Introduction

Nous souhaitons dans ce livre montrer en quoi la *réécriture de graphes* est un outil adapté au *traitement automatique de la langue naturelle*. Nous ne proposons pas ici une nouvelle théorie linguistique qui dépasserait les précédentes. Non, nous voulons montrer que la réécriture de graphes est un langage de programmation commun à de nombreuses modélisations linguistiques existantes, qu'elle permet de représenter les concepts mis en œuvre et de transformer les représentations les unes dans les autres de manière simple et pragmatique. S'il y a une forme d'universalité à laquelle nous pouvons nous attacher, c'est celle de la manière de calculer–le langage du calcul–plutôt que l'objet du calcul. Il reste que l'hétérogénéité de la langue demeure, hétérogénéité qui transparaît dans les théories linguistiques que nous allons rencontrer, et dont il faudra bien tenir compte dans la description du modèle de calcul.

Si nous prétendons que la réécriture de graphes est adaptée au traitement automatique de la langue naturelle, c'est parce que celle-ci possède des propriétés qui s'y prêtent.

Première observation importante, la langue suit des règles, par exemple celles que l'on qualifie couramment de *règles grammaticales*, parfois apprises à l'école (par exemple : « le verbe s'accorde avec son sujet »), mais souvent implicites et généralement « évidentes » pour un locuteur natif (en français, on peut dire *une voiture rouge* mais pas *une rouge voiture*). L'observation importante pour la suite, c'est que chacune d'elle ne concerne qu'un petit nombre d'éléments de la phrase directement liés par une *relation* (de sujet à verbe, de verbe à préposition, de complément à nom, etc.). Nous dirons qu'elles sont *locales*. Relevons que ces relations peuvent s'appliquer à des mots ou des syntagmes arbitrairement éloignés dans la phrase, par exemple un sujet séparé par une relative de son verbe.

Notons néanmoins dans l'usage de la langue quotidienne, notamment à l'oral, que l'on peut facilement trouver des occurrences de textes qui ne respectent pas ou seulement partiellement les règles établies. Pour des applications pratiques, il est donc nécessaire de considérer la langue dans ses différentes formes et d'être capable de gérer à la fois les règles et leurs emplois effectifs avec d'éventuelles exceptions.

Second aspect important que nous souhaitons relever à propos de la langue naturelle, elle présente de nombreuses formes d'ambiguïté. Par contraste avec les langages de programmation qui sont conçus pour être non ambigus et porteur d'une sémantique précise, les phrases en langue naturelle contiennent des ambiguïtés à tous les niveaux. Elles peuvent être d'ordre lexical comme dans la phrase *Mon avocat est très bon !* où l'avocat peut être un fruit ou une personne. Elles peuvent être d'ordre syntaxique comme dans l'exemple *Jean préfère la tarte aux pommes* qui peut signifier *La tarte aux pommes est le plat que Jean préfère* ou *Jean préfère choisir une tarte plutôt que des pommes*. Elles peuvent être de l'ordre du discours, par exemple au travers des anaphores : qui est « la » dans *Je la vois* ?

Dans l'usage courant de la langue par les humains, les ambiguïtés passent souvent inaperçues, car le contexte ou les connaissances extérieures permettent de les lever. En revanche, lors d'un traitement automatique, ces ambiguïtés sont beaucoup plus problématiques à trancher. Nous soutenons qu'un bon modèle de calcul doit laisser au programmeur le choix de résoudre les ambiguïtés ou non et à quel moment, et qu'à la manière de la programmation par contrainte, toutes les solutions doivent être *a priori* possibles. La gestion de la coexistence de solutions partielles est à la charge du programme, pas du programmeur.

L'étude de la langue sous les différentes facettes évoquées ci-dessus est l'objet principal de la linguistique. Dans cet ouvrage, notre objectif est de proposer des méthodes automatiques qui permettent de manipuler des représentations formelles de la langue naturelle, de les transformer les unes dans les autres. Nous nous appuierons donc systématiquement sur les modèles linguistiques existants pour décrire et motiver les représentations que nous allons considérer. Il est bien sûr hors de propos de donner ici des explications et des justifications linguistiques détaillées pour chaque formalisme employé, mais nous ferons de chacun d'eux une présentation suffisamment précise pour pouvoir suivre la suite sans connaissance préalable. Nous donnerons des références pour une étude approfondie.

I.1. Niveaux d'analyse

De nombreuses théories linguistiques existent et elles proposent des visions assez différentes des langues naturelles. Un des points communs à ces différentes théories est le fait de considérer plusieurs niveaux d'analyse qui se complètent : du plus élémentaire au plus complexe, le phonème à l'oral ou la lettre à l'écrit, puis le mot, la

phrase, le texte ou le discours. Notre objectif est de proposer un modèle suffisamment générique pour être compatible avec ces différents niveaux d'analyse et avec les différents choix linguistiques propres à chaque théorie.

Même si les structures de graphes peuvent rendre compte de différentes dimensions d'analyse de la langue, nous nous intéresserons ici essentiellement à la syntaxe et à la sémantique à l'échelle de la phrase. Ces deux dimensions sont incontournables dans le traitement de la langue et elles nous permettront d'illustrer de nombreux aspects de la réécriture de graphes. De plus, l'existence de corpus annotés de bonne qualité nous permet de valider les systèmes présentés en comparant les données calculées à des données de référence.

Le but de la syntaxe est de rendre compte de la structure d'une phrase. À ce niveau, on considère comme atomiques les unités lexicales (en pratique, très proches de ce que l'on appelle les « mots » dans le langage courant) et on s'intéresse à la façon dont ces unités atomiques s'assemblent pour former des phrases. Il n'y a pas de façon canonique de représenter ces structures et les travaux linguistiques utilisent, le plus souvent un des deux types de représentation : les syntagmes ou les dépendances.

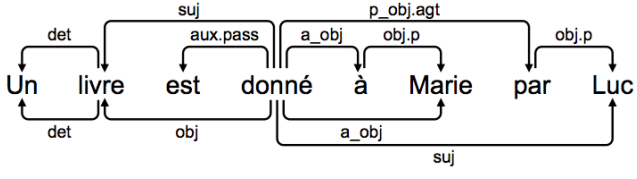
Le but de la représentation sémantique est de rendre compte du sens de la phrase. De façon très informelle, il s'agit de savoir « qui » a fait « quoi », « où », « comment », etc. La structure sémantique peut donc très bien s'éloigner de la forme linéaire des phrases. Ainsi, deux phrases syntaxiquement très différentes peuvent avoir la même représentation sémantique (on parle alors de paraphrases). Dans les faits, on constate vite que la modélisation de la sémantique de la langue est très complexe du fait de l'ambiguïté et des références externes non explicites. C'est pourquoi on retrouve dans la littérature de nombreux formalismes qui se focalisent sur une partie seulement de la sémantique. Cette focalisation peut porter sur le domaine des textes (par exemple, le domaine juridique) ou sur les phénomènes sémantiques (la DMRS, *Dependency Minimal Recursion Semantics*, s'intéresse aux portées des quantificateurs, l'AMR, *Abstract Meaning Representation*, à la mise en évidence des prédicats et de leurs arguments).

La logique formelle est toujours présente – plus ou moins explicitement – dans ces formalismes. En effet, pour une phrase transitive très simple comme *Max déteste Luc*, on interprète les deux noms propres comme des constantes et le verbe comme un prédicat $Hate$ dont les arguments sont les deux constantes. Les quantificateurs de la logique peuvent rendre compte de certains déterminants. La phrase, « Un homme entre. » peut ainsi être représentée par la formule de logique du premier ordre $\exists x(Man(x) \wedge Enter(x))$.

Dans la suite de cet ouvrage, nous reviendrons en détail sur plusieurs vues de la syntaxe et de la sémantique en nous appuyant sur des formalismes de la littérature et des exemples issus de corpus reflétant l'usage courant de la langue.

Les différences entre les structures syntaxiques et les structures sémantiques sont grandes et il est difficile de construire des modèles de l'interface entre ces deux niveaux. C'est pourquoi de nombreuses modélisations linguistiques (Mel'čuk, Chomsky) considèrent un niveau intermédiaire entre la syntaxe telle qu'on l'a décrite ci-dessus et la sémantique. Ce niveau intermédiaire est souvent appelé *syntaxe profonde*. Pour distinguer la syntaxe présentée ci-avant et la syntaxe profonde, la première est qualifiée de *syntaxe de surface*.

Sans entrer dans les détails qui viendront plus tard, la syntaxe profonde représente le plus grand dénominateur commun aux différents formalismes de représentation sémantique. Pour rester neutre vis-à-vis d'un formalisme sémantique précis, la syntaxe profonde utilise les mêmes étiquettes que celles de la syntaxe de surface pour décrire de nouvelles relations. C'est d'ailleurs pour cette raison que l'on continue de la qualifier de « syntaxe ». Ainsi, en syntaxe profonde, il s'agira par exemple d'identifier de nouveaux liens (absents en surface) entre un prédicat et l'un de ces arguments sémantiques, de neutraliser les changements de diathèses ou de repérer des mots grammaticaux qui sont amenés à disparaître dans la représentation sémantique. La structure profonde fait ainsi abstraction de certains détails de réalisation qui ne sont pas pertinents pour la sémantique. Dans le schéma ci-dessous, on illustre le cas d'une diathèse passive, la syntaxe de surface est représentée au-dessus et la syntaxe profonde en dessous.



I.2. Des arbres ou des graphes ?

Avec Chomsky et les structures syntagmatiques, la notion d'arbre s'est imposée en syntaxe comme structure mathématique sous-jacente. En effet, la structure récursive qui décrit un constituant à partir de ses sous-constituants directs conduit naturellement aux arbres. Dans la représentation en dépendances introduite par Tesnière, les informations linguistiques sont exprimées comme des relations binaires entre les unités lexicales atomiques. Ces unités atomiques peuvent être considérées comme des nœuds, les relations binaires comme des arcs entre ces nœuds, formant ainsi un graphe. Même si c'est d'une manière détournée, les dépendances sont également guidées par une vision syntagmatique de la syntaxe qui conduit naturellement à ne considérer que des structures en dépendances qui sont des arbres. Ainsi en pratique, dans la plupart

des corpus ou des outils, les relations de dépendances sont organisées en sorte qu'un des mots de la phrase soit considéré comme la racine de la structure et chacun des autres, la cible d'une et une seule relation. La structure est alors un arbre.

Dans cet ouvrage, nous plaidons pour un emploi systématique et unifié des graphes. Les arbres sont sensés faciliter les traitements et simplifier les algorithmes d'analyse. Mais l'argument n'est pas très solide, et comme nous le justifierons par de nombreuses expériences, les coûts calculatoires sont tout à fait acceptables en pratique. En outre, les outils que nous présenterons par la suite ont été conçus pour qu'ils n'aient aucun surcoût lorsqu'ils s'appliquent à des arbres.

Si la restriction à des arbres est concevable dans les structures syntaxiques, elle est beaucoup plus problématique pour les autres niveaux, en particulier dans les structures sémantiques : une même entité peut jouer simultanément un rôle pour différents prédicats, cette entité est alors la cible d'une relation pour chacun de ces rôles. *A minima*, cela produit des graphes acycliques, dans les faits des graphes dans toute leur généralité. Les formalismes existants pour la sémantique que nous allons utiliser plus loin (AMR et DMRS) utilisent ainsi pleinement les structures de graphes.

Mais en fait, même au niveau syntaxique les arbres se révèlent insuffisants. Lorsque l'on enrichit les structures avec des informations de syntaxe profonde (comme les sujets d'infinitifs ou les antécédents des pronoms relatifs), on produit des structures avec des cycles qui justifient alors l'utilisation de graphes. Les graphes permettent en outre de prendre en compte simultanément plusieurs niveaux linguistiques de façon uniforme (par exemple, la structure syntaxique et l'ordre linéaire de mots). On peut noter qu'en pratique les formalismes basés sur les arbres introduisent souvent des mécanismes *ad hoc* comme les co-indexations d'indices pour représenter des relations qui casseraient la structure d'arbre. Les graphes permettent un traitement uniforme de ces mécanismes.

I.3. Corpus annotés linguistiquement

Même si le travail des lexicographes et des linguistes basé sur l'introspection est souvent indispensable pour construire des dictionnaires et des grammaires (inventaires de règles) en étudiant les constructions de la langue, leurs usages et leurs limites, ce travail n'est pas toujours suffisant. L'utilisation de corpus de grande taille permet d'aborder d'autres aspects dans l'étude de la langue. Du point de vue linguistique, la recherche sur corpus permet d'observer les fréquences d'usage de différentes constructions, d'étudier les variations de la langue suivant divers paramètres : géographiques, historiques ou par type de texte que l'on considère (littérature, journalistique, technique, etc.). Comme nous l'avons dit précédemment l'usage de la langue fait parfois fi des règles décrites par le linguiste. Ainsi, même si une construction ou un usage

est considéré comme incorrect, mais présent dans les corpus, il est nécessaire d'en tenir compte dans les applications pratiques.

Les approches linguistiques fondées sur l'intelligence artificielle, et plus généralement les probabilités, s'appuient sur des corpus d'observations pour leur apprentissage. Ils sont également employés comme référence pour le test d'outils.

Les corpus bruts (des collections de texte) permettent de réaliser quelques-unes des tâches décrites ci-avant. Cependant, pour de nombreuses applications et pour les recherches linguistiques plus complexes, ce texte brut ne suffit pas, il faut qu'il soit complété par des informations linguistiques, on parle alors de corpus annotés. La création de tels corpus est une tâche fastidieuse et de longue haleine. Dans ce livre, nous aborderons cette problématique, notamment en proposant des outils soit en amont pour préparer (on dira pré-annoter) les corpus, soit en aval, dans les phases de maintenance et de corrections de corpus existants. Une solution souvent utilisée pour construire des ressources annotées qui suivent des choix linguistiques précis est de transformer, automatiquement dans la mesure du possible, des ressources préexistantes. La plupart des corpus présents dans le projet *Universal Dependencies*[1] (UD) sont des corpus qui étaient déjà annotés dans le cadre d'autres projets et qui ont été convertis vers le format UD. Nous reviendrons plus loin sur ce type d'applications.

I.4. La réécriture de graphes

Rappelons que notre objectif est de montrer que la réécriture de graphes est un modèle de calcul intéressant pour le traitement automatique des langues (TAL). L'idée centrale de la réécriture est de décomposer les transformations en une série de transformations élémentaires qu'il est plus facile à décrire et contrôler. Plus précisément, la réécriture consiste à exécuter des règles, c'est-à-dire à 1) décrire par des motifs les conditions d'applications locales de la transformation élémentaire et 2) de décrire par des commandes locales la transformation du graphe.

Une idée sous-jacente à cette thèse, c'est que la description des transformations s'appuie sur une analyse linguistique qui – comme nous l'avons déjà indiqué – se prête bien aux analyses locales. Deuxième point, la réécriture est indépendante des formalismes, elle peut même gérer correctement la coexistence de plusieurs niveaux linguistiques, typiquement elle peut s'appliquer à des graphes composites, c'est-à-dire constitués de liens hétérogènes (à la fois syntaxiques et sémantiques par exemple). Troisième point, la réécriture ne fixe pas d'ordre d'application des règles ni le lieu de leur application. Dans les faits, cela libère largement le programmeur de la conception

1. http://universaldependencies.org.

des algorithmes et de la planification et il peut se concentrer sur les aspects linguistiques du problème. Quatrième point, le modèle de calcul est intrinsèquement non déterministe : deux règles « contradictoires » peuvent s'appliquer au même graphe au même endroit. Le phénomène se produit en cas d'ambiguïté linguistique (lexicale, syntaxique, sémantique) lorsque deux options se présentent (dans, *il voit la fille avec un télescope*, sur quoi porte le *téléscope* ?) qui chacune correspond à une règle. En s'appuyant sur une stratégie, le programmeur a le choix de poursuivre le calcul selon les deux possibilités ou de privilégier l'une des deux.

Nous reviendrons en détail sur le formalisme de réécriture de graphes considéré, mais nous donnons ici ses principales caractéristiques. Comme traditionnellement en réécriture, on parlera de partie gauche d'une règle pour décrire les conditions de son application et de partie droite pour décrire l'effet de la règle sur la structure hôte.

La partie gauche d'une règle, appelée *motif*, est décrite par un graphe (qui sera recherché dans le graphe à modifier) et par un ensemble de contraintes négatives permettant de mieux contrôler le contexte d'application des règles. Toujours dans la partie gauche, il est également possible de paramétrer les règles par des informations lexicales externes. La recherche de motifs de graphes est un problème NP-complet et donc potentiellement problématique pour les applications pratiques, l'utilisation qui en est faite ici évite cet écueil : les motifs sont petits (ils ont rarement plus de 5 nœuds) et sont recherchés dans des graphes de quelques dizaines (ou plus rarement quelques centaines) de nœuds. Par ailleurs, les motifs ont souvent une structure d'arbre, auquel cas la recherche est extrêmement efficace.

Pour la partie droite des règles, des commandes atomiques (création d'arcs, suppression d'arc) décrivent la transformation opérée localement sur le graphe. Il existe également des commandes plus globales (décalage) qui permettent de gérer les liens entre le motif repéré et le reste du graphe. La création de nouveaux nœuds est limitée : des commandes pour ajouter des nœuds existent, mais elles ont un statut particulier. La majeure partie des systèmes travaillent sans créer de nouveaux nœuds. Le fait peut être exploité pour améliorer l'efficacité de la réécriture.

Les transformations globales peuvent faire intervenir un grand nombre d'étapes intermédiaires, décrites par un grand nombre de règles (plusieurs centaines dans les exemples présentés plus loin). Il est donc nécessaire de contrôler la façon dont les règles sont appliquées lors des transformations. Pour cela, l'ensemble des règles d'un système est organisé de façon modulaire : des paquets permettent de regrouper des sous-ensembles cohérents de règles et les stratégies décrivent l'ordre et la manière d'appliquer les règles.

La notion de réécriture de graphes pose des problèmes de définition mathématique, notamment pour décrire comment chaque transformation locale interagit avec

le contexte du motif de la règle. Il y a une approche fondée sur la théorie des catégories avec deux sous-variantes, la version SPO (*Single Pushout*) et la version DPO (*Double Pushout*) [ROZ 97]. Un autre point de vue vient de la logique [COU 12] en profitant de la décidabilité de la logique monadique du second ordre. Ces approches ne sont pas adéquates pour l'usage que nous en avons. Les graphes employés n'ont pas, à notre connaissance, de structure algébrique sous-jacente ou de paramètre limitant (comme la largeur d'arbre) attendu pour l'approche logique. D'autre part, nous avons besoin des commandes de type `shift` (décalage) qui ne sont pas compatibles avec les approches actuelles de la théorie des catégories. Nous ne pouvons qu'encourager le lecteur à développer lui-même l'aspect théorique sous-jacent à la forme de réécriture que nous déployons ici.

Nous présentons donc de façon plus opérationnelle la réécriture et les règles en mettant l'accent sur un langage adapté au traitement de la langue naturelle. Nous avons identifié quelques éléments clés spécifiques à ce sujet :

– les conditions négatives sont indispensables pour éviter les sur-interprétations ;

– il est également indispensable de disposer de modules/paquets sans quoi la conception de systèmes de réécritures devient inextricable ;

– il est nécessaire de prévoir une forte connexion avec des lexiques : sans cela, il y aurait des milliers de règles rendant la réécriture difficile à mettre au point et sans doute inefficace ;

– il faut prévoir une notion de stratégie pour l'organisation séquentielle des modules et la résolution des ambiguïtés. À nouveau, l'objectif est de simplifier la conception générale du système de réécriture.

Dans ce livre nous nous appuierons sur GREW qui est un outil générique de réécriture de graphes qui répond aux besoins évoqués ci-dessus. Avec cet outil, des systèmes de règles ont été écrits pour chacune des applications décrites plus loin dans l'ouvrage. Dans la littérature, il existe d'autres outils et quelques descriptions ponctuelles d'emploi de la réécriture de graphes dans le contexte du traitement automatique de la langue (voir par exemple [BED 09, BOH 01, CHA 10, CRO 05, HYV 84, JIJ 07]).

Cependant, à notre connaissance, il existe peu de systèmes génériques de réécriture de graphes largement déployés et permettant de travailler à divers niveaux de description de la langue (une exception notable est le système Ogre [RIB 12]). Nous proposons ici un tel système et donnons un large éventail d'applications possibles de cette approche pour le traitement des langues.

I.5. Questions pratiques

Bien que l'écrit et l'oral soient deux manifestations de la langue naturelle, l'oral pose des problèmes spécifiques (traitement du signal, disfluences, ambiguïté phonétique) que nous ne traiterons pas ici ; nous nous intéresserons uniquement à la langue écrite.

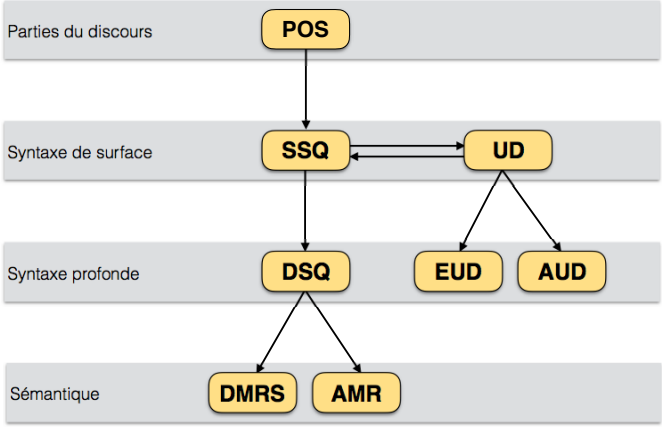


Figure I.1. *Formats et systèmes de réécriture considérés dans le livre*

Comme indiqué plus haut, nous avons travaillé sur les niveaux de la syntaxe et de la sémantique. Pour valider notre approche et l'appliquer à des données réelles de grande taille, nous avons utilisé le français comme terrain d'expérimentation. La figure I.1 présente les différents niveaux linguistiques que l'on a considéré dans les exemples de ce livre (les boîtes horizontales) et pour chaque niveau un ou plusieurs formats linguistiques existant dans la littérature pour ce niveau.

Notre objectif est d'étudier comment il est possible de programmer les conversions entre ces formats. Ces transformations peuvent être internes à un niveau linguistique (les flèches horizontales du schéma) et permettent de convertir automatiquement des données entre différentes descriptions linguistiques d'un même niveau. Les transformations peuvent également passer d'un niveau à un autre (les flèches descendantes du schéma) et réalisent donc des outils automatiques d'analyse syntaxique ou sémantique pour la langue[2]. Ces différentes transformations sont détaillées plus loin dans cet ouvrage.

2. Nous n'avons pas pour l'instant expérimenté de transformations inverses (des flèches montantes) qui seraient utiles pour de la génération de texte.

Pour expérimenter les outils et méthodes proposés, nous nous appuierons sur deux corpus librement disponibles, annotés en syntaxe de dépendances pour la langue française. Nous avons choisi Sequoia[3], qui est un corpus contenant 3 099 phrases issues de différents domaines : articles de presse (sous-corpus *annodis_er*), textes du parlement européen (sous-corpus *Europar.550*), notices médicales (sous-corpus *emea-fr-dev* et *emea-fr-test*), Wikipédia français (sous-corpus *frwiki_50.1000*). À l'origine, il a été annoté en constituants en suivant le schéma d'annotation du *French Treebank* (FTB) [ABE 04]. Il a été ensuite converti automatiquement en dépendances de surface [CAN 12b]. Les dépendances à distance ont été corrigées manuellement [CAN 12a]. Enfin, Sequoia a été annoté en dépendances profondes [CAN 14]. Même si le schéma d'annotation suivi est celui du FTB (qui a été défini bien avant l'existence de Sequoia), nous l'appellerons abusivement format « Sequoia » car c'est comme format accompagnant le corpus Sequoia que nous l'utilisons dans la suite.

Le second corpus utilisé fait partie du projet *Universal Dependencies*[4] (UD). Le but du projet UD est de définir un schéma d'annotation commun pour un maximum de langues et de coordonner la création d'un ensemble de corpus pour ces langues. C'est une tâche difficile, car les schémas d'annotation des corpus pré-existants ont tendance à être spécifiques à chaque langue. Dans UD, un guide général d'annotation précise un certain nombre de choix que les développeurs des corpus pour chaque langue doivent suivre et compléter pour leur langue. En pratique, ce guide général n'est pas encore complètement fixé et fait l'objet de nombreuses discussions. Le corpus UD_FRENCH est l'un des corpus en français dans UD. Il est constitué d'environ 16 000 phrases issues de différents types de texte (blog, news, avis de consommateurs et Wikipédia). Les annotations ont été créées dans le cadre du projet Google DataSet [MCD 13] avec une seule validation manuelle des données. Pour être intégrées dans le projet UD, les annotations ont été converties automatiquement (UD, version 1.0, janvier 2015). Cinq nouvelles versions ont été publiées depuis, la dernière étant la version 2.0 (mars 2017). De nombreuses vérifications, corrections et enrichissements ont été apportés dans les différentes versions (dont une partie significative à l'aide des outils présentés dans cet ouvrage). Cependant, le corpus actuel n'a pas fait l'objet d'une validation manuelle systématique.

I.6. Plan du livre

Le premier chapitre présente de façon pratique, les notions qui seront utilisées dans toute la suite du livre. Le lecteur pourra se familiariser avec la manipulation des graphes en PYTHON et avec l'outil GREW qui permet d'exprimer les règles de

3. https ://deep-sequoia.inria.fr.

4. http ://universaldependencies.org.

réécriture et les transformations de graphes utilisées ensuite. Les quatre chapitres suivants alternent des présentations linguistiques décrivant les niveaux d'analyse qui sont considérés : le chapitre 2 présente la syntaxe (en distinguant syntaxe de surface et syntaxe profonde) et le chapitre 4 aborde la question de la représentation sémantique (à travers deux propositions de formalisation sémantique, l'AMR et la DMRS). Chacun de ces chapitres est suivi par une mise en application dans des systèmes de réécriture de graphes manipulant les structures linguistiques présentées. Ainsi, le chapitre 3 décrit l'application de la réécriture aux transformations d'annotations syntaxiques et le chapitre 5 présente l'utilisation de la réécriture pour calculer des représentations sémantiques. Le chapitre 6 revient sur la syntaxe et présente l'analyse syntaxique par réécriture de graphes ; même si son objectif est complémentaire de celui du chapitre 3, il présente un système plus complexe et nous avons donc préféré le présenter dans un chapitre dédié, plus loin dans le livre. Les deux derniers chapitres reviennent sur les notions présentées auparavant en donnant les définitions mathématiques rigoureuses au chapitre 7 qui permettent d'étudier les propriétés du modèle de calcul au chapitre 8 notamment en ce qui concerne de terminaison et de confluence. La plupart des chapitres contiennent également des exercices et des parties présentant les « bonnes pratiques ». Nous espérons que cela permettra au lecteur de s'approprier les notions et les outils qu'il pourra ensuite réutiliser pour son propre usage.

Le travail présenté ici est le fruit d'une collaboration de plusieurs années des trois auteurs. Il est difficile d'attribuer précisément à chaque auteur ses contributions respectives, mais les rôles ont été complémentaires. Guillaume Bonfante a contribué de façon essentielle à la partie mathématique, et donc notamment au contenu des deux derniers chapitres ; Bruno Guillaume est le développeur de l'outil GREW ; Guy Perrier a développé la plus grande partie des systèmes de réécriture décrits dans le livre et a contribué aux chapitres qui décrivent ces systèmes, ainsi qu'aux parties linguistiques du livre. Les auteurs remercient Mathieu Morey qui a participé aux premiers travaux sur le sujet [MOR 11] ainsi que Marie Candito et Djamé Seddah avec qui les travaux [CAN 14, CAN 17] ont été réalisés.

Ce livre reprend en partie les publications suivantes : [BON 10, BON 11a, BON 11b, BON 13a, BON 13b, CAN 14, CAN 17, GUI 12, GUI 15a, GUI 15b, PER 12]. L'ensemble des outils et des ressources présentées dans le livre sont librement disponibles à partir du site grew.fr.