Linguistics.

Goldberg, E. (1993). FoG: Synthesizing forecast text directly from weather maps, *Proceedings of 9th IEEE Conference on Artificial Intelligence for Applications*. 156-162.

Hausmann, F. (1985). Kollokationen im Deutschen Woerterbuch: ein Beitrag zur Theorie des lexicographischen Biespiels. In H. Bergenholtz & J. Mugdan (Éd.), *Lexikographie und Grammatik. Niemeyer, Turgen, Germany* (p. 118-129).

Heid, U., & Raab, S. (1989). Collocations in Multilingual Generation. In *Proceedings of the Fourth Conference on European Chapter of the Association for Computational Linguistics* (p. 130–136). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.3115/976815.976833

Heylen, D., Maxwell, K. G., & Verhagen, M. (1994). Lexical Functions and Machine Translation. In *Proceedings of the 15th Conference on Computational Linguistics - Volume 2* (p. 1240–

1244). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.3115/991250.991354

Iordanskaja, L., Kim, M., Kittredge, R., Lavoie, B., & Polguère, A. (1992). Generation of Extended Bilingual Statistical Reports. In *Proceedings of the 14th Conference on Computational Linguistics - Volume 3* (p. 1019–1023). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.3115/993079.993120

Iordanskaja, L., Kim, M., & Polguère, A. (1996). Some Procedural Problems in the Implementation of Lexical Functions for Text Generation. In L. Wanner (Éd.), *Lexical Functions in Lexicography and Natural Language Processing* (p. 279-298). Amsterdam/Philadelphia: John Benjamins Publishing.

Jackendoff, R. (1997). *The architecture of the language faculty*. Cambridge, Mass.: MIT Press.

Bateman, J. (1997). Sentence generation and systemic grammar: an introduction. *Iwanami Lecture Series : language Sciences*, *8*.

Joshi, A. K., & Vijay-Shanker, K. (2001). Compositional Semantics With Lexicalized Tree-Adjoining Grammar (LTAG): How Much Underspecification is Necessary? In H. Bunt, R. Muskens, & E. Thijsse (Éd.), *Computing Meaning* (p. 147-163). Springer Netherlands. https://doi.org/10.1007/978-94-010-0572-2_9

Jousse, A.-L. (2003). *Normalisation des fonctions lexicales*. Pari 7.

Jousse, A.-L. (2010). *Modèle de structuration des relations lexicales fondé sur le formalisme des fonctions lexicales*. Paris 7.

Kahane, S. (2003). The Meaning Text Theory. In V. Agel, L. Eichinger, H.-W. Eroms, P. Hellwig, H. J. Heringer, & H. Lobin (Éd.), *Dependency and Valency. An International Handbook of Contemporary Research* (Vol. 1, p. 546-570). Berlin - New York: W. de Gruyter.

Kahane, S., & Polguère, A. (2001). Formal foundation of lexical functions. In *Proceedings of ACL/EACL 2001 Workshop on Collocation* (p. 8–15).

Kasper, R. (1989). *SPL: A sentence plan language for text generation* (Technical report). Information Science Institute, University of Claifornia.

Kim, R., Dalrymple, M., Kaplan, R., Holloway King, T., Masuichi, H., & Ohkuma, T. (2003). Multilingual Grammar Development via Grammar Porting. In *Proceedings of the ESSLLI Workshop on Ideas and Strategies for Multilingual Grammar Development*. Vienna, Austria.

Kittredge, R., Iordanskaja, L., & Polguère, A. (1988). Multi-Lingual Text Generationand the Meaning-Text Theory. In *Conference on Theoretical and Methodological Issues in Machine Translation of Natur al Languages*. Pittsburg, PA, USA.

Kjellmer, G. (1994). *Dictionary of English Collocations: Based on the Brown Corpus*. Oxford: Claredon Press.

Lambrey, F., & Lareau, F. (2015). Le traitement des collocations en génération de texte multilingue. In *Actes de TALN 2015* (p. 579-585). Caen.

Lareau, F., Dras, M., Börschinger, B., & Dale, R. (2011). Collocations in Multilingual Natural Language Generation: Lexical Functions meet Lexical Functional Grammar. In *Proceedings of ALTA'11* (p. 95-104). Canberra.

Lareau, F., Dras, M., Börschinger, B., & Turpin, M. (2012). Implementing lexical functions in XLE. In *Proceedings of LFG12* (p. 362-382). Denpasar, Indonesia.

Lareau, F., & Wanner, L. (2007). Towards a Generic Multilingual Dependency Grammar for Text Generation. In *Proceedings of the GEAF07 Workshop* (p. 203-223). Stanford: CSLI.

Lavoie, B., & Rambow, O. (1997). A Fast and Portable Realizer for Text Generation Systems. In *Proceedings of the Fifth Conference on Applied Natural Language Processing* (p. 265–268). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.3115/974557.974596

Lavoie, B., Rambow, O., & Reiter, E. (1996). The modelexplainer. In *Proceedings of the 8th international workshop on natural language generation* (p. 9–12). Hermonceux Castle, Sussex, UK.

Lee, W., & Evens, M. (1996). Generating Cohesive Text Using Lexical Functions. In L. Wanner (Éd.), *Lexical Functions in Lexicography and Natural Language Processing* (p. 299-306). Amsterdam/Philadelphia: John Benjamins Publishing.

L'homme, M.-C. (2002). Fonctions lexicales pour représenter les relations sémantiques entre termes. *TAL. Traitement automatique des langues*, *43*(1), 19–41.

L'Homme, M.-C. (2005). Conception d'un dictionnaire fondamental de l'informatique et de l'Internet: sélection des entrées. *Le langage et l'homme*, *40*(1), 137–154.

L'Homme, M.-C. (2009). DiCoInfo : le dictionnaire fondamental de l'informatique et de l'internet. Consulté à l'adresse http://olst.ling.umontreal.ca/cgi-bin/dicoinfo/search.cgi

L'Homme, M.-C., & Lanneville, M.-È. (s. d.). DiCoEnviro : les dictionnaire fondamental de l'environnement. Consulté à l'adresse http://olst.ling.umontreal.ca/cgi-bin/dicoenviro/search.cgi

Li, H., Zhu, Y., & Jin, Y. (2015). Identifying Verb-Preposition Multi-Category Words in Chinese-English Patent Machine Translation. In S. K. Chalup, A. D. Blair, & M. Randall (Éd.), *Artificial Life and Computational Intelligence: First Australasian Conference, ACALCI 2015, Newcastle, NSW, Australia, February 5-7, 2015. Proceedings* (p. 409-421). Cham: Springer International Publishing.

Mann, W. C., Matthiessen, C. M. I. M., & UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION SCIENCES INST. (1982). *Two Discourse Generators. A Grammar and a Lexicon for a Text-Production System.* Ft. Belvoir: Defense Technical Information Center.

Manning, C. D., & Schütze, H. (1999). *Foundations of statistical natural language processing*. Cambridge, Mass.: MIT Press.

Marengo, S., & Robichaud, B. (2016). *Des connaissances à la ressource et de la ressource aux connaissances : genèse du DicoLiLex*. Présenté à 84e congrès de l'ACFAS, Montréal.

Masuichi, H., & Ohkuma, T. (2003). Constructing a practical Japanese parser based on Lexical-Functional Grammar. *Journal o fNatural Language Processing*, p. 19-109.

Matthiessen, C. M. I. M., & Bateman, J. A. (1991). *Text generation and systemic-functional linguistics: experiences from English and Japanese*. London: Pinter.

Maxwell, K., & Heylen, D. (1994). Lexical Functions and the Translation of Collocations. In *Euralex* (p. 298-305).

McKeown, K., & Radev, D. (2000). Collocations. In R. Dale (Éd.), *Handbook of natural language processing* (p. 507-523). New York: Marcel Dekker.

Mel′čuk, I. (1973). Linguistic Theory and « Meaning-Text » Type Models. In R. Bogdan & I. Niiniluoto (Éd.), *Logic, language and Probability* (p. 223-235). Dordrecht-Boston: Reidel.

Mel′čuk, I. (1996). Lexical Functions: A Tool for the Description of Lexical Relations in a Lexicon. In L. Wanner (Éd.), *Lexical Functions in Lexicography and Natural Language Processing* (p. 37-102). Benjamins.

Mel′čuk, I. (1997). *Vers une linguistique Sens-Texte. Leçon inaugurale.* Collège de France, Paris.

Mel′čuk, I. (1998). Collocations and lexical functions. *en Anthony Paul COWIE (ed.)(2001 [1998])*, 23–54.

Mel′čuk, I. (2003). Les collocations : définition, rôle et utilité. In F. Grossmann & A. Tutin (Éd.), *Les collocations : analyse et traitement*. Amsterdam: De Werelt.

Mel′čuk, I. (2004). Les verbes supports sans peine. *Lingvisticae Investigationes*, *27*(2), 203-217.

Mel′čuk, I. (2013). Tout ce que nous voulions savoir sur les phrasèmes, mais ... *Cahiers de lexicologie*, *102*, 129-150.

Mel′čuk, I. A. (1988). *Dependency syntax: theory and practice*. Albany: State University Press of New York.

Mel′čuk, I., Arbatchewsky-Jumarie, N., & Clas, A. P. (1999). *DEC: dictionnaire explicatif et combinatoire du français contemporain : recherches lexico-sématiques IV*. Montréal: Presses de l'Université de Montréal.

Mel′čuk, I., Clas, A., & Polguère, A. (1995). *Introduction à la lexicologie explicative et combinatoire*. `Bruxelles: Duculot.

Meunier, F. (1997). *Implantation du formalisme de génération GTAG*. Paris 7.

Ministère de l'Éducation Nationale. (1959). Le Français fondamental 1er Degré.

Molins, P. (2014). *JSrealB: Approche systématique pour la réalisation multilingue de textes*. INSA de Lyon.

Molins, P., & Lapalme, G. (2015). JSrealB : A bilingual text realizer for web programming. In *Proceedings of the 15th European Workshop on Natural Language Generation* (p. 109-111). Brighton, England: Association for Computational Linguistics.

Nérima, L., Seretan, V., & Wehrli, E. (2006). Le problème des collocations en TAL. *Nouveaux cahiers de linguistique française*, *27*, 95-115.

Polguère, A. (1998). Pour un modèle stratifié de la lexicalisation en génération de texte. *Traitement Automatique des Langues (TAL)*, *39*(2), 57–76.

Polguère, A. (2000). A « NATURAL » LEXICALIZATION MODELFOR LANGUAGE GENERATION. In *Proceedings of the Fourth Symposium on Natural Language Processing2000* (Vol. 10-12, p. 37-50). Chiangmai, Thailand,.

Polguère, A. (2007). Lexical Function Standardness. In L. Wanner (Éd.), *selected lexical and grammatical issues in the Meaning-Text Theory: In honour of Igor Mel'cuk* (p. 43-96). John Benjamins Publishing.

Rayner, M., Hockey, B. A., & Bouillon, P. (2006). *Putting linguistics into speech recognition: the Regulus grammar compiler*. Stanford, Calif: Center for the Study of Language and Information.

Reiter, E. (1995). NLG vs. templates. In *Proceedings of the 5th European Workshop on Natural language Generation*. Leiden.

Reiter, E., & Dale, R. (1997). Building applied natural language generation systems. *Natural Language Engineering*, *3*(1), 57-87.

Reiter, E., & Dale, R. (2000). *Building Natural Language Generation Systems*. New York, NY, USA: Cambridge University Press.

Roussarie, L. (2000). *Un modele theorique d'inference de structures semantiques et discursives dans le cadre de la generation automatique de textes*. Paris 7.

Sag, I. A., Baldwin, T., Bond, F., Copestake, A., & Flickinger, D. (2002). Multiword Expressions: A Pain in the Neck for NLP. In A. Gelbukh (Éd.), *Computational Linguistics and Intelligent Text Processing* (p. 1-15). Mexico city: Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-45715-1_1

Santaholma, M. (2008). Multilingual Grammar Resources in Multilingual Application Development. In *Proceedings of the Workshop on Grammar Engineering Across Frameworks* (p. 25–32). Stroudsburg, PA, USA: Association for Computational Linguistics.

Sinclair, J. (1991). *Corpus, concordance, collocation*. Oxford: Oxford University Press.

Sinclair, J. (1995). *Collins COBUILD English dictionary.* London: Harper Collins.

Sowa, J. F. (2000). *Knowledge representation: logical, philosophical, and computational foundations*. Pacific Grove: Brooks/Cole.

Stede, M. (1996). Lexical options in multilingual generation from a knowledge base. In G. Adorni & M. Zock (Éd.), *Trends in Natural Language Generation An Artificial Intelligence Perspective* (p. 222-237). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-60800-1_32

Steinlin, J. (2003). *Générer des collocations*. Paris 7, Paris.

Swartout, W. R., & Smoliar, S. W. (1987). On making expert systems more like experts. *EXSY Expert Systems*, *4*(3), 196-208.

Ters, F., Mayers, G., & Reichenbach, D. (1969). *L'échelle Dubois - Buyse d'orthographe usuelle française*. Neuchâtel : Messeiller.

Van Deemter, K., Krahmer, E., & Theune, M. (2005). Real Versus Template-Based Natural Language Generation: A False Opposition? *Computational Linguistics*, *31*(1), 15–24. https://doi.org/10.1162/0891201053630291

Van Der Wouden, T. (1992). Prolegomena to a Multilingual Description of Collocations. In H. Tommola & K. Varantola (Éd.), *Proceedings of EURALEX 1992* (p. 449-456). Tampere.

Vaudry, P.-L., & Lapalme, G. (2013). Adapting SimpleNLG for bilingual English - French realisation. In *Proceedings of the 14th European Workshop on Natural Language Generation* (p. 183-187). Sofia, Bulgaria: Association for Computational Linguistics.

Wanner, L. (Éd.). (1996). *Lexical Functions in Lexicography and Natural Language Processing*. Amsterdam/Philadelphia: John Benjamins Publishing.

Wanner, L. (Éd.). (2007). *Selected Lexical and Grammatical Issues in the Meaning–Text Theory. In honour of Igor Mel'cuk*. John Benjamins.

Wanner, L., Bohnet, B., Bouayad-Agha, N., Lareau, F., Lohmeyer, A., & Nicklaß, D. (2007). On the Challenge of Creating and Communicating Air Quality Information. In *Proceedings of ISESS 2007* (p. 356-367).

Wanner, L., Bohnet, B., Bouayad-Agha, N., Lareau, F., & Nicklaß, D. (2010). Marquis: Generation of User-Tailored Multilingual Air Quality Bulletins. *Applied Artificial Intelligence*, *24*(10), 914-952.

Wanner, L., & Lareau, F. (2009). Applying the Meaning-Text Theory Model to Text Synthesis with Low- and Middle-Density Languages in Mind. In S. Nirenburg (Éd.), *Language Engineering for Lesser-Studied Languages* (p. 207-242). IOS Press.

Wanner, L., Nicklaß, D., Bouayad-Agha, N., Bohnet, B., Bronder, J., Ferreira, F., … others. (2007). From Measurement Data to Environmental Information: MARQUIS-A Multimodal AiR QUality Information Service for the General Public. In *Poceedings of ISESS 2007*. Prague.

Weizenbaum, J. (1966). ELIZA—a Computer Program for the Study of Natural Language Communication Between Man and Machine. *Commun. ACM*, *9*(1), 36–45.

Žolkovskij, A., & Mel'čuk, I. (1967). O sisteme semantičeskogo sinteza. II. Pravila perefrazirovanija.[Sur le système de la synthèse sémantique (des textes) II: règles de paraphrasage]. *Naučno-texničeskaja informacija, Serija 2, Informacionnye processy i sistemy*, 17–27.

# Annexe. Recensement des règles de transduction

Cette annexe recense toutes les règles de transduction créées dans le cadre de ce mémoire. Il existe 10 patrons en tout, se répartissant sur 129 règles de transduction.

Pour chaque patron, nous fournissons ses règles de transduction affiliées les plus représentatives. Quand les fonctions lexicales sont attestées, nous fournissons également des exemples d'usage.

De même, une structure sémantique typique et la structure syntaxique profonde résultante sont également intégrées. Il s'agit d'une partie représentative des structures de test. La structure sémantique est reconnaissable par le label 'S' en haut de la figure. Elle contient également un arc étiqueté 'main' représentant le nœud communicativement dominant de la structure. Les structures syntaxiques sont constitués de nœuds étiquetés par des labels en majuscule, 'X_N', 'V', etc. La notation complète des nœuds a été présentée dans la section 4.2.2.1 (Chapitre 4).

Une version de démo de GÉCO sera disponible courant 2017. Elle contiendra toutes ces structures, ainsi que les dictionnaires *semanticon*, *lexicon* et *lf* requis pour les générer.

# I. Modification

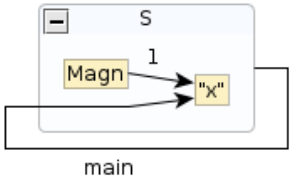
- Lex_modification

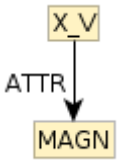
```
Sem<=>DSynt lex_modification : modification
leftside (∀)
l:?Xl{                                  // LF node
  sem=Magn|Ver|Bon|Pos                  // LF name
  l:1-> ?Yl {}                          // base
}
?L <- semanticon::(?Yl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Yr {                                // base
  <=> ?Yl
  lf::(Mod).(gp).(L)-> ?Xr{             // modifier
    <=> ?Xl
    dlex=?F.value
    lf=?F.name
    base=?L
    dpos=lf::(mod).dpos
    dsynt=OK
  }
}

conditions (∃)
?F.name=?Xl.sem;                        // matches the sem node with the name of LF
not lf::(?Yr.dlex).dpos;                // ?Yr is not a LF
?Yr.dsynt=OK;                           //not sure if always has to be the case.
//not ?F.merged=yes;                    // The collocate must not be merged with the base
```

Exemples :

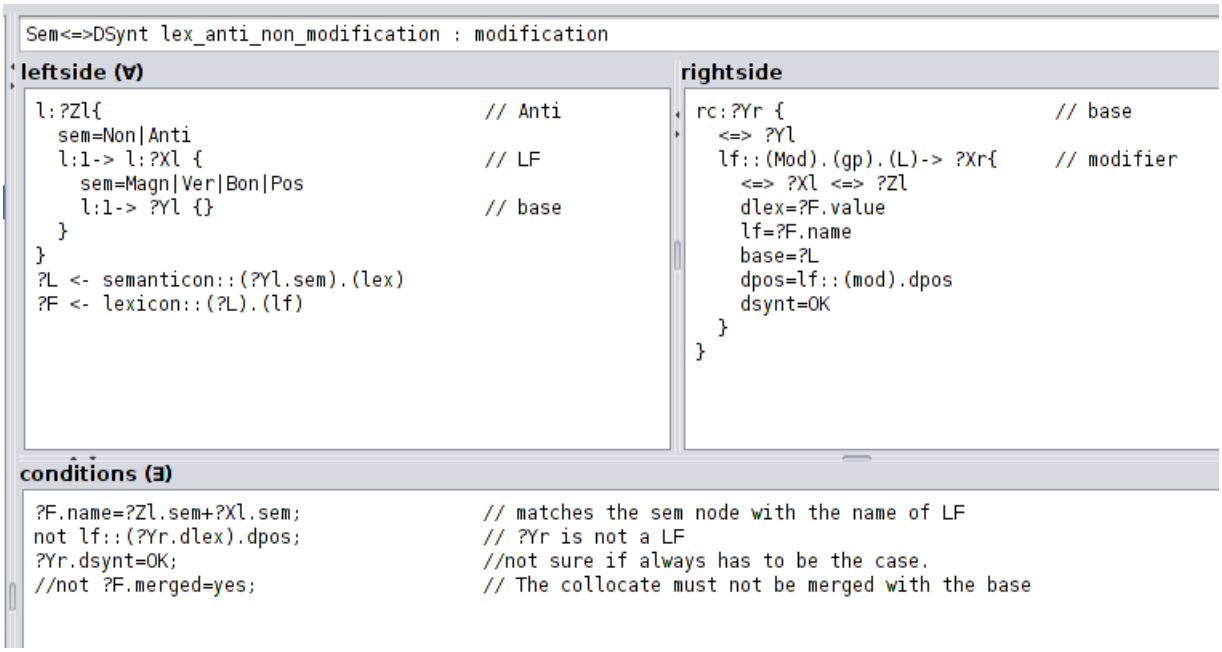
(164) Magn(peur) = bleue

(165) Ver (horloge) = exacte



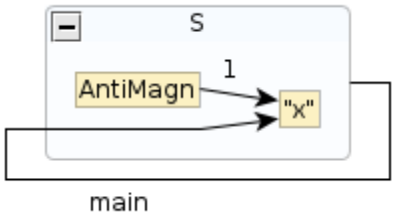
- Lex_anti_non_modification

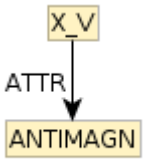



Exemples :

(166) AntiBon(choix) = mauvais

(167) AntiMagn(appétit) = léger



## II. Préposition

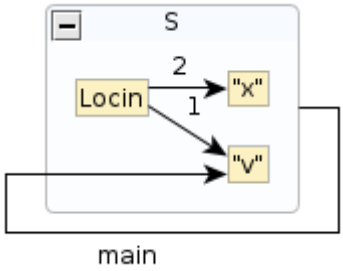
- Lex_preposition

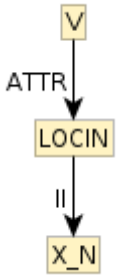
```
Sem<=>DSynt lex_preposition : preposition
leftside (∀)
?Zl{                                        // LF
  sem=Locin|Locab|Locad|Instr|Propt
  l:?l-> ?Xl {}                             // action
  l:?r-> ?Yl {}                             // base
}
?L <- semanticon::(?Yl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Xr {  <=> ?Xl                           // main node
   lf::(Loc).(gp).(X1)->
   ?Zr {
        <=> ?Zl                             // Locx|Propt|Instr
        dpos=lf::(Loc).(dpos)
        dlex=?F.value
        dsynt=OK
        lf=?F.name
        base=?L
        lf::(Loc).(gp).(L)-> ?Yr{ <=> ?Yl   // base
           dlex=?L
           dpos=N                           //must be a noun
           dsynt=OK
        }
     }
}

conditions (∃)
?F.name=?Zl.sem;                            // matches the sem node with the name of LF
?Xr.dpos=V;                                 // must be a verb
?Xr.dsynt=OK;                               // wait for X to be lexicalized
not ?F.merged=yes;                          // The collocate must not be merged with the base
```

Exemples :

(168) Locin(Ontario) = en Ontario

(169) Locab(balcon) = depuis le balcon



## III. Noms gouverneurs sémantiques

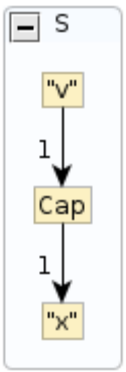
- Lex_ngov_sem

```
Sem<=>DSynt lex_ngov_sem : semantic_governor_noun
leftside (∀)
l:?Xl{                    // LF
  sem=Sing|Mult|Culm|Germ|Centr|Cap|Equip|Gener
  l:1-> ?Yl {}            // base
}
?L <- semanticon::(?Yl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc: ?Xr {                                  // LF
     <=> ?Xl
     dlex=?F.value
     lf=?F.name
     dpos=lf::(NGov).dpos
     dsynt=OK
     lf::(NGov).(gp).(L)-> ?Yr{            // Base
        <=> ?Yl
        dlex=?L
        dpos=lf::(NGov).dpos
        synt=OK
      }
}

conditions (∃)
?F.name=?Xl.sem;                    // matches the sem node with the name of LF
```

Exemples :

(170) Sing(chocolat) = carré de chocolat

(171) Mult(loup) = meute de loup

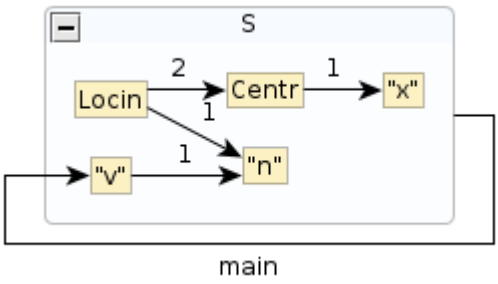


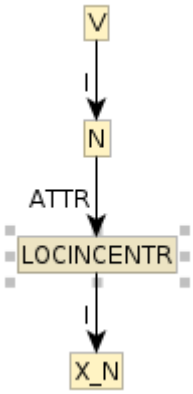

- Lex_loc_ngov_sem

```
Sem<=>DSynt lex_loc_ngov_sem : semantic_governor_noun
leftside (∀)
l:?Zl{                                          // Loc
  sem=Locin|Locab|Locad
  l:1->?Wl{}
  l:2->l:?Xl{                                   //NgovSem
    sem=Sing|Mult|Culm|Germ|Centr|Cap|Equip
    l:1-> ?Yl {}                                // NGovSem base
}
}
?L <- semanticon::(?Yl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc: ?Wr {  <=> ?Wl                              // main node
   lf::(?F.name).(gp).(X1)->
     ?Xr {  <=> ?Xl <=>?Zl                      // Locx_NGovSem
     dlex=?F.value
     lf=?F.name
     dpos=lf::(NGov).dpos
     dsynt=OK
     lf::(?F.name).(gp).(L)-> ?Yr{              // Base
        <=> ?Yl
        dlex=?L
        dpos=lf::(?F.name).dpos
        synt=OK
      }
}
}

conditions (∃)
?F.name=?Zl.sem+?Xl.sem;                // matches the sem node with the name of LF
?Wr.dsynt=OK;                           // wait for Wr to be lexicalized
```

Exemples :

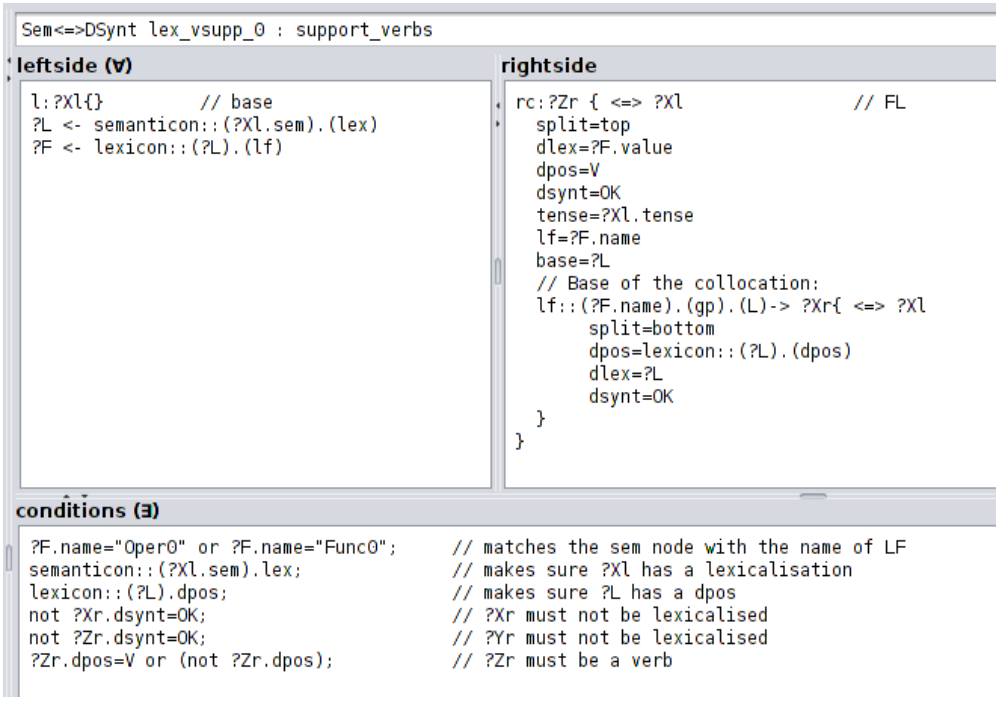
(172) LocinCentr(forêt) = au cœur de la forêt



## IV. Verbes supports

- Lex_vsupp_0




Exemple :

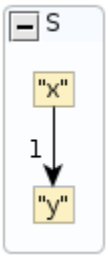
(173) Func0(pluie) = tomber

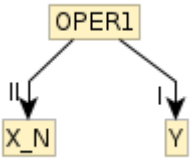
- Lex_vsupp_i

```
Sem<=>DSynt lex_vsupp_i : support_verbs
leftside (∀)
l:?Xl{                    // base
  l:?r-> ?Yl {}           // rth actant
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Zr { <=> ?Xl                                  // FL
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  tense=?Xl.tense
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl           // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
  lf::(?F.name).(gp).(?r)-> ?Yr{ <=> ?Yl          // rth actant to be lexicalised
  }
}

conditions (∃)
?F.name="Oper"+?r or ?F.name="Func"+?r or ?F.name="Labor"+?r; // matches the sem node with the name of LF
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(174) Oper1(question) = poser une question



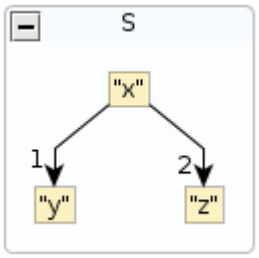



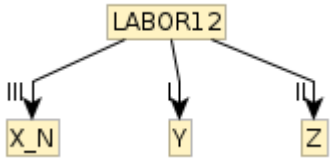
- Lex_vsupp_ij

```
Sem<=>DSynt lex_vsupp_ij : support_verbs
leftside (∀)
l:?Xl{                          // base
  l:?r-> ?Yl {}                 // rth actant
  l:?l-> ?Wl {}                 // lth actant
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Zr { <=> ?Xl                                // FL
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  tense=?Xl.tense
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
  lf::(?F.name).(gp).(?r)-> ?Yr{ <=> ?Yl        // rth actant
  }
  lf::(?F.name).(gp).(?l)-> ?Wr{ <=> ?Wl        // lth actant
  }
}

conditions (∃)
?F.name="Oper"+?r+?l or ?F.name="Func"+?r+?l or ?F.name="Labor"+?r+?l;
                                  // matches the sem node with the name of LF
semanticon::(?Xl.sem).lex;        // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;               // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                 // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                 // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);     // ?Zr must be a verb
```

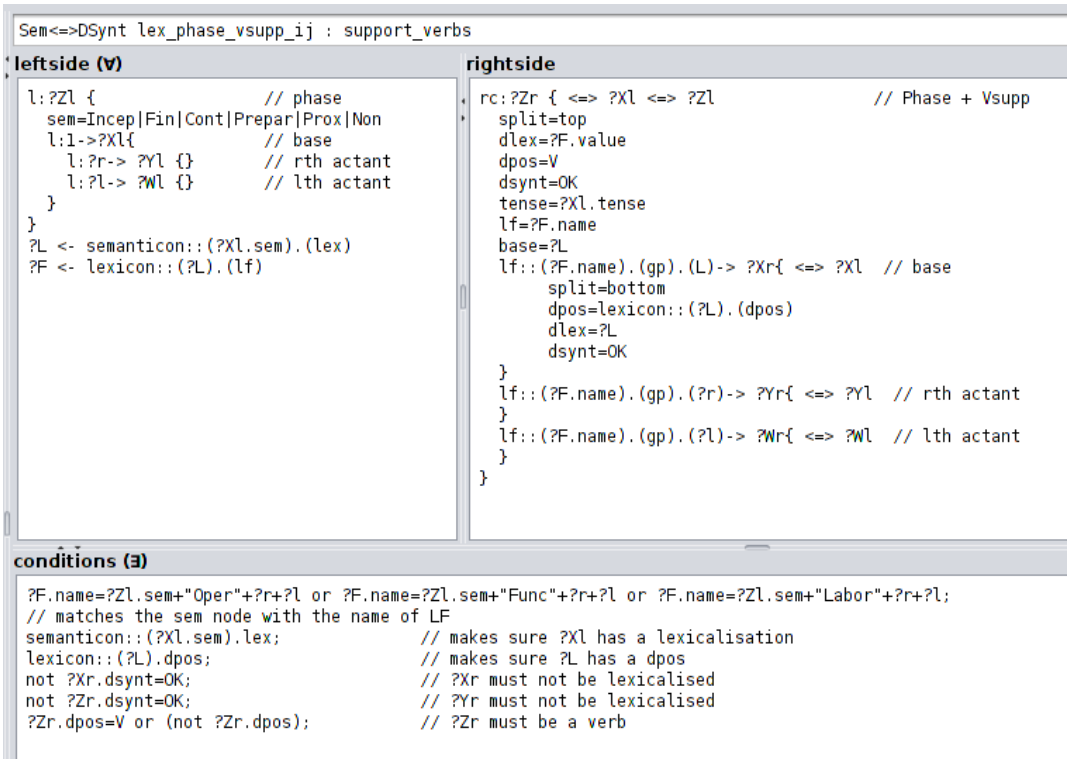
Exemple :

(175) Labor12(aversion) = avoir



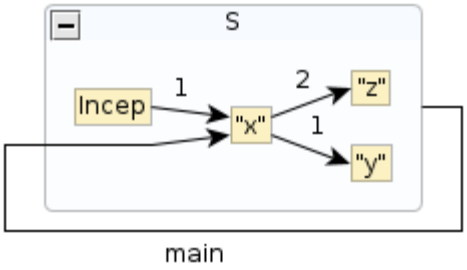
- Lex_phase_vsupp_ij

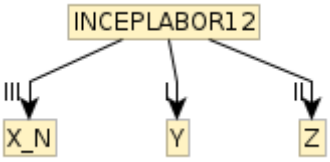


```
Sem<=>DSynt lex_phase_vsupp_ij : support_verbs

leftside (∀)
l:?Zl {                  // phase
  sem=Incep|Fin|Cont|Prepar|Prox|Non
  l:1->?Xl{              // base
    l:?r-> ?Yl {}        // rth actant
    l:?l-> ?Wl {}        // lth actant
  }
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Zr { <=> ?Xl <=> ?Zl                // Phase + Vsupp
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  tense=?Xl.tense
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl  // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
  lf::(?F.name).(gp).(?r)-> ?Yr{ <=> ?Yl  // rth actant
  }
  lf::(?F.name).(gp).(?l)-> ?Wr{ <=> ?Wl  // lth actant
  }
}

conditions (∃)
?F.name=?Zl.sem+"Oper"+?r+?l or ?F.name=?Zl.sem+"Func"+?r+?l or ?F.name=?Zl.sem+"Labor"+?r+?l;
// matches the sem node with the name of LF
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(176) IncepLabor12(affection) = prendre en affection



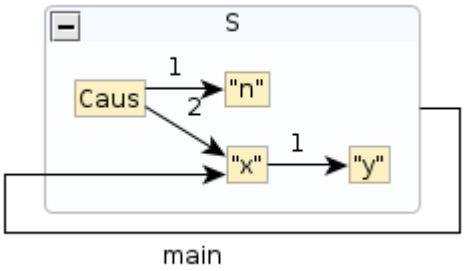



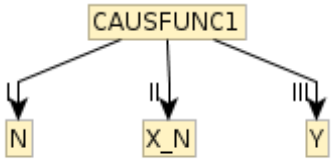
- Lex_causExt_vsupp_i

```
Sem<=>DSynt lex_causExt_vsupp_i : support_verbs
leftside (∀)
l:?Zl {                  // Caus FL
  sem=Caus|Liqu|Perm
  l:2->l:?Xl{            // base
    l:?r-> ?Wl {}        // rth ctant
   }
  l:1->?Yl{}             // source
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Zr { <=> ?Xl <=> ?Zl        // Caus FL + Vsupp
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(X1)-> ?Yr{ <=> ?Yl        // source
  }
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
  lf::(?F.name).(gp).(?r)-> ?Wr{ <=> ?Wl        // rth actant
  }
}
conditions (∃)
?F.name=?Zl.sem+"Oper"+?r or ?F.name=?Zl.sem+"Func"+?r or ?F.name=?Zl.sem+"Labor"+?r;
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(177) CausFunc1(frisson) = donner



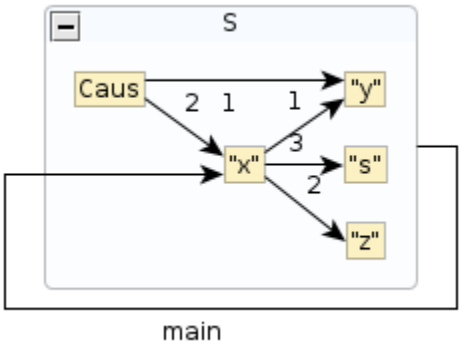



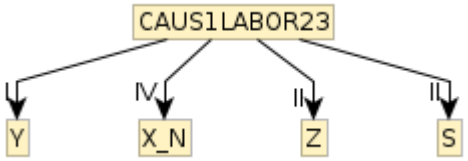
- Lex_causInt_vsupp_ij

```
Sem<=>DSynt lex_causInt_vsupp_ij : support_verbs
leftside (∀)
l:?Zl {                 // FL Caus
  sem=Caus|Liqu|Perm
  l:1->?Yl{}            // source
  l:2->l:?Xl{           // base
    l:?r->?Yl{}         // source
    l:?l-> ?Wl {}               // lth actant
    l:?s-> ?Ul {}               // sth actant
  }
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Zr { <=> ?Xl <=> ?Zl        // FL Caus + Vsupp
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(X1)-> ?Yr{ <=> ?Yl        // source
  }
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L // won't work if ?Xr should be a LF
       dsynt=OK
  }
  lf::(?F.name).(gp).(?l)-> ?Wr{ <=> ?Wl        // lth actant
  }
  lf::(?F.name).(gp).(?l)-> ?Ur{ <=> ?Ul        // sth actant
  }
}
conditions (∃)
?F.name=?Zl.sem+?r+"Oper"+?l+?s or ?F.name=?Zl.sem+?r+"Func"+?l+?s or ?F.name=?Zl.sem+?r+"Labor"+?l+?s;
lexicon::(?L).dpos;             // makes sure ?L has a dpos
semanticon::(?Xl.sem).lex;      // makes sure ?Xl has a lexicalisation
not ?Xr.dsynt=OK;               // must not be lexicalized
not ?Zr.dsynt=OK;               // must not be lexicalized
?Zr.dpos=V or (not ?Zr.dpos);
```

Exemple :

(178) Caus1Labor32(ordonnance) = prescrire



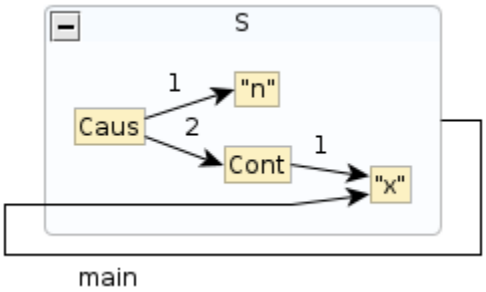



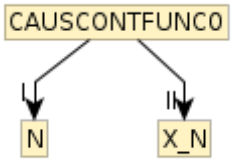
- Lex_causExt_phase_vsupp_0

```
Sem<=>DSynt lex_causExt_phase_vsupp_0 : support_verbs
leftside (∀)
l:?Cl{                                  // FL Caus
  sem=Caus|Liqu|Perm
  l:2-> l:?Zl {}                        // FL phase
  l:1->?Yl {}                           // source
}
l:?Zl {                                 //FL phase
  sem=Incep|Fin|Cont|Prepar|Prox
  l:1-> l:?Xl {}                        //base
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Zr { <=> ?Xl   <=> ?Zl  <=> ?Cl     // FL Caus + Phase + Vsupp
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(X1)-> ?Yr{ <=> ?Yl        // source
  }
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
}

conditions (∃)
?F.name=?Cl.sem+?Zl.sem+"Oper0" or ?F.name=?Cl.sem+?Zl.sem+"Func0";
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

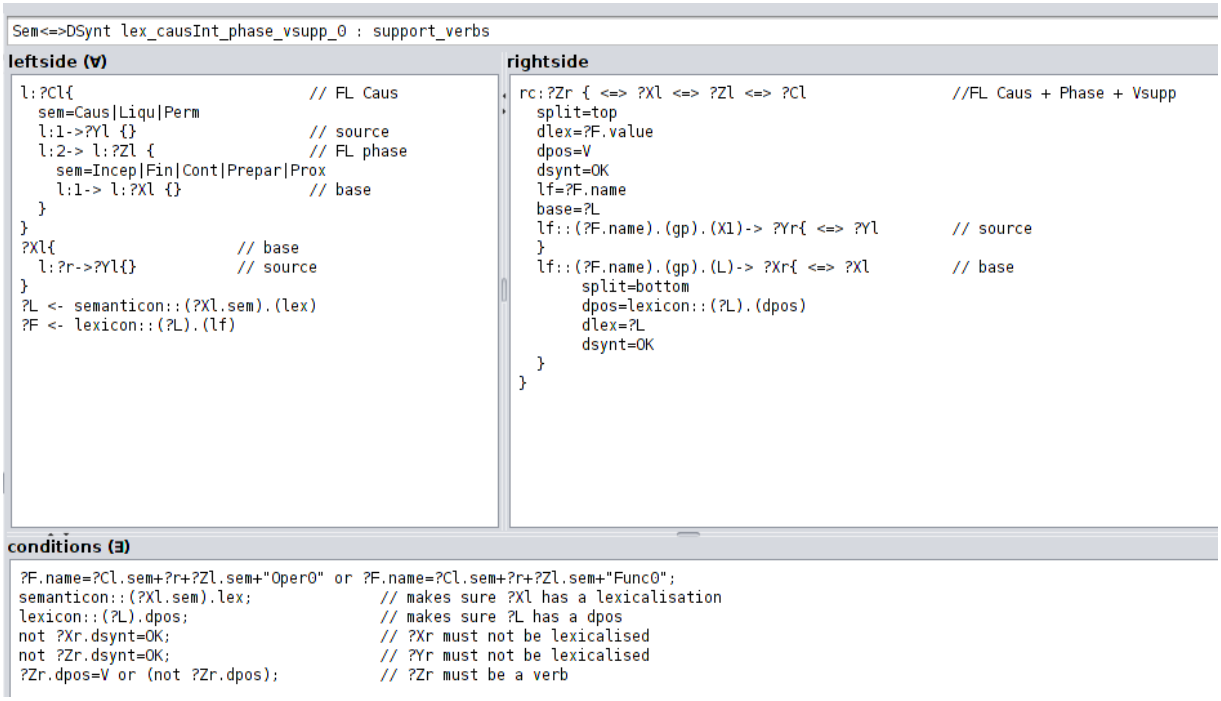
Exemple :

(179) CausContFunc0(défiance) = nourrir



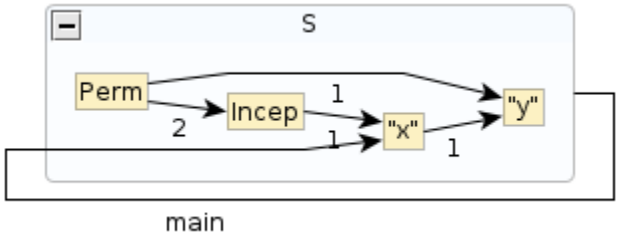
- Lex_causInt_ phase_vsupp_0

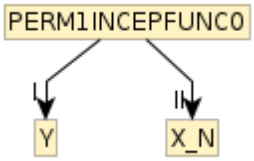


```
Sem<=>DSynt lex_causInt_phase_vsupp_0 : support_verbs

leftside (∀)
l:?Cl{                          // FL Caus
  sem=Caus|Liqu|Perm
  l:1->?Yl {}                   // source
  l:2-> l:?Zl {                 // FL phase
    sem=Incep|Fin|Cont|Prepar|Prox
    l:1-> l:?Xl {}              // base
  }
}
?Xl{                    // base
  l:?r->?Yl{}           // source
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Zr { <=> ?Xl <=> ?Zl <=> ?Cl                //FL Caus + Phase + Vsupp
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(X1)-> ?Yr{ <=> ?Yl        // source
  }
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
}

conditions (∃)
?F.name=?Cl.sem+?r+?Zl.sem+"Oper0" or ?F.name=?Cl.sem+?r+?Zl.sem+"Func0";
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(180) Caus1ContFunc0(déposition) = maintenir



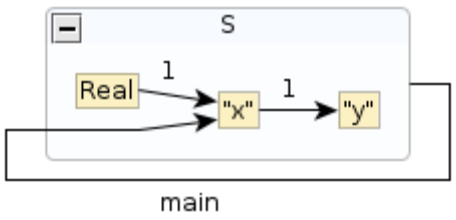
## V. Verbes de réalisation

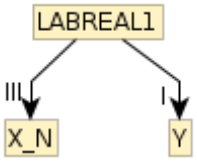
- Lex_vreal_i

```
Sem<=>DSynt lex_vreal_i : realisation_verbs
leftside (∀)
l:?Wl{                                  // FL Real
  sem=Real|AntiReal
  l:1-> l:?Xl{                          // base
      l:?r-> ?Yl {}                     // rth actant
  }
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Zr { <=> ?Xl <=> ?Wl                          // Vreal
  split=top
  dlex=?F.value
  dpos=V
  lf=?F.name
  base=?L
  dsynt=OK
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl           // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
  lf::(?F.name).(gp).(?r)-> ?Yr{ <=> ?Yl          // rth actant
  }
}

conditions (∃)
?F.name=?Wl.sem+?r  or
(?F.name="Labreal"+?r and ?Wl.sem=lf::(?F.name).(sem)) or
(?F.name="AntiLabreal"+?r and lf::(?F.name).(sem)=?Wl.sem) or
(?F.name="Fact"+?r and lf::(?F.name).(sem)=?Wl.sem) or
(?F.name="AntiFact"+?r and lf::(?F.name).(sem)=?Wl.sem);
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(181) Real1(couteau) = couper



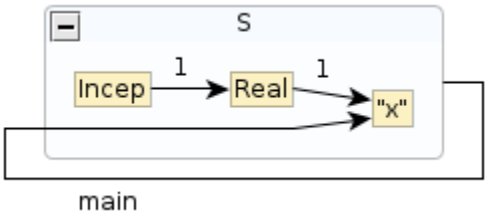



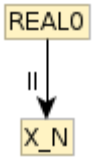
- Lex_phase_vreal_0

```
Sem<=>DSynt lex_phase_vreal_0 : realisation_verbs
leftside (∀)
l:?Cl{                          // phase
  sem=Incep|Fin|Cont|Prepar|Prox
  l:1-> l:?Zl{}
}
?Zl{
   sem=Real|AntiReal
   l:?1->l:?Xl{}                // base
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Zr { <=> ?Xl  <=> ?Zl <=> ?Cl              // Phase + Real
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  tense=?Xl.tense
  voice=ACT
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl        // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
}

conditions (∃)
?F.name=?Cl.sem+?Zl.sem+"0" or
(?F.name=?Cl.sem+"AntiFact0" and lf::(?F.name).(sem)=?Cl.sem+?Zl.sem) or
(?F.name=?Cl.sem+"Fact0" and lf::(?F.name).(sem)=?Cl.sem+?Zl.sem);
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(182) IncepFact0(clinique) = ouvrir



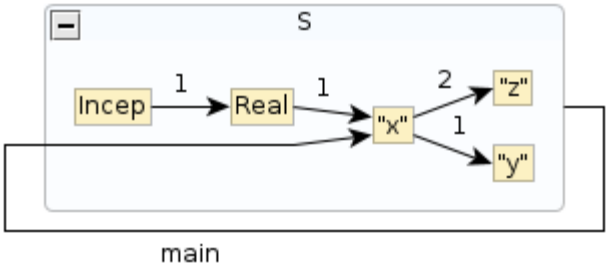



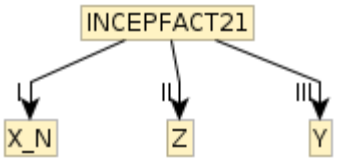
- Lex_phase_vreal_ij

```
Sem<=>DSynt lex_phase_vreal_ij : realisation_verbs

leftside (∀)
l:?Cl{                              // phase
  sem=Incep|Fin|Cont|Prepar|Prox
  l:1-> l:?Zl {}
}
?Zl{
  sem=Real|AntiReal                 // FL
  l:?1->l:?Xl{}
}
?Xl {                               // base
      l:?r->?Yl {}                  // rth actant
      l:?l->?Wl {}                  // lth actant
   }
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Zr { <=> ?Xl <=> ?Zl <=> ?Cl                // Phase + Vreal
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  tense=?Xl.tense
  voice=ACT
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
  lf::(?F.name).(gp).(?r)-> ?Yr{ <=> ?Yl        // rth actant
  }
  lf::(?F.name).(gp).(?l)-> ?Wr{ <=> ?Wl        // lth actant
  }
}

conditions (∃)
?F.name=?Cl.sem+?Zl.sem+?r+?l or
(?F.name=?Cl.sem+"AntiFact"+?r+?l and lf::(?F.name).(sem)=?Cl.sem+?Zl.sem) or
(?F.name=?Cl.sem+"Fact"+?r+?l and lf::(?F.name).(sem)=?Cl.sem+?Zl.sem) or
(?F.name=?Cl.sem+"Labreal"+?r+?l and lf::(?F.name).(sem)=?Cl.sem+?Zl.sem) or
(?F.name=?Cl.sem+"AntiLabreal"+?r+?l and lf::(?F.name).(sem)=?Cl.sem+?Zl.sem);
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(183) IncepLabreal12(frigo) = mettre



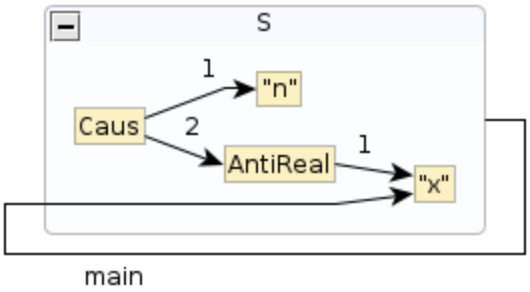



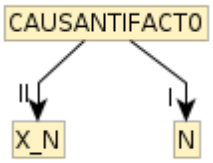
- Lex_causExt_vreal_0

```
Sem<=>DSynt lex_causExt_vreal_0 : realisation_verbs

leftside (∀)
l:?Cl{                          // cause
  sem=Caus|Liqu|Perm
  l:2-> l:?Zl{}
  l:1-> ?Ul{}                   // source
}
?Zl{
   sem=Real|AntiReal
   l:?1->l:?Xl{}                // base
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Zr { <=> ?Xl  <=> ?Zl <=> ?Cl              // cause + Real
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  tense=?Xl.tense
  voice=ACT
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl        // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
  lf::(?F.name).(gp).(X1)-> ?Ur{ <=> ?Ul       // source
  }
}

conditions (∃)
?F.name=?Cl.sem+?Zl.sem+"0" or
(?F.name=?Cl.sem+"AntiFact0" and lf::(?F.name).(sem)=?Cl.sem+?Zl.sem) or
(?F.name=?Cl.sem+"Fact0" and lf::(?F.name).(sem)=?Cl.sem+?Zl.sem);
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(184) CausAntiFact0(lampe) = éteindre



- Lex_causInt_vreal_i

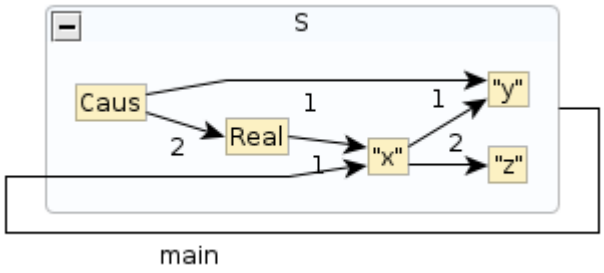
```
Sem<=>DSynt lex_causInt_vreal_i : realisation_verbs
leftside (∀)
l:?Cl{                                  // cause
  sem=Caus|Liqu|Perm
  l:1-> ?Yl{}                           // source
  l:2-> l:?Zl {}
}
?Zl{
  sem=Real|AntiReal
  l:1->l:?Xl{
      l:?r->?Yl {}              // source =  rth actant
      l:?l->?Wl {}                      // lth actant
    }
  }

?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
```

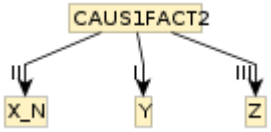
```
rightside
rc:?Zr { <=> ?Xl <=> ?Zl <=> ?Cl                // cause + Vreal
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
  lf::(?F.name).(gp).(Xl)-> ?Yr{ <=> ?Yl        // source = rth actant
  }
  lf::(?F.name).(gp).(?l)-> ?Wr{ <=> ?Wl        // lth actant
  }
}
```

```
conditions (∃)
?F.name=?Cl.sem+?r+?Zl.sem+?l or
(?F.name=?Cl.sem+?r+"AntiFact"+?l and lf::(?F.name).(sem)=?Cl.sem+?Zl.sem) or
(?F.name=?Cl.sem+?r+"Fact"+?l and lf::(?F.name).(sem)=?Cl.sem+?Zl.sem) or
(?F.name=?Cl.sem+?r+"Labreal"+?l and lf::(?F.name).(sem)=?Cl.sem+?Zl.sem) or
(?F.name=?Cl.sem+?r+"AntiLabreal"+?l and lf::(?F.name).(sem)=?Cl.sem+?Zl.sem);
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(185) Caus1Fact2(curiostié) = braquer



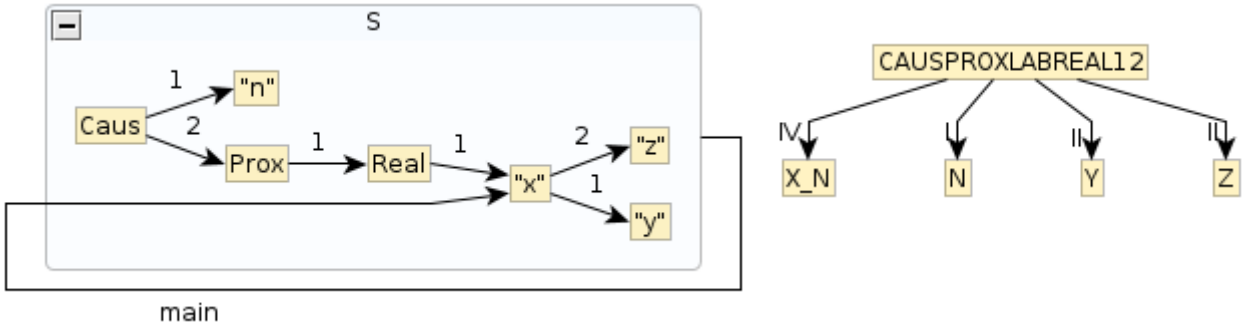



- Lex_causExt_ phase_vreal_ij

```
Sem<=>DSynt lex_causExt_phase_vreal_ij : realisation_verbs
leftside (∀)
l:?Al{                          // caus
  sem=Liqu|Caus|Perm
  l:2-> l:?Cl {}
  l:1-> ?Ul {}                  // source
}
?Cl{                            //phase
  sem=Incep|Fin|Cont|Prepar|Prox
  l:1-> l:?Zl{}
}
?Zl{
    sem=Real|AntiReal
    l:1->?Xl{}                  // base
  }
?Xl {                           //base
      l:?r->?Yl {}              // rth actant
      l:?l->?Wl {}              // rth actant
    }
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Zr { <=> ?Xl <=> ?Zl <=> ?Cl <=> ?Al // Caus + Phase + Vreal
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  tense=?Xl.tense
  lf=?F.name
  base=?L
  dsynt=OK
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
  lf::(?F.name).(gp).(X1)-> ?Ur{ <=> ?Ul        // source
  }
  lf::(?F.name).(gp).(?r)-> ?Yr{ <=> ?Yl        // rth actant
  }
  lf::(?F.name).(gp).(?r)-> ?Wr{ <=> ?Wl        // lth actant
  }
}

conditions (∃)
?F.name=?Al.sem+?Cl.sem+?Zl.sem+?r+?l or
(?F.name=?Al.sem+?Cl.sem+"AntiFact"+?r+?l and lf::(?F.name).(sem)=?Al.sem+?Cl.sem+?Zl.sem) or
(?F.name=?Al.sem+?Cl.sem+"Fact"+?r+?l and lf::(?F.name).(sem)=?Al.sem+?Cl.sem+?Zl.sem) or
(?F.name=?Al.sem+?Cl.sem+"Labreal"+?r+?l and lf::(?F.name).(sem)=?Al.sem+?Cl.sem+?Zl.sem) or
(?F.name=?Al.sem+?Cl.sem+"AntiLabreal"+?r+?l and lf::(?F.name).(sem)=?Al.sem+?Cl.sem+?Zl.sem);
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(186) CausProxLabreal12

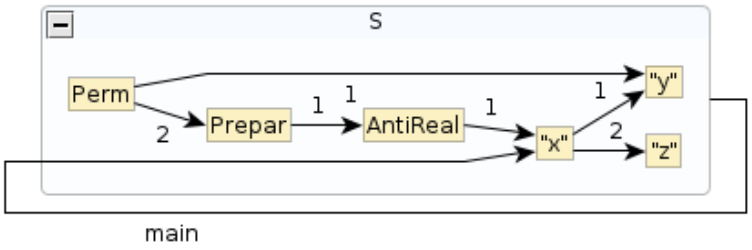


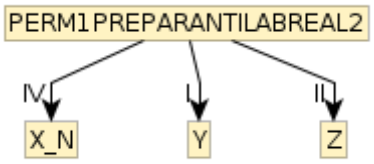
- Lex_causInt_ phase_vreal_i

```
Sem<=>DSynt lex_causInt_phase_vreal_i : realisation_verbs
leftside (∀)
l:?Al{                              // Caus
  sem=Liqu|Caus|Perm
  l:1-> ?Ul {}                      // source
  l:2-> l:?Cl {}
}
?Cl{                                // phase
  sem=Incep|Fin|Cont|Prepar|Prox
  l:1-> l:?Zl{}
}
?Zl{                                // real
    sem=Real|AntiReal
    l:1->l:?Xl{}                    // base
  }
?Xl{
  l:?r-> ?Ul{}          // rth actant = source
  l:?l-> ?Yl{}          // lth actant
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Zr { <=> ?Xl <=> ?Zl <=> ?Cl <=> ?Al // Caus + Phase + Vreal
  split=top
  dlex=?F.value
  dpos=V
  lf=?F.name
  base=?L
  dsynt=OK
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
  lf::(?F.name).(gp).(X1)-> ?Ur{ <=> ?Ul        // source = rth actant
  }
  lf::(?F.name).(gp).(?l)-> ?Yr{ <=> ?Yl        // lth actant
  }
}
conditions (∃)
?F.name=?Al.sem+?r+?Cl.sem+?Zl.sem+?l or
(?F.name=?Al.sem+?r+?Cl.sem+"AntiFact"+?l and lf::(?F.name).(sem)=?Al.sem+?Cl.sem+?Zl.sem) or
(?F.name=?Al.sem+?r+?Cl.sem+"Fact"+?l and lf::(?F.name).(sem)=?Al.sem+?Cl.sem+?Zl.sem) or
(?F.name=?Al.sem+?r+?Cl.sem+"Labreal"+?l and lf::(?F.name).(sem)=?Al.sem+?Cl.sem+?Zl.sem) or
(?F.name=?Al.sem+?r+?Cl.sem+"AntiLabreal"+?l and lf::(?F.name).(sem)=?Al.sem+?Cl.sem+?Zl.sem);
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(187) Perm1PreparAntiLabreal2



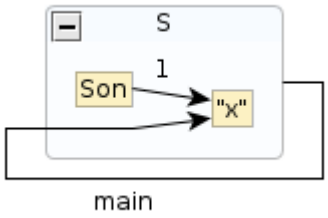
## VI. Autres verbes sémantiquesment pleins

- Lex_vsem_0

```
Sem<=>DSynt lex_vsem_0 : semantic_verbs
leftside (∀)
l:?Wl{                              // FL
  sem=Manif|Degrad|Excess|Stop|Obstr|Son|
  AntiManif|AntiDegrad|AntiExcess|AntiStop|AntiObstr|AntiSon
  l:1-> l:?Xl{}                     // base

}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Zr { <=> ?Wl <=> ?Xl        // VSem
  split=top
  dlex=?F.value
  dpos=V
  lf=?F.name
  base=?L
  dsynt=OK
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
}
conditions (∃)
?F.name=?Wl.sem;
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(188) Son(cloche) = sonner

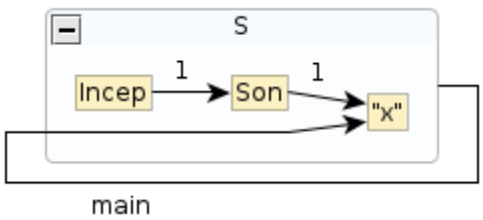


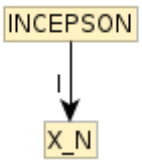

- Lex_phase_vsem_0

```
Sem<=>DSynt lex_phase_vsem_0 : semantic_verbs
leftside (∀)
l:?Cl{                              //phase
  sem=Incep|Fin|Cont|Prepar|Prox|Non
  l:1-> l:?Zl {}
}
?Zl{                                // FL
  sem=Manif|Degrad|Excess|Stop|Obstr|Son|
  AntiManif|AntiDegrad|AntiExcess|AntiStop|AntiObstr|AntiSon
  l:1->l:?Xl{}                      // base
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Zr { <=> ?Xl <=> ?Zl <=> ?Cl        // Phase + Vsem
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  tense=?Xl.tense
  voice=ACT
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
}
conditions (∃)
?F.name=?Cl.sem+?Zl.sem;
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(189) FinSon(cloche) = se taire



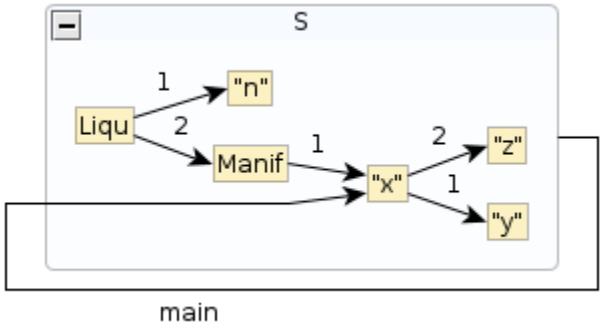



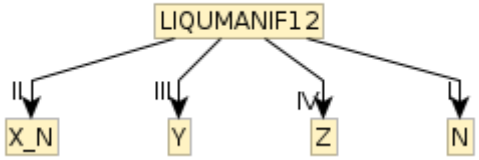
- Lex_causExt_vsem_ij

```
Sem<=>DSynt lex_causExt_vsem_ij : semantic_verbs
leftside (∀)
l:?Cl{                                  // caus
  sem=Caus|Liqu|Perm
  l:2-> l:?Zl {}
  l:1-> ?Ul{}                           // source
}
?Zl{                                    // FL
  sem=Manif|Degrad|Excess|Stop|Obstr|Son|Sympt|
  AntiManif|AntiDegrad|AntiExcess|AntiStop|AntiObstr|AntiSon|AntiSympt
  l:1->l:?Xl{}
}
?Xl {                           // base
      l:?r->?Yl {}              // rth actant
      l:?l->?Wl {}              // lth actant
    }
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Zr { <=> ?Xl <=> ?Zl <=> ?Cl                // cause + Vsem
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
  lf::(?F.name).(gp).(?r)-> ?Yr{ <=> ?Yl        // rth actant
  }
  lf::(?F.name).(gp).(?l)-> ?Wr{ <=> ?Wl        // lth actant
  }
  lf::(?F.name).(gp).(X1)-> ?Ur{ <=> ?Ul        // source
  }
}
conditions (∃)
?F.name=?Cl.sem+?Zl.sem+?r+?l;
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(190) LiquManif12



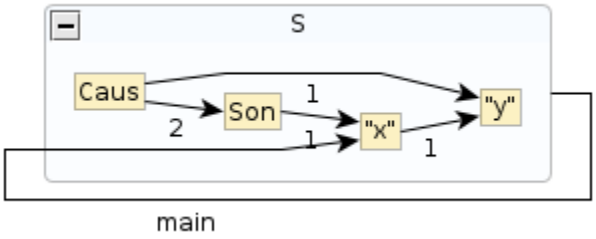



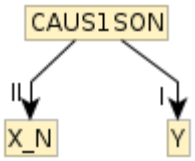
- Lex_causInt_vsem_0

```
Sem<=>DSynt lex_causInt_vsem_0 : semantic_verbs
leftside (∀)
l:?Cl{                              // cause
  sem=Caus|Liqu|Perm
  l:1-> ?Yl{}                       // source
  l:2-> l:?Zl{}
}
?Zl{
  sem=Manif|Degrad|Excess|Stop|Obstr|Son|
  AntiManif|AntiDegrad|AntiExcess|AntiStop|AntiObstr|AntiSon
   l:1->l:?Xl{                      // base
     l:?r->?Yl{}                    // source
  }
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Zr { <=> ?Xl  <=> ?Zl <=> ?Cl               // cause + Vsem
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
  lf::(?F.name).(gp).(X1)-> ?Yr{ <=> ?Yl        // source
  }
}
conditions (∃)
?F.name=?Cl.sem+?r+?Zl.sem;
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

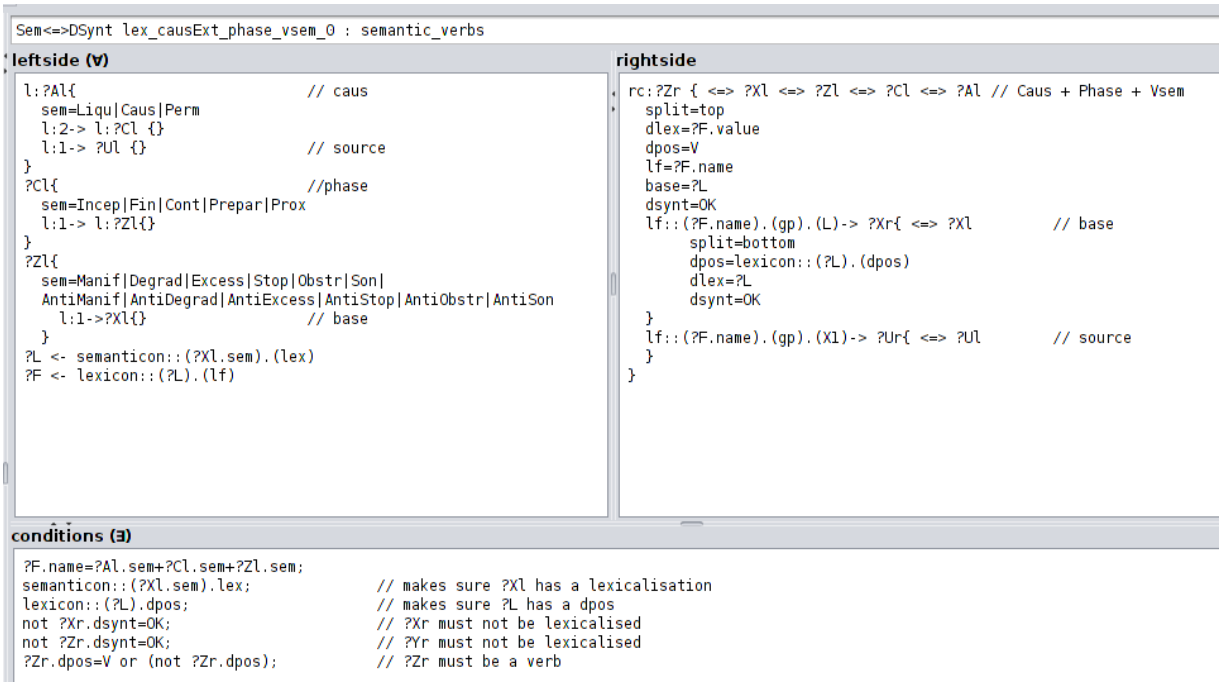
Exemple :

(191) Caus1Manif(désaccord) = exprimer



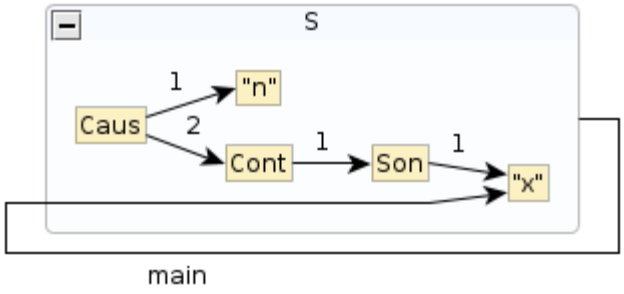
- Lex_causExt_phase_vsem_0

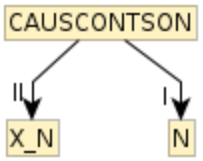


```
Sem<=>DSynt lex_causExt_phase_vsem_0 : semantic_verbs

leftside (∀)
l:?Al{                          // caus
  sem=Liqu|Caus|Perm
  l:2-> l:?Cl {}
  l:1-> ?Ul {}                  // source
}
?Cl{                            //phase
  sem=Incep|Fin|Cont|Prepar|Prox
  l:1-> l:?Zl{}
}
?Zl{
  sem=Manif|Degrad|Excess|Stop|Obstr|Son|
  AntiManif|AntiDegrad|AntiExcess|AntiStop|AntiObstr|AntiSon
    l:1->?Xl{}                  // base
  }
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Zr { <=> ?Xl <=> ?Zl <=> ?Cl <=> ?Al // Caus + Phase + Vsem
  split=top
  dlex=?F.value
  dpos=V
  lf=?F.name
  base=?L
  dsynt=OK
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
  lf::(?F.name).(gp).(X1)-> ?Ur{ <=> ?Ul        // source
  }
}

conditions (∃)
?F.name=?Al.sem+?Cl.sem+?Zl.sem;
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(192) CausContSon



- Lex_causInt_ phase_vsem_ij

Sem<=>DSynt lex_causInt_phase_vsem_ij : semantic_verbs

**leftside (∀)**

```
l:?Al{                          // Caus
  sem=Liqu|Caus|Perm
  l:1-> ?Ul {}                  // source
  l:2-> l:?Cl {}
}
?Cl{                            // phase
  sem=Incep|Fin|Cont|Prepar|Prox
  l:1-> l:?Zl{}
}
?Zl{                            // real
  sem=Manif|Degrad|Excess|Stop|Obstr|Son|Sympt|
  AntiManif|AntiDegrad|AntiExcess|AntiStop|AntiObstr|AntiSon|AntiSympt
    l:1->l:?Xl{}                // base
  }
?Xl{
  l:?r-> ?Ul{}          // rth actant = source
  l:?l-> ?Yl{}          // lth actant
  l:?s-> ?Wl{}          // sth actant
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
```

**rightside**

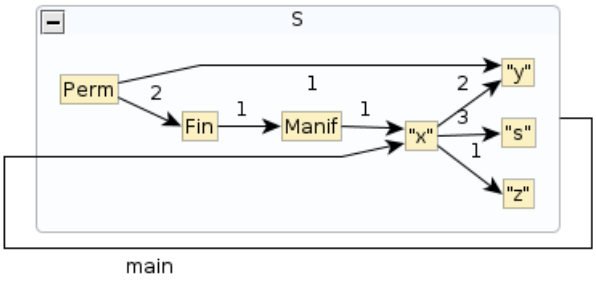
```
rc:?Zr { <=> ?Xl <=> ?Zl <=> ?Cl <=> ?Al // Caus + Phase + Vsem
  split=top
  dlex=?F.value
  dpos=V
  lf=?F.name
  base=?L
  dsynt=OK
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
  lf::(?F.name).(gp).(X1)-> ?Ur{ <=> ?Ul        // source = rth actant
  }
  lf::(?F.name).(gp).(?l)-> ?Yr{ <=> ?Yl        // lth actant
  }
  lf::(?F.name).(gp).(?s)-> ?Wr{ <=> ?Wl        // sth actant
  }
}
```

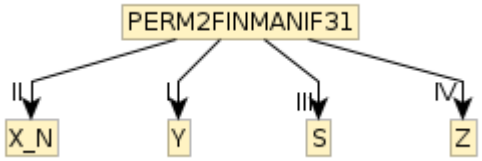
**conditions (∃)**

```
?F.name=?Al.sem+?r+?Cl.sem+?Zl.sem+?l+?s;
semanticon::(?Xl.sem).lex;              // makes sure ?Xl has a lexicalisation
lexicon::(?L).dpos;                     // makes sure ?L has a dpos
not ?Xr.dsynt=OK;                       // ?Xr must not be lexicalised
not ?Zr.dsynt=OK;                       // ?Yr must not be lexicalised
?Zr.dpos=V or (not ?Zr.dpos);           // ?Zr must be a verb
```

Exemple :

(193) Perm2FinManif31



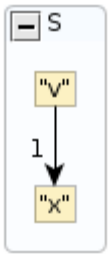
## VII. Figur

- Lex_figur

```
Sem<=>DSynt lex_Figur : no_patterns
leftside (∀)
l:?Xl{}                          // base
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Zr {        <=> ?Xl // LF
  split=top
  dlex=?F.value
  dpos=N
  dsynt=OK
  // Base of the collocation:
  lf::(Figur).(gp).(L)-> ?Xr{ <=> ?Xl
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L
       dsynt=OK
  }
}
conditions (∃)
?F.name="Figur";
```

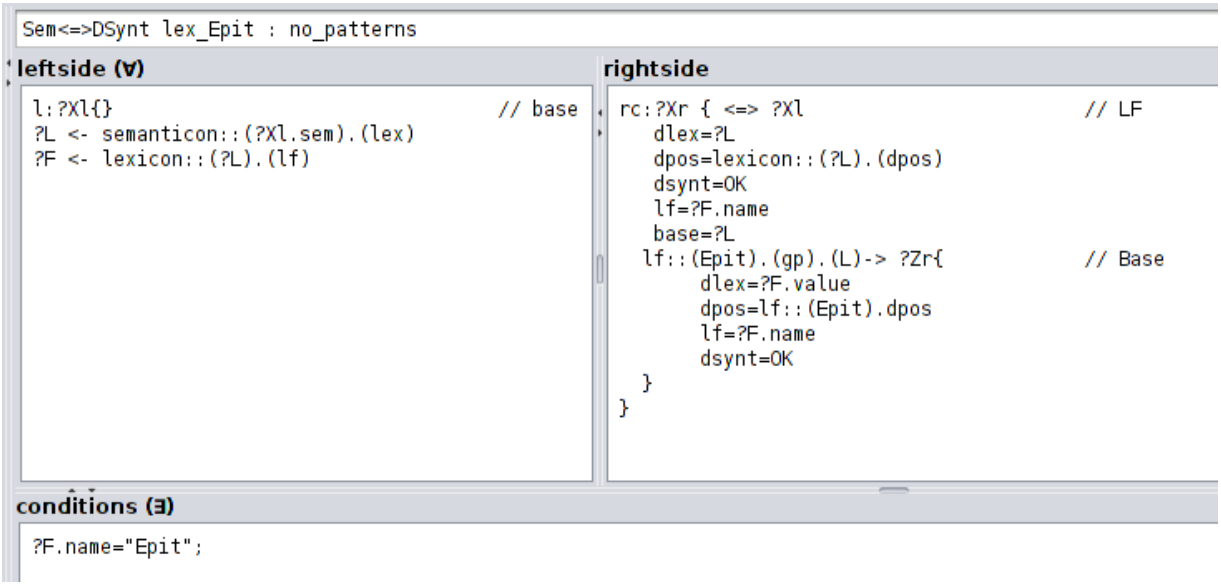
Exemple

(194) Figur(fumée) = écran de fumée




# VIII. Épit

- Lex_Epit




Exemple :

(195) Epit(océan) = immense

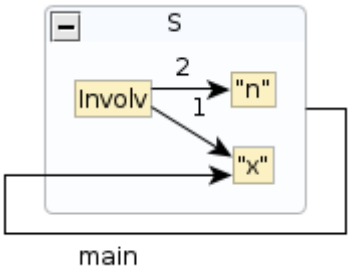
# IX. Involv

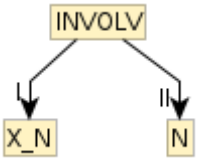
- Lex_involv

```
Sem<=>DSynt lex_Involv : no_patterns
leftside (∀)
l:?Zl {                 // LF
  sem=Involv
  l:1->l:?Xl{}          // base
  l:2->?Yl{}            // what is affected
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Zr { <=> ?Xl                                //Involv
  split=top
  dlex=?F.value
  dpos=V
  dsynt=OK
  tense=?Xl.tense
  voice=ACT
  lf=?F.name
  base=?L
  lf::(?F.name).(gp).(L)-> ?Xr{ <=> ?Xl         // Base
       split=bottom
       dpos=lexicon::(?L).(dpos)
       dlex=?L // won't work if ?Xr should be a LF
       dsynt=OK
  }
  lf::(?F.name).(gp).(X1)-> ?Yr{ <=> ?Yl        // what is affected
       dpos=lf::(?F).(gp).(lf::(?F).(gp).(X1)).(dpos)
       finiteness=lexicon::(lexicon::(?L).(?F)).(gp).(lf::(?F).(gp).(X1)).(finiteness)
       mood=lexicon::(lexicon::(?L).(?F)).(gp).(lf::(?F).(gp).(X1)).(mood)
       subject=lexicon::(lexicon::(?L).(?F)).(gp).(lf::(?F).(gp).(X1)).(subject)
       det=lexicon::(lexicon::(?L).(?F)).(gp).(lf::(?F).(gp).(X1)).(det)
  }
}

conditions (∃)
?F.name=?Zl.sem;
not ?Zr.dsynt=OK;               // ?Zr must not be lexicalized
not ?F.merged=yes;              // The collocate must not be merged with the base
```

Exemple :

(196) Involv(soleil) = inonder X

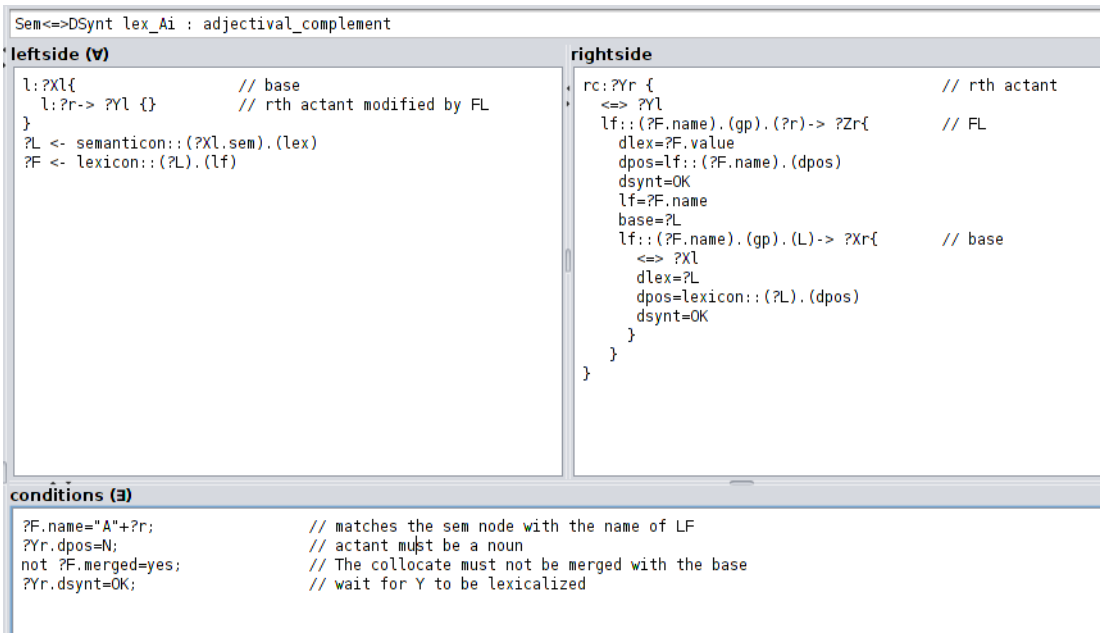


## X. Dérivation adjectivale

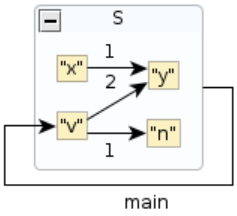
Les tests menés sur les dérivations adjectivales mettent en œuvre un nœud sémantique étiqueté ‘v’ désignant un sémant`me se lexicalaisant par un verbe quelconque V et son actant sémantique 1 ‘n’, une sémant`me se lexicalisant par un nom quelconque N. Ces éléments sont requis car seul un élément verbal peut être à la racine de l’arbre syntaxique profond.

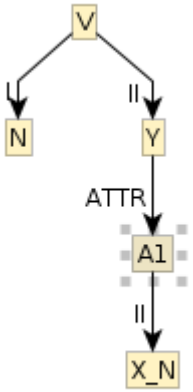
- Lex_ai




Exemple :

(197) A1(angoisse) = saisi



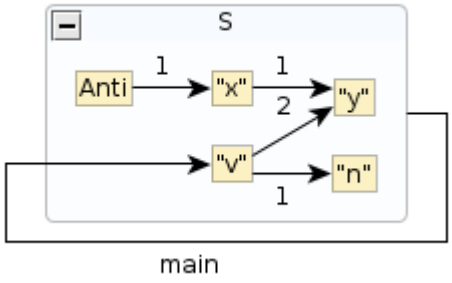



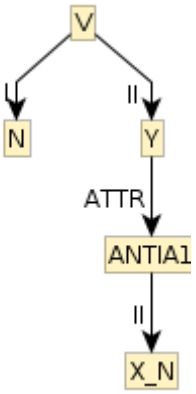
- Lex_anti_non_ai

```
Sem<=>DSynt lex_Anti_Non_Ai : adjectival_complement
leftside (∀)
l:?Wl {
  sem=Non|Anti
  l:1->l:?Xl{           //base
    l:?r-> ?Yl {}
}
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Yr {                                    // rth actant
  <=> ?Yl
  lf::(?F.name).(gp).(?r)-> ?Zr{            // FL
    dlex=?F.value
    dpos=lf::(?F.name).(dpos)
    dsynt=OK
    lf=?F.name
    base=?L
    lf::(?F.name).(gp).(L)-> ?Xr{           // base
      <=> ?Xl
      dlex=?L
      dpos=lexicon::(?L).(dpos)
      dsynt=OK
     }
   }
}
conditions (∃)
?F.name=?Wl.sem+"A"+?r;          // matches the sem node with the name of LF
?Yr.dpos=N;                      // actant must be a noun
not ?F.merged=yes;               // The collocate must not be merged with the base
?Yr.dsynt=OK;                    // wait for Y to be lexicalized
```

Exemple :

(198) AntiA1(faim) = repu



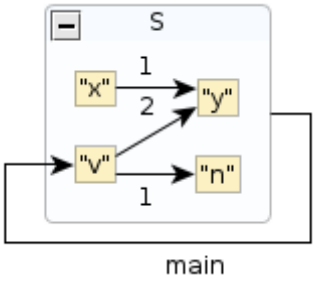
1. Combinaison avec les verbes supports

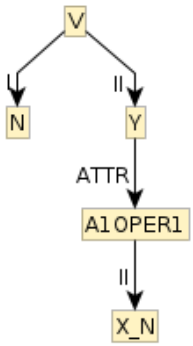
    - Lex_ai_vsupp_actant

```
Sem<=>DSynt lex_Ai_vsupp_actant : Ai_Vsupp
leftside (∀)
l:?Xl{          //base
  l:?r-> ?Yl {} //actant
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Yr {                                          // rth actant
  <=> ?Yl
  lf::(?F.name).(gp).(?r)-> ?Zr{    // Ai node
    dlex=?F.value
    dpos=lf::(?F.name).(dpos)
    dsynt=OK
    lf=?F.name
    base=?L
    lf::(?F.name).(gp).(L)-> ?Xr{
      <=> ?Xl                              // base
      dlex=?L
      dpos=lexicon::(?L).(dpos)
      dsynt=OK
     }
   }
}
conditions (∃)
?F.name="A1"+"Oper"+?r or ?F.name="A2"+"Func"+?r;
?Yr.dpos=N;                        // actant must be a noun
?Yr.dsynt=OK;                      // wait for Y to be lexicalized
not ?Xr.dsynt=OK;                  // Base must not be lexicalized
```

Exemple :

(199) A1Oper2



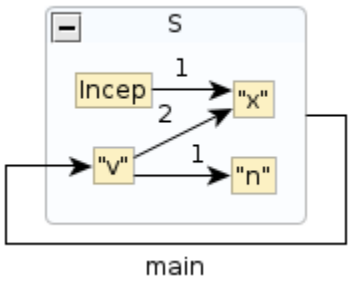



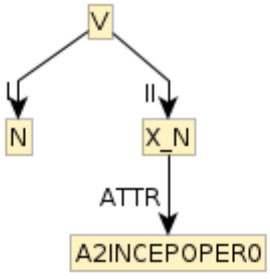
```
Sem<=>DSynt lex_Ai_phase_vsupp_base_0 : Ai_Vsupp
leftside (∀)
l:?Wl {
  sem=Prox|Incep|Cont|Fin|Prepar|Non
  l:1->l:?Xl{}          //base
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Xr {                          // base
  <=> ?Xl
   dlex=?L
   dpos=lexicon::(?L).(dpos)
   dsynt=OK
   lf::(?F.name).(gp).(L)-> ?Yr{  <=> ?Wl   // Ai node
      dlex=?F.value
      dpos=lf::(?F.name).(dpos)
      dsynt=OK
      lf=?F.name
      base=?L
  }
}
conditions (∃)
?F.name="A1"+?Wl.sem+"Func0" or ?F.name="A2"+?Wl.sem+"Oper0";
not ?Xr.dsynt=OK;                                  // not sure if we have to keep it
```

- Lex_ ai_phase_vsupp_base_0

Exemple

(200) A1IncepFunc0



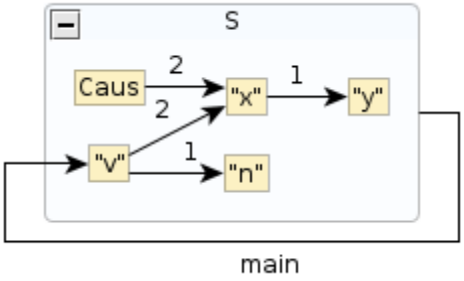



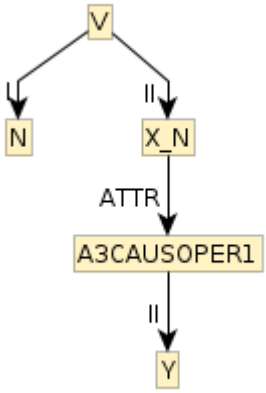
- Lex_ ai_causExt_vsupp_base_i

```
Sem<=>DSynt lex_Ai_causExt_vsupp_base_i : Ai_Vsupp
leftside (∀)
l:?Wl {
  sem=Caus|Liqu|Perm
  l:2->l:?Xl{           //base
    l:?r-> ?Yl {}
  }
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Xr {                          // base
  <=> ?Xl
   dlex=?L
   dpos=lexicon::(?L).(dpos)
   dsynt=OK
   lf::(?F.name).(gp).(L)-> ?Wr{ <=> ?Wl  // Ai node
      dlex=?F.value
      dsynt=OK
      lf=?F.name
      base=?L
      lf::(?F.name).(gp).(?r)-> ?Yr{            // actant
          <=> ?Yl
         }
  }
}

conditions (∃)
?F.name="A2"+?Wl.sem+"Func"+?r or "A3"+?Wl.sem+"Oper"+?r;
not ?Xr.dsynt=OK;                              // not sure if we have to keep it
```

Exemple :

(201) A3CausOper1



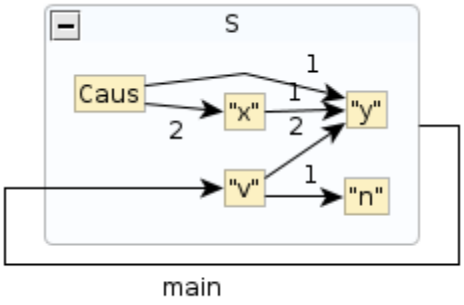



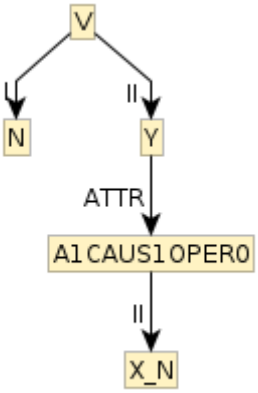
- Lex_ ai_causInt_vsupp_cause

```
Sem<=>DSynt lex_Ai_causInt_vsupp_cause : Ai_Vsupp
leftside (∀)
l:?Wl {
  sem=Caus|Liqu|Perm
  l:1-> ?Zl {}                    //agent
  l:2->l:?Xl{}                    //base
  }
?Xl{                              //base
  l:?r->?Zl {}                    //actant
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Zr {                                  // agent
  <=> ?Zl
   lf::(?F.name).(gp).(X1)-> ?Wr {
    <=> ?Wl                 // Ai + Cause
    dlex=?F.value
    dpos=lf::(?F.name).(dpos)
    dsynt=OK
    lf=?F.name
    base=?L
    lf::(?F.name).(gp).(L)-> ?Xr{         // base
          <=> ?Xl
         dlex=?L
         dpos=lexicon::(?L).(dpos)
         dsynt=OK
         }
  }
}
conditions (∃)
?F.name="A1"+?Wl.sem+?r+?Ul.sem+"Oper0" or ?F.name="A1"+?Wl.sem+?r+?Ul.sem+"Func0";
not ?F.merged=yes;                                  // The collocate must not be merged with the base
?Zr.dsynt=OK;                                       // wait for Y to be lexicalized
```

Exemple :

(202) A1Caus1Oper0



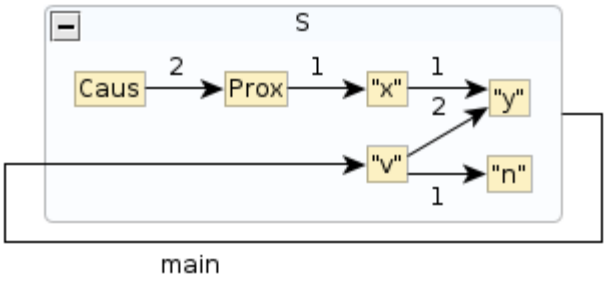



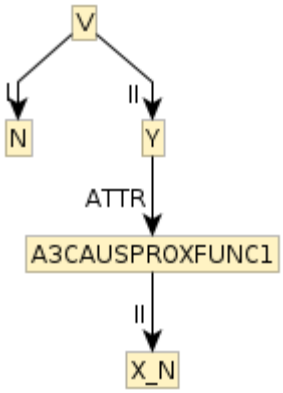
- Lex_ ai_causExt_phase_vsupp_actant

```
Sem<=>DSynt lex_Ai_causExt_Phase_vsupp_actant : Ai_Vsupp
leftside (∀)
l:?Wl {
  sem=Caus|Liqu|Perm
  l:2->l:?Ul{                          //phase
    sem=Incep|Fin|Cont|Prox|Prepar
    l:1->l:?Xl{ }                      //base
  }
}
?Xl{                             //base
  l:?r->?Zl {}                   //actant
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Zr {                                        // actant
  <=> ?Zl
   lf::(?F.name).(gp).(X1)-> ?Wr {
    <=> ?Wl <=> ?Ul                              // Ai node + Caus + Phase
    dlex=?F.value
    dpos=lf::(?F.name).(dpos)
    dsynt=OK
    lf=?F.name
    base=?L
    lf::(?F.name).(gp).(L)-> ?Xr{                // base
          <=> ?Xl
         dlex=?L
         dpos=lexicon::(?L).(dpos)
         dsynt=OK
         }
  }
}
conditions (∃)
?F.name="A2"+?Wl.sem+?Ul.sem+"Oper"+?r or ?F.name="A3"+?Wl.sem+?Ul.sem+"Func"+?r;
not ?F.merged=yes;                               // The collocate must not be merged with the base
?Zr.dsynt=OK;                                    // wait for Y to be lexicalized
```

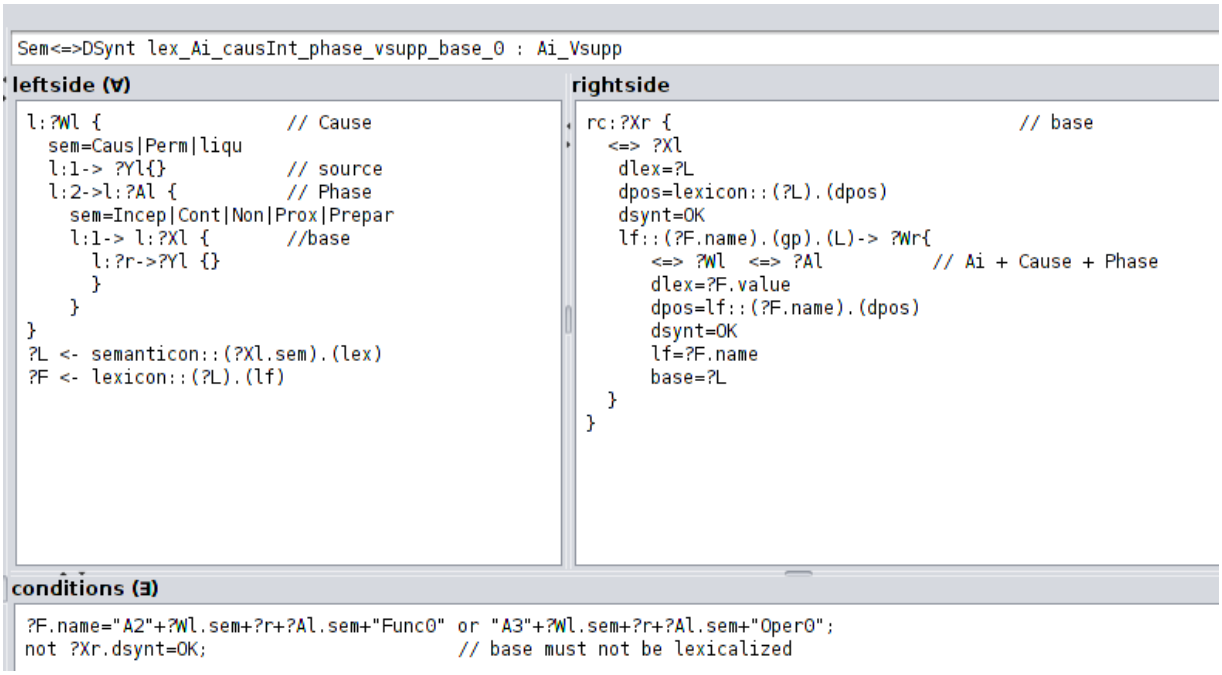
Exemple :

(203) A3CausProxFunc1



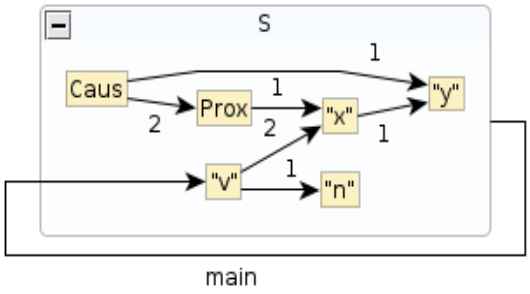
- Lex_ ai_causInt_ phase_vsupp_base_0

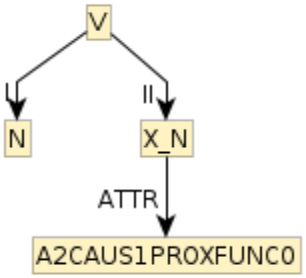


```
Sem<=>DSynt lex_Ai_causInt_phase_vsupp_base_0 : Ai_Vsupp
leftside (∀)
l:?Wl {                  // Cause
  sem=Caus|Perm|liqu
  l:1-> ?Yl{}            // source
  l:2->l:?Al {           // Phase
    sem=Incep|Cont|Non|Prox|Prepar
    l:1-> l:?Xl {        //base
      l:?r->?Yl {}
      }
    }
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Xr {                                  // base
  <=> ?Xl
   dlex=?L
   dpos=lexicon::(?L).(dpos)
   dsynt=OK
   lf::(?F.name).(gp).(L)-> ?Wr{
      <=> ?Wl  <=> ?Al           // Ai + Cause + Phase
      dlex=?F.value
      dpos=lf::(?F.name).(dpos)
      dsynt=OK
      lf=?F.name
      base=?L
  }
}
conditions (∃)
?F.name="A2"+?Wl.sem+?r+?Al.sem+"Func0" or "A3"+?Wl.sem+?r+?Al.sem+"Oper0";
not ?Xr.dsynt=OK;                       // base must not be lexicalized
```

Exemple :

(204) A2Caus1ProxFunc0



2. Combinaison avec les verbes de réalisation

- Lex_ ai_vreal_base_0

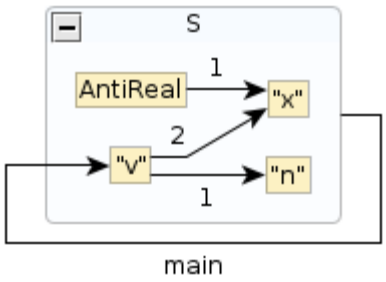
```
Sem<=>DSynt lex_Ai_vreal_base_0 : Ai_Vreal
leftside (∀)
l:?Zl{                  //Real
  sem=Real|AntiReal
  l:1-> l:?Xl{}         //base

}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
```

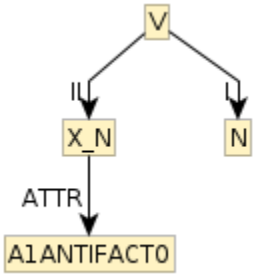
```
rightside
rc:?Xr {                                          // base
  <=> ?Xl
   dlex=?L
   dpos=lexicon::(?L).(dpos)
   dsynt=OK
   lf::(?F.name).(gp).(L)-> ?Yr{  <=> ?Zl   // Ai
      dlex=?F.value
      dpos=lf::(?F.name).(dpos)
      dsynt=OK
      lf=?F.name
      base=?L
  }
}
```

```
conditions (∃)
?F.name="A2"+?Zl.sem+"0" or
(?F.name="A1AntiFact0" and ?Zl.sem=lf::(?F.name).(sem)) or
(?F.name="A2Fact0" and ?Zl.sem=lf::(?F.name).(sem));
not ?Xr.dsynt=OK;               // Base must not be lexicalized
```

Exemple :

(205) A1AntiFact0



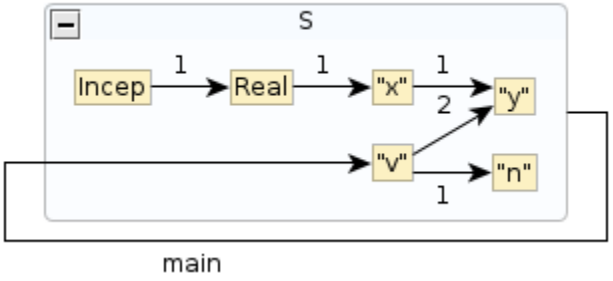



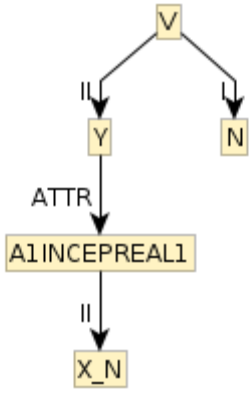
- Lex_ ai_phase_vreal_actant

```
Sem<=>DSynt lex_Ai_phase_vreal_actant : Ai_Vreal

leftside (∀)
l:?Wl {                          // Phase
  sem=Prox|Incep|Cont|Fin|Prepar|Non
  l:1->l:?Zl{                    //Real
    sem=Real|AntiReal
    l:1-> l:?Xl{                 //base
      l:?r-> ?Yl {}              //actant
    }
  }
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Yr {                                    // actant
  <=> ?Yl
  lf::(?F.name).(gp).(?r)-> ?Zr{
    <=> ?Zl <=>?Wl                          // Ai + Phase
    dlex=?F.value
    dpos=lf::(?F.name).(dpos)
    dsynt=OK
    lf=?F.name
    base=?L
    lf::(?F.name).(gp).(L)-> ?Xr{
      <=> ?Xl                               // base
      dlex=?L
      dpos=lexicon::(?L).(dpos)
      dsynt=OK
     }
   }
}

conditions (∃)
?F.name="A1"+?Wl.sem+?Zl.sem+?r or
(?F.name="A2"+?Wl.sem+"AntiFact"+?r and ?Zl.sem=lf::(?F.name).(sem)) or
(?F.name="A2"+?Wl.sem+"Fact"+?r and ?Zl.sem=lf::(?F.name).(sem));
?Yr.dpos=N;                      // actant must be a noun
?Yr.dsynt=OK;                    // wait for Y to be lexicalized
not ?Xr.dsynt=OK;                // Base must not be lexicalized
```

Exemple

(206) A1InceptReal1



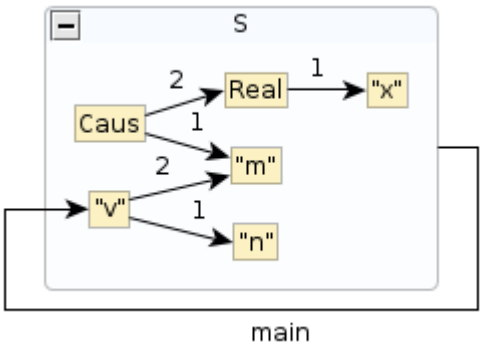



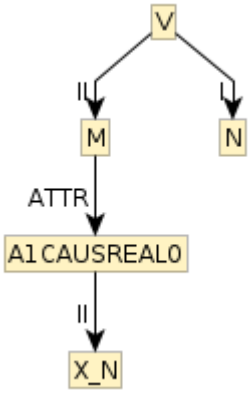
- Lex_ ai_causExt_vreal_cause

```
Sem<=>DSynt lex_Ai_causExt_vreal_cause : Ai_Vreal
leftside (∀)
l:?Wl {                    // Cause
  sem=Caus|Liqu|Perm
  l:2->l:?Zl {}
  l:1-> ?Ul {}             // agent
}
?Zl{                       // VReal
  sem=Real|AntiReal
  l:1-> l:?Xl {}           // base
  }
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Ur {                                  // agent
  <=> ?Ul
   lf::(?F.name).(gp).(X1)-> ?Zr {
    <=> ?Zl <=> ?Wl                        // Ai + Cause+  Vreal
    dlex=?F.value
    dpos=lf::(?F.name).(dpos)
    dsynt=OK
    lf=?F.name
    base=?L
    lf::(?F.name).(gp).(L)-> ?Xr{          // base
          <=> ?Xl
         dlex=?L
         dpos=lexicon::(?L).(dpos)
         dsynt=OK
         }
  }
}
conditions (∃)
//?F.name="A1CausReal0";
?F.name="A1"+?Wl.sem+?Zl.sem+"0" or
(?F.name="A1"+?Wl.sem+"AntiFact0" and ?Wl.sem+?Zl.sem=lf::(?F.name).(sem)) or
(?F.name="A1"+?Wl.sem+"Fact0" and ?Wl.sem+?Zl.sem=lf::(?F.name).(sem));
not ?Xr.dsynt=OK;              // base must not be lexicalized
```

Exemple

(207) A1CausReal0



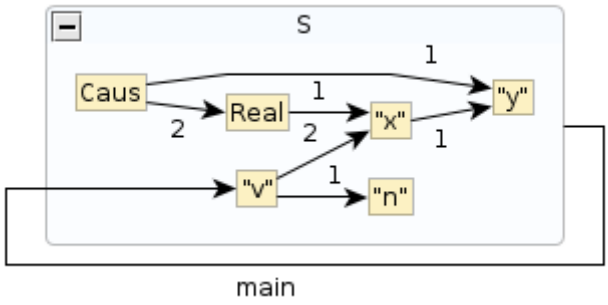



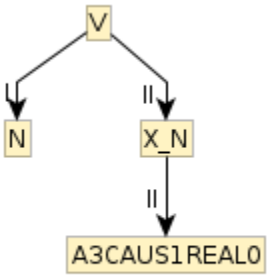
- Lex_ ai_causInt_vreal_base_0

```
Sem<=>DSynt lex_Ai_causInt_vreal_base_0 : Ai_Vreal
leftside (∀)
l:?Wl {                  // Cause
  sem=Caus|Perm|liqu
  l:1-> ?Yl{}            // source
  l:2->l:?Zl{            // Vreal
    sem=Real|AntiReal
    l:1-> ?Xl {}         // base
  }
}
?Xl {                          //base
     l:?r->?Yl {}              // source
   }
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Xr {                                  // base
  <=> ?Xl
   dlex=?L
   dpos=lexicon::(?L).(dpos)
   dsynt=OK
   lf::(?F.name).(gp).(L)-> ?Wr{
      <=> ?Wl <=> ?Zl                     // Ai + Cause + Vreal
      dlex=?F.value
      dpos=lf::(?F.name).(dpos)
      dsynt=OK
      lf=?F.name
      base=?L
  }
}
conditions (∃)
?F.name="A3"+?Wl.sem+?r+?Zl.sem+"0" or
(?F.name="A2"+?Wl.sem+?r+"AntiFact0" and ?Wl.sem+?Zl.sem=lf::(?F.name).(sem)) or
(?F.name="A2"+?Wl.sem+?r+"Fact0" and ?Wl.sem+?Zl.sem=lf::(?F.name).(sem));
not ?Xr.dsynt=OK;                       // base must not be lexicalized
```

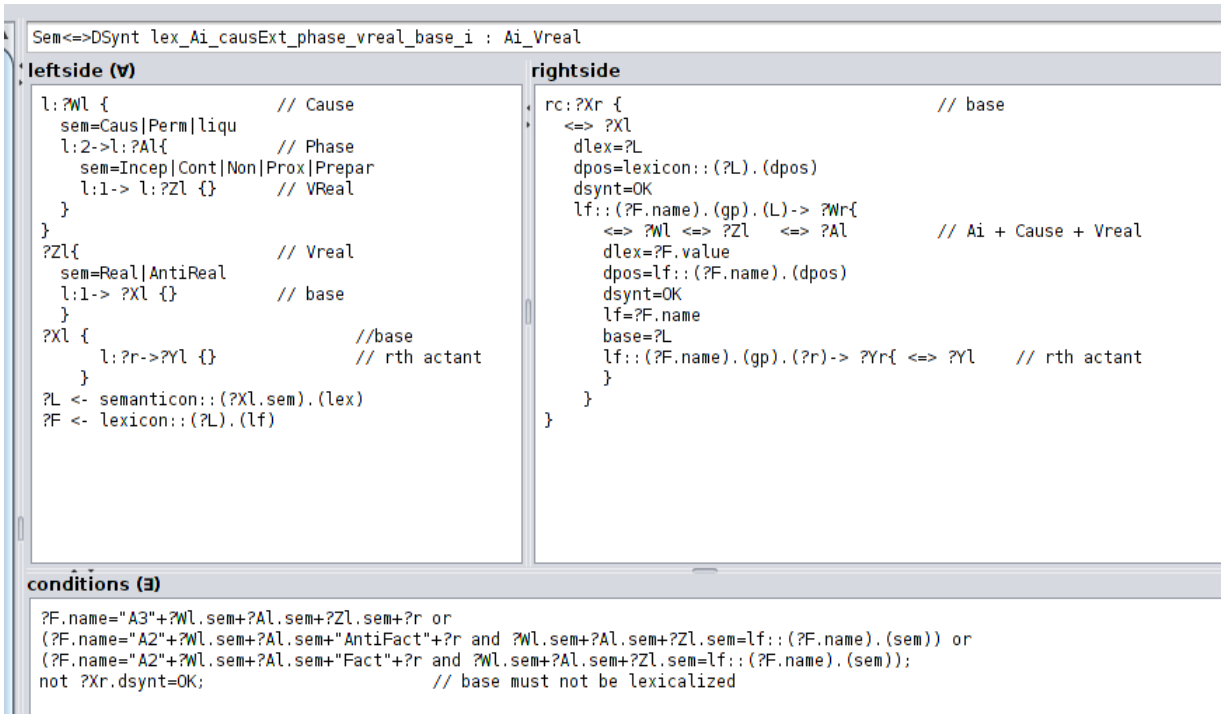
Exemple

(208) A3Caus1Real0



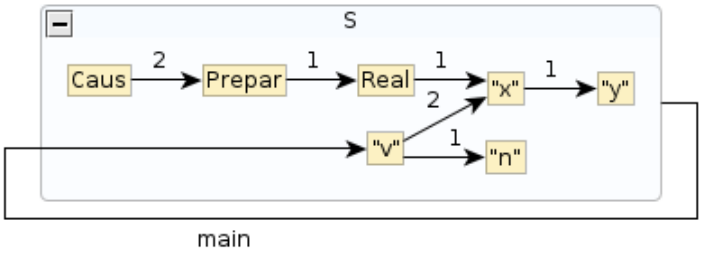
- Lex_ ai_causExt_ phase_vreal_base_i

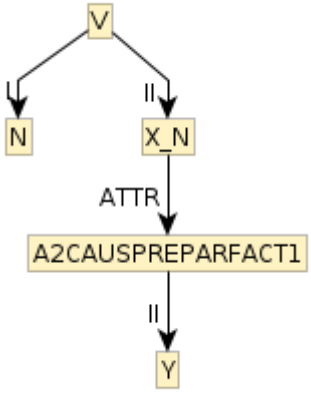


```
Sem<=>DSynt lex_Ai_causExt_phase_vreal_base_i : Ai_Vreal

leftside (∀)
l:?Wl {                  // Cause
  sem=Caus|Perm|liqu
  l:2->l:?Al{            // Phase
    sem=Incep|Cont|Non|Prox|Prepar
    l:1-> l:?Zl {}       // VReal
  }
}
?Zl{                     // Vreal
  sem=Real|AntiReal
  l:1-> ?Xl {}           // base
  }
?Xl {                            //base
      l:?r->?Yl {}               // rth actant
    }
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)

rightside
rc:?Xr {                                    // base
  <=> ?Xl
   dlex=?L
   dpos=lexicon::(?L).(dpos)
   dsynt=OK
   lf::(?F.name).(gp).(L)-> ?Wr{
      <=> ?Wl <=> ?Zl   <=> ?Al          // Ai + Cause + Vreal
      dlex=?F.value
      dpos=lf::(?F.name).(dpos)
      dsynt=OK
      lf=?F.name
      base=?L
      lf::(?F.name).(gp).(?r)-> ?Yr{ <=> ?Yl    // rth actant
      }
   }
}

conditions (∃)
?F.name="A3"+?Wl.sem+?Al.sem+?Zl.sem+?r or
(?F.name="A2"+?Wl.sem+?Al.sem+"AntiFact"+?r and ?Wl.sem+?Al.sem+?Zl.sem=lf::(?F.name).(sem)) or
(?F.name="A2"+?Wl.sem+?Al.sem+"Fact"+?r and ?Wl.sem+?Al.sem+?Zl.sem=lf::(?F.name).(sem));
not ?Xr.dsynt=OK;                       // base must not be lexicalized
```

Exemple

(209) A2CausPreparFact1



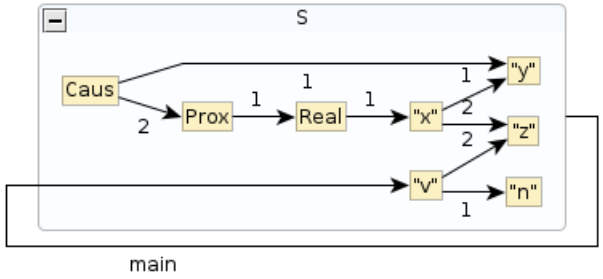



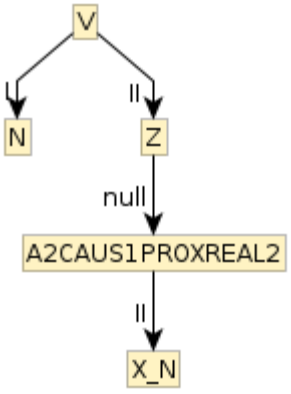
- Lex_ ai_causInt_ phase_vreal_actant

```
Sem<=>DSynt lex_Ai_causInt_phase_vreal_actant : Ai_Vreal
leftside (∀)
l:?Wl {                  // Cause
  sem=Caus|Perm|liqu
  l:1->?Yl {}            // agent
  l:2->l:?Al{            // Phase
    sem=Incep|Cont|Non|Prox|Prepar
    l:1-> l:?Zl {}       // VReal
  }
}
?Zl{                     // Vreal
  sem=Real|AntiReal
  l:1-> ?Xl {}           // base
  }
?Xl {                    //base
      l:?r->?Yl {}       // rth actant
      l:?l->?Ul {}       // lth actant
   }
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Ur {                                  // lth actant
  <=> ?Ul
  lf::(?F.name).(gp).(?r)-> ?Zr{
    <=> ?Zl <=> ?Wl <=> ?Al      // Ai + Cause + Phase + Vreal
    dlex=?F.value
    dpos=lf::(?F.name).(dpos)
    dsynt=OK
    lf=?F.name
    base=?L
    lf::(?F.name).(gp).(L)-> ?Xr{
      <=> ?Xl                             // base
      dlex=?L
      dpos=lexicon::(?L).(dpos)
      dsynt=OK
     }
   }
}
conditions (∃)
?F.name="A2"+?Wl.sem+?r+?Al.sem+?Zl.sem+?l or
(?F.name="A3"+?Wl.sem+?r+?Al.sem+"AntiFact"+?l and ?Wl.sem+?Al.sem+?Zl.sem=lf::(?F.name).(sem)) or
(?F.name="A3"+?Wl.sem+?r+?Al.sem+"Fact"+?l and ?Wl.sem+?Al.sem+?Zl.sem=lf::(?F.name).(sem));
?Ur.dpos=N;                    // actant must be a noun
?Ur.dsynt=OK;                  // wait for U to be lexicalized
not ?Xr.dsynt=OK;              // Base must not be lexicalized
```

Exemple

(210) A2Caus1ProxReal2



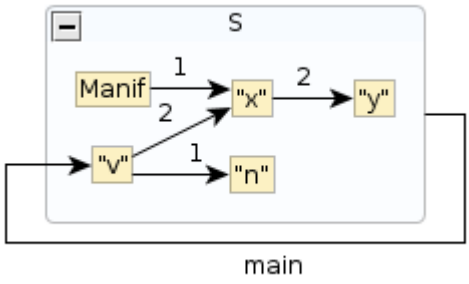
3. Combinaison avec les autres verbes sémantiquement pleins

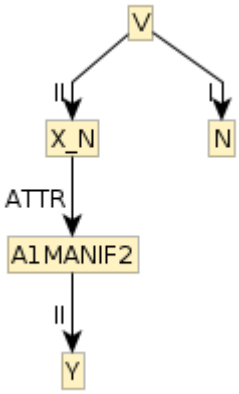
- Lex_ ai_vsem_base_i

```
Sem<=>DSynt lex_Ai_vsem_base_i : Ai_Vsem
leftside (∀)
l:?Zl{                  //VSem
  sem=Manif|Degrad|Excess|Stop|Obstr|Son|Sympt|
  AntiManif|AntiDegrad|AntiExcess|AntiStop|AntiObstr|AntiSon|AntiSympt
  l:1-> l:?Xl{}
}
?Xl{                    //base
    l:?r-> ?Yl {}       // rth actant
 }
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Xr {                                        // base
  <=> ?Xl
   dlex=?L
   dpos=lexicon::(?L).(dpos)
   dsynt=OK
   lf::(?F.name).(gp).(L)-> ?Zr {  <=> ?Zl      // Ai
      dlex=?F.value
      dpos=lf::(?F.name).(dpos)
      dsynt=OK
      lf=?F.name
      base=?L
      lf::(?F.name).(gp).(?r)-> ?Yr{            // rth actant
          <=> ?Yl
         }
  }
}
conditions (∃)
?F.name="A1"+?Zl.sem+?r;
not ?Xr.dsynt=OK;               // Base must not be lexicalized
not ?Yr.dsynt=OK;               // actant must not be lexicalized
```

Exemple :

(211) A1Manif2



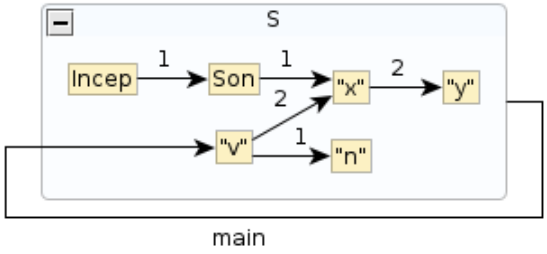



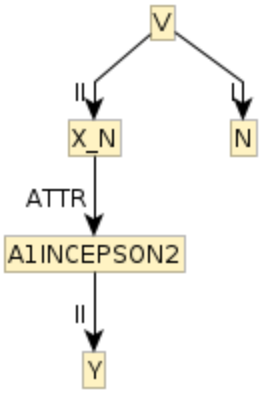
- Lex_ ai_phase_vsem_base_i

```
Sem<=>DSynt lex_Ai_phase_vsem_base_i : Ai_Vsem
leftside (∀)
l:?Wl {
  sem=Prox|Incep|Cont|Fin|Prepar|Non
  l:1->l:?Zl{                    //Real
    sem=Manif|Degrad|Excess|Stop|Obstr|Son|Sympt|
  AntiManif|AntiDegrad|AntiExcess|AntiStop|AntiObstr|AntiSon|AntiSympt
    l:1-> l:?Xl{                 //base
      l:?r-> ?Yl {}
    }
  }
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Xr {                                  // base
  <=> ?Xl
   dlex=?L
   dpos=lexicon::(?L).(dpos)
   dsynt=OK
   lf::(?F.name).(gp).(L)-> ?Wr{
      <=> ?Zl <=> ?Wl                     // Ai node + Phase + VSem
      dlex=?F.value
      dpos=lf::(?F.name).(dpos)
      dsynt=OK
      lf=?F.name
      base=?L
      lf::(?F.name).(gp).(?r)-> ?Yr{      // rth actant
          <=> ?Yl
        }
  }
}
conditions (∃)
?F.name="A1"+?Wl.sem+?Zl.sem+?r;
not ?Xr.dsynt=OK;                 // Base must not be lexicalized
```

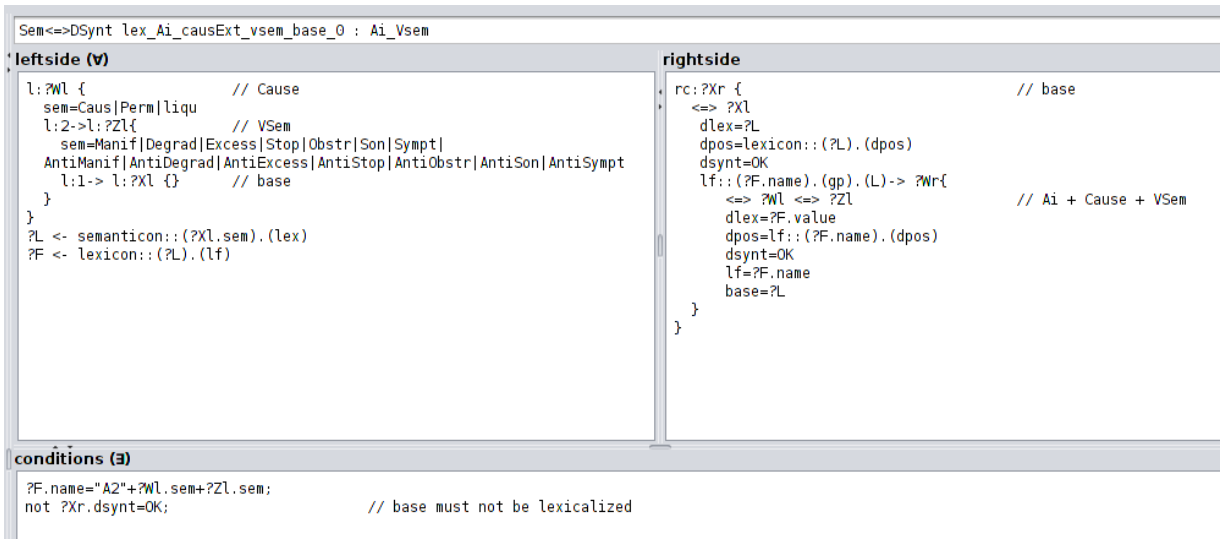
Exemple :

(212) A1InCepSon2



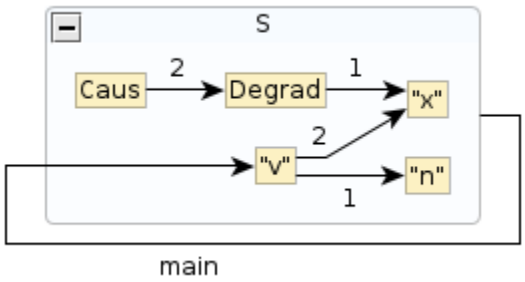



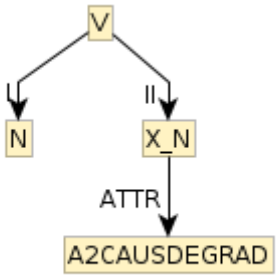
- Lex_ ai_causExt_vsem_base_0




Exemple

(213) A2CausDegrad



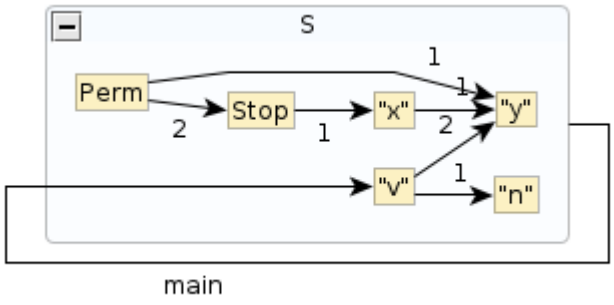



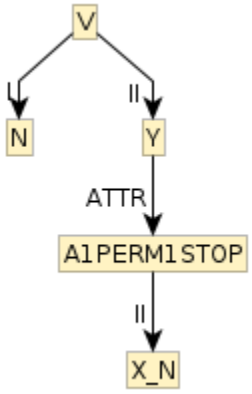
- Lex_ ai_causInt_vsem_cause

```
Sem<=>DSynt lex_Ai_causInt_vsem_cause : Ai_Vsem
leftside (∀)
l:?Wl {                  // Cause
  sem=Caus|Liqu|Perm
  l:2->l:?Zl {}          // VSem
  l:1-> ?Ul {}           // agent
}
?Zl{                     // VSem
  sem=Manif|Degrad|Excess|Stop|Obstr|Son|Sympt|
  AntiManif|AntiDegrad|AntiExcess|AntiStop|AntiObstr|AntiSon|AntiSympt
  l:1-> l:?Xl {          // base
    l:?r-> ?Ul {}        // agent
  }
}
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Ur {                                  // agent
  <=> ?Ul
   lf::(?F.name).(gp).(X1)-> ?Zr {
     <=> ?Zl <=> ?Wl                       // Ai + Cause+  VSem
     dlex=?F.value
     dpos=lf::(?F.name).(dpos)
     dsynt=OK
     lf=?F.name
     base=?L
     lf::(?F.name).(gp).(L)-> ?Xr{        // base
           <=> ?Xl
         dlex=?L
         dpos=lexicon::(?L).(dpos)
         dsynt=OK
         }
  }
}
conditions (∃)
?F.name="A1"+?Wl.sem+?r+?Zl.sem;
not ?Xr.dsynt=OK;                // base must not be lexicalized
```

Exemple :

(214) A1Perm1Stop



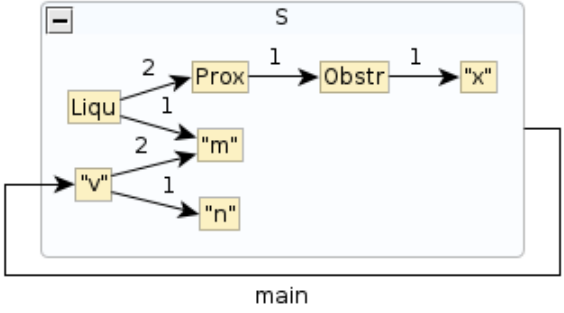



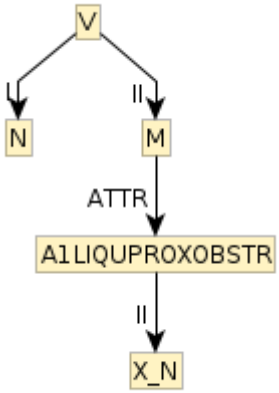
- Lex_ ai_causExt_ phase_vsem_cause

```
Sem<=>DSynt lex_Ai_causExt_phase_vsem_cause : Ai_Vsem
leftside (∀)
l:?Wl {                  // Cause
  sem=Caus|Liqu|Perm
  l:2->l:?Al{            // Phase
    sem=Incep|Cont|Non|Prox|Prepar
    l:1-> l:?Zl {}       // VSem
  }
  l:1-> ?Ul {}           // agent
}
?Zl{                     // VSem
  sem=Manif|Degrad|Excess|Stop|Obstr|Son|Sympt|
  AntiManif|AntiDegrad|AntiExcess|AntiStop|AntiObstr|AntiSon|AntiSympt
  l:1-> l:?Xl {}         // base
  }
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Ur {                                  // agent
  <=> ?Ul
   lf::(?F.name).(gp).(X1)-> ?Zr {
      <=> ?Wl <=> ?Zl   <=> ?Al           // Ai + Cause + Phase + Vreal
    dlex=?F.value
    dpos=lf::(?F.name).(dpos)
    dsynt=OK
    lf=?F.name
    base=?L
    lf::(?F.name).(gp).(L)-> ?Xr{         // base
          <=> ?Xl
        dlex=?L
        dpos=lexicon::(?L).(dpos)
        dsynt=OK
        }
  }
}
conditions (∃)
?F.name="A1"+?Wl.sem+?Al.sem+?Zl.sem;
not ?Xr.dsynt=OK;                // base must not be lexicalized
```

Exemple

(215) A1LiquProxObstr



- Lex_ ai_causInt_ phase_vsem_actant

```
Sem<=>DSynt lex_Ai_causInt_phase_vsem_actant : Ai_Vsem
leftside (∀)
l:?Wl {                  // Cause
  sem=Caus|Perm|liqu
  l:1->?Yl {}            // agent
  l:2->l:?Al{            // Phase
    sem=Incep|Cont|Non|Prox|Prepar
    l:1-> l:?Zl {}       // VSem
  }
}
?Zl{                     // VSem
  sem=Manif|Degrad|Excess|Stop|Obstr|Son|Sympt|
  AntiManif|AntiDegrad|AntiExcess|AntiStop|AntiObstr|AntiSon|AntiSympt
  l:1-> ?Xl {}           // base
  }
?Xl {                    //base
      l:?r->?Yl {}       // rth actant
      l:?l->?Ul {}       // lth actant
    }
?L <- semanticon::(?Xl.sem).(lex)
?F <- lexicon::(?L).(lf)
rightside
rc:?Ur {                                    // lth actant
  <=> ?Ul
  lf::(?F.name).(gp).(?r)-> ?Zr{
    <=> ?Zl <=> ?Wl <=> ?Al     // Ai + Cause + Phase + VSem
    dlex=?F.value
    dpos=lf::(?F.name).(dpos)
    dsynt=OK
    lf=?F.name
    base=?L
    lf::(?F.name).(gp).(L)-> ?Xr{
      <=> ?Xl                               // base
      dlex=?L
      dpos=lexicon::(?L).(dpos)
      dsynt=OK
     }
   }
}
conditions (∃)
?F.name="A3"+?Wl.sem+?r+?Al.sem+?Zl.sem+?l;
?Ur.dpos=N;                          // actant must be a noun
?Ur.dsynt=OK;                        // wait for U to be lexicalized
not ?Xr.dsynt=OK;                    // Base must not be lexicalized
```

Exemple :

(216) A3Caus1PreparManif2