Implication Textuelle et Réécriture

Paul Bedaride

Nancy-Université
Université Henri Poincaré

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Implication Textuelle et Réécriture

THÈSE

présentée et soutenue publiquement le 18 octobre 2010

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Paul Bédaride

Composition du jury

Rapporteurs :	Philippe Blache	Directeur de Recherche, CNRS, Aix-en-Provence
	Patrick Saint-Dizier	Directeur de Recherche, CNRS, Toulouse
Examinateurs :	Claire Gardent (Directrice)	Directrice de Recherche, CNRS, Nancy
	Monique Grandbastien	Professeur, Université Nancy 1, Nancy
	Christian Retoré	Professeur, Université Bordeaux 1, Bordeaux

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria

Mis en page avec la classe thloria.

Remerciements

J'aimerais tout d'abord remercier Claire Gardent pour m'avoir fourni un encadrement où je n'étais pas laissé à moi-même et où je pouvais faire mes propres choix, et pour m'avoir conseillé de bonnes pistes de recherche à étudier. Je remercie aussi les membres de mon jury qui ont accepté d'évaluer mon travail de thèse.

Je tiens ensuite à remercier ma petite Laura, que j'ai rencontré au début de ma thèse, et qui m'a soutenu durant celle-ci et m'a permis de me changer les idées quand j'en avait besoin.

Je remercie ensuite les personnes avec qui j'ai discuté de recherche pendant ma thèse, c'est-à-dire (par ordre alphabétique pour ne pas faire de jaloux) : Carlos Areces, Paul Brauner, Alexande Denis, Karën Fort, Sebastien Hinderer, Guillaume Hoffmann, Jonathan Marchand, Yannick Parmentier, Florent Pompigne, et ceux que j'ai oublié.

Je remercie aussi les membres du chan linux pour tout les trolls qu'ils m'ont fourni (avec une mention spéciale pour Jonathan), c'est-à-dire chap, fred, jarod, jonathlela, mumu, pini, polux, smelc, trigger.

Enfin pour finir je remercie ma famille et le reste de mes amis pour les moments de détente passés avec eux et leur soutient.

Résumé

Cette thèse propose plusieurs contributions sur le thème de la détection d'implications textuelles (DIT). La DIT est la capacité humaine, étant donné deux textes, à pouvoir dire si le sens du second texte peut être déduit à partir de celui du premier. Une des contributions apportée au domaine est un système de DIT hybride prenant les analyses d'un analyseur syntaxique stochastique existant afin de les étiqueter avec des rôles sémantiques, puis transformant les structures obtenues en formules logiques grâce à des règles de réécriture pour tester finalement l'implication à l'aide d'outils de preuve. L'autre contribution de cette thèse est la génération de suites de tests finement annotés avec une distribution uniforme des phénomènes couplée avec une nouvelle méthode d'évaluation des systèmes utilisant les techniques de fouille d'erreurs développées par la communauté de l'analyse syntaxique permettant une meilleure identification des limites des systèmes. Pour cela nous créons un ensemble de formules sémantiques puis nous générons les réalisations syntaxiques annotées correspondantes à l'aide d'un système de génération existant. Nous testons ensuite s'il y a implication ou non entre chaque couple de réalisations syntaxiques possible. Enfin nous sélectionnons un sous-ensemble de cet ensemble de problèmes d'une taille donnée et satisfaisant un certain nombre de contraintes à l'aide d'un algorithme que nous avons développé.

Mots-clés: chat, chien, puces.

Abstract

This thesis presents several contributions on the theme of recognising textual entailment (RTE). The RTE is the human capacity, given two texts, to determine whether the meaning of the second text could be deduced from the meaning of the first or not. One of the contributions made to the field is a hybrid system of RTE taking analysis of an existing stochastic parser to label them with semantics roles, then turning obtained structures in logical formulas using rewrite rules to finally test the entailment using proof tools. Another contribution of this thesis is the generation of finely annotated tests suites with a uniform distribution of phenomena coupled with a new methodology of systems evaluation using error minning techniques developed by the community of parsing allowing better identification of systems limitations. For this, we create a set of formulas, then we generate annotated syntactics realisations corresponding by using an existing generation system. Then, we test whether or not there is an entailment between each pair of possible syntactics realisations. Finally, we select a subset of this set of problems of a given size and a satisfactory a certain number of constraints using an algorithm that we developed.

Keywords: chat, chien, puces.

Table des matières

Table des figures

Liste des tableaux

Introduction

Le traitement automatique des langues (TAL) a pour objectif d'interpréter ou de produire des textes en langue naturelle. Il existe de nombreux types d'applications relevant de ce domaine, comme les systèmes de recherche d'information permettant de trouver des documents sur un thème particulier, les systèmes de question-réponse permettant de chercher la réponse la plus appropriée à une question formulée en langue naturelle dans un ensemble de documents ou encore les systèmes de synthèse de document permettant de générer une version résumée du contenu d'un document. L'un des points communs de ces différents systèmes est qu'ils doivent interpréter des textes en langue naturelle et plus précisément, qu'ils doivent tous traiter du problème de la détection d'implications textuelles.

La détection d'implications textuelles (DIT) consiste à décider, étant donné deux fragments de texte, si le sens de l'un des textes est impliqué par celui de l'autre. Cette tâche a été introduite par Dagan et Glickman et la première campagne d'évaluation appelée *Recognizing Textual Entailment* (RTE) a été introduite en 2005. Depuis, une nouvelle campagne d'évaluation a lieu chaque année avec différentes redéfinitions de la tâche. La détection d'implication textuelle est utile aux systèmes de recherche d'information au sens où un document est pertinent s'il implique la phrase utilisée comme requête. Pour les systèmes de question-réponse, un texte est une réponse à une question si la clôture existentielle de la représentation de la question est impliquée par la représentation de ce texte. Et pour la synthèse de document, elle permet de vérifier que la synthèse est bien impliquée par le texte et d'éviter les redondances entre phrases (une phrase impliquée par une autre phrase présente dans le résumé ne sera pas incluse).

L'objectif de cette tâche est de modéliser la capacité humaine à savoir si une hypothèse peut être déduite à partir d'un texte. La campagne RTE propose une suite de problèmes composés d'un texte, d'une hypothèse et d'une annotation permettant de savoir si le texte implique l'hypothèse. Les problèmes (1) et (2) montrent des exemples d'implication et de non-implication. Il est important de noter que l'implication textuelle n'est pas équivalente à l'implication logique. En effet, pour qu'il y ait implication textuelle i.e., pour qu'un être humain juge qu'un texte en implique un autre, il suffit, d'après la définition du RTE, que la déduction soit fortement

probable. Ainsi, bien que le problème (3) soit annoté comme étant une implication textuelle, il est tout à fait possible d'imaginer un monde où la personne s'appelant Shapiro ait un bureau à Century City mais travaille dans un autre lieu. Dans ce cas, l'implication est textuelle mais pas logique puisqu'il existe un monde où le texte est vrai et où l'hypothèse est fausse.

(1) **T :** « The diplomate leaves Paris. »
H : « The diplomate was in Paris. »
Implication : OUI

(2) **T :** « John loves Mary. »
H : « Mary loves John. »
Implication : NON

(3) **T :** « About two weeks before the trial started, I was in Shapiro's office in Century City. »
H : « Shapiro works in Century City. »
Implication : OUI

Pour détecter les implications textuelles, il est nécessaire de comprendre précisément le sens du texte et de l'hypothèse et de pouvoir dire si l'implication est probable étant donnée une connaissance générale du monde et de la langue. L'un des problèmes est que la langue est ambigüe à de multiples niveaux (lexical, syntaxique, sémantique et pragmatique) et qu'il est très difficile pour un système formel de savoir quel choix prendre pour chaque ambigüité. Par exemple, selon le contexte situationnel, la phrase « la petite brise la glace » impliquera soit l'hypothèse « la petite fille casse la glace », soit l'hypothèse « le petit vent lui donne froid ». Afin de résoudre une telle ambigüité, il faudrait pouvoir à la fois détecter l'ambigüité sémantique du texte et la résoudre en prenant en compte le contexte d'énonciation.

Par ailleurs, la distinction entre connaissance générale et spécialisée ainsi que la probabilité qu'une implication tienne dépend de facteurs individuels. Par exemple, le sens informatique du mot « navigateur » sera considéré comme de la connaissance générale par toute personne utilisant internet alors qu'il relèvera très probablement de la connaissance technique pour toute personne n'ayant jamais utilisé un ordinateur. Au final, nombre d'implications textuelles ne peuvent pas avoir une réponse tranchée. Il est donc impossible de définir un système permettant de trouver la bonne réponse à tous les coups, mais il est possible d'avoir un système répondant juste dans certains cas et donnant une probabilité (pouvant représenter la probabilité qu'une personne prise au hasard donne cette réponse) que sa réponse soit juste pour les autres cas.

L'approche que nous proposons vise à traiter une version restreinte de la tâche de détection d'implications textuelles : elle ne prend pas en compte la probabilité d'une implication et traite l'implication textuelle comme une implication logique ; les

connaissances utilisées sont limitées à des connaissances linguistiques (e.g., il n'y a pas de connaissances comme « John Lennon est né en 1940 » ou « Tout homme a un père et une mère ») ; enfin, nous traitons de l'implication entre phrases et non entre textes. En restreignant ainsi la problématique, nous visons non pas à développer un système capable de traiter de textes tout venant mais plutôt, à mieux comprendre les difficultés posées par le système linguistique et en particulier, par le pouvoir paraphrastique de la langue. Pour ces cas où les connaissances du monde n'entrent pas en jeu, l'objectif de cette thèse est d'examiner dans quelle mesure il est possible de détecter différentes façons de verbaliser des contenus proches et en particulier, différentes façons de verbaliser des contenus entrant dans une relation d'implication.

Contributions

Les contributions apportées par cette thèse portent sur trois grands points : le développement d'un système de reconnaissance d'implication textuelle (modulo les restrictions mentionnées ci-dessus) ; la spécification et l'implémentation d'un système de génération de problèmes d'implication textuelle annotés ; et la définition d'une nouvelle méthodologie d'évaluation.

Le système de détection d'implications textuelles (Afazio) que nous présentons, se base sur la réécriture et se limite au traitement des implications fondées sur les connaissances linguistiques. Il est composé de trois modules de réécriture appliqués en cascade sur les analyses syntaxiques de l'analyseur de Stanford avec pour résultat final des formules de logique du premier ordre donnant une représentation "normalisée" du contenu sémantique des phrases analysées. Les représentations sont normalisées au sens où elles visent à faire abstraction des différences de surface et en particulier, des différences de réalisations syntaxiques (e.g., active/passive) et morpho-syntaxiques (e.g., verbe/nom déverbal). Ces formules sont utilisées pour tester l'implication logique entre deux textes à l'aide d'outils de preuve. Les deux premiers modules de réécriture sont des étiqueteurs de rôles sémantiques symboliques pour les propositions verbales et nominales développés à partir de ressources linguistiques existantes. Le troisième module de réécriture permet de dériver des formules logiques à partir des structures syntaxiques étiquetées par les deux modules de réécriture précédents. Il présente une alternative aux systèmes de calcul sémantique basés sur le lambda calcul, alternative motivée par des considérations pratiques plutôt que théoriques : la réécriture permet de simplifier le calcul sémantique, un avantage non négligeable pour le développement de systèmes traitant de textes tout venant.

Une deuxième contribution concerne la spécification et la mise en œuvre d'un système de génération de suites de tests annotés pour la DIT. Ce système permet de générer un ensemble de problèmes à partir d'un ensemble de formules sémantiques. Parce qu'il met en jeu un réalisateur de surface basé sur une grammaire symbolique,

ce système permet d'annoter automatiquement chaque problème avec une information sémantique (y a-t-il ou non une relation d'implication entre texte et hypothèse ?) et une information syntaxique fine (quelles sont les différences/ressemblances syntaxiques entre texte et hypothèse ?). En outre, nous avons développé un algorithme permettant de sélectionner un échantillon d'une suite de tests de manière à satisfaire au mieux un ensemble de contraintes. Ainsi, le système permet de générer des suites de tests contrôlées et équilibrées i.e., des suites de test dans lesquelles les différents phénomènes et combinaisons de phénomènes possibles sont représentés le plus uniformément possible.

Enfin, nous avons adapté à la DIT des techniques de fouille d'erreurs développées dans le domaine de l'analyse syntaxique permettant ainsi d'avoir des résultats plus pertinents et de mieux comprendre les avantages et les faiblesses des différentes approches. Nous avons utilisé ces techniques pour évaluer les systèmes existants et comparer Afazio, le système que nous avons développé, avec Nutcracker, un autre système hybride qui, comme Afazio, combine analyse syntaxique stochastique et raisonneurs automatiques pour détecter l'implication textuelle.

Organisation

Le chapitre 1 présente et situe la campagne d'évaluation RTE. Nous y explicitons la notion d'implication couverte par cette campagne, la méthode de création des données adoptée et le protocole d'évaluation utilisé. Nous comparons ensuite les données du RTE avec d'autres ensembles de données utilisés pour tester les capacités de raisonnement de systèmes de TAL à savoir, les suites de tests FraCaS et AQUAINT. Enfin, nous identifions un certain nombre de critiques qui peuvent être formulées à l'égard de cette campagne.

Le chapitre 2 décrit la structure générale de notre système de détection d'implications textuelles, Afazio. Son fonctionnement ressemble un peu à celui d'un prouveur de théorème au sens où il décompose la tâche de détection en deux phases : la première consistant à normaliser la représentation d'un texte, et la seconde visant à vérifier si l'implication est vraie ou non. Afazio est un système hybride combinant un analyseur syntaxique stochastique avec des modules symboliques de réécriture et d'inférence logique. Nous expliquons comment Afazio normalise les analyses syntaxiques en utilisant plusieurs systèmes de réécriture permettant d'abord, de gérer les variations verbales et nominales puis de dériver les formules logiques correspondantes. Ensuite nous présentons le procédé de vérification permettant de savoir si le Texte implique l'Hypothèse ou non. Enfin, nous commentons l'utilisation de la réécriture dans les systèmes de RTE.

Dans le chapitre 3, nous argumentons en faveur de la construction de suites de tests complémentaires à celle du RTE. Alors que le RTE vise à évaluer la capacité des

systèmes à détecter l'implication textuelle sur des données produites dans le cadre d'applications données (systèmes de question-réponse, de synthèse de document, de recherche et d'extraction d'information), les suites de tests que nous proposons visent à permettre une évaluation analytique des systèmes à traiter certains types d'implications comme les implications basées sur la syntaxe, la sémantique lexicale ou la sémantique compositionnelle. Nous commençons par motiver cette nouvelle approche puis nous décrivons le processus de construction des suites de tests. Enfin, nous proposons un algorithme permettant de sélectionner un échantillon d'une suite de tests satisfaisant au mieux un certain nombre de contraintes.

Dans le chapitre 4, nous commençons par décrire les structures qu'Afazio utilise ainsi que la nomenclature graphique que nous avons défini pour les représenter. Ensuite, nous présentons la méthodologie utilisée pour obtenir les règles de réécriture permettant d'étiqueter sémantiquement les propositions verbales sur les structures syntaxiques. Nous avons dans un premier temps développé des règles manuellement puis nous avons mis en place une méthodologie permettant de dériver semi-automatiquement de nouvelles règles à partir des règles définies manuellement. De même, nous décrivons comment de nouvelles règles de réécriture peuvent être dérivées pour étiqueter sémantiquement les propositions nominales à partir de la ressource linguistique NomLexPlus. Enfin, nous détaillons les différents traitements appliqués par Afazio aux structures syntaxiques étiquetées, pour dériver les arbre syntaxiques des formules de logique du premier ordre leur correspondant.

Enfin, le chapitre 5 présente une technique de recherche d'erreur que nous avons adapté à la DIT et explique en quoi, celle-ci permet d'obtenir de meilleurs résultats que les calculs d'exactitude [1]. Nous appliquons cette technique au corpus de la seconde campagne du RTE, sur laquelle des annotations supplémentaires et des évaluations ont précédemment été réalisées par [Garoufi, 2007] ; nous comparons nos résultats à ceux existant et discutons des différences observées. Enfin, nous appliquons cette nouvelle méthode d'évaluation à Afazio et à Nutcracker sur les suites de tests que nous avons générées permettant ainsi de mieux comprendre les différences entre les deux systèmes.

La conclusion résume les principaux résultats obtenus et identifie un certain nombre de pistes de recherche naturellement évoquées par le travail réalisé pour cette thèse.

1. L'exactitude (accuracy) est le pourcentage de problèmes de DIT bien reconnus

Chapitre 1

Reconnaître l'implication textuelle

Ce chapitre présente et situe la campagne d'évaluation RTE (Recognising Textual Entailment / Détection d'Implication Textuelle). Nous y explicitons la notion d'implication couverte par cette campagne, la méthode de création des données adoptée et le protocole d'évaluation utilisé. Nous comparons ensuite les données du RTE avec d'autres ensembles de données utilisés pour tester les capacités de raisonnement de systèmes de TAL à savoir, les suites de tests FraCaS et AQUAINT. Pour finir, nous identifions un certain nombre de critiques qui peuvent être formulées à l'égard de cette campagne.

1.1 La campagne RTE

1.1.1 La suite de tests

L'objectif du RTE est de tester les systèmes de TAL sur leur capacité à détecter l'implication textuelle. Il s'agira par exemple, de détecter que dans l'exemple (4), la première phrase (i.e., le texte) implique la seconde (i.e., l'hypothèse) alors que dans l'exemple (5), la première phrase n'implique pas la seconde.

(4) **T :** « The diplomate leaves Paris. »
H : « The diplomate was in Paris. »
Implication : Oui

(5) **T :** « John loves Mary. »
H : « Mary loves John. »
Implication : Non

Les critères de validité d'une implication textuelle sont les suivants :

– L'implication étant une relation dirigée, l'hypothèse doit être impliquée par le texte, mais le texte n'a pas besoin d'être impliqué par l'hypothèse.

- L'hypothèse doit être complètement impliquée par le texte. Si elle comporte une partie ne pouvant être inférée à partir du texte, alors l'implication textuelle est fausse.
- Les cas où l'implication est très probable (mais pas complètement certaine) sont considérés comme vrais.
- Une connaissance du monde est présumée (e.g., la capitale d'un pays est situé dans ce pays, le président d'un état est aussi un citoyen de cet état et ainsi de suite).

L'implication textuelle ressemble donc à une implication logique avec des axiomes pour représenter la connaissance du monde. Le troisième critère dénote cependant une différence importante, celle de la probabilité d'une implication. Dans l'exemple (6), un système logique obtiendrait que l'implication est fausse car bien que le bureau de Shapiro soit à Century City, celui-ci peut travailler actuellement dans un autre lieu. Cependant, cette interprétation est très improbable, et donc l'implication textuelle est considérée comme étant vraie.

(6) **T :** « About two weeks before the trial started, I was in Shapiro's office in Century City. »
H : « Shapiro works in Century City. »
Implication : OUI

Pour permettre le développement de systèmes capables de traiter de textes tout venant, les données d'évaluation sont construites à partir de textes existants par le biais d'applications permettant de traiter les tâches de question-réponse (QR), de recherche d'information (RI), d'extraction d'information (EI), et de synthèse de document (SD).

La méthode d'acquisition des problèmes est approximativement la même pour les différentes tâches. Elle consiste à reprendre des données d'évaluations correspondantes à la tâche, et à transformer les données ainsi obtenues en une suite de couples de textes $< T, H >$ annotés de manière à savoir si T implique textuellement H ou non. Les textes sont corrigés manuellement, afin d'éliminer les fautes d'orthographe et de ponctuation.

Pour la tâche de recherche d'information, les hypothèses proviennent des évaluations TREC et CLEF [2] et sont adaptées et simplifiées à partir des requêtes. Les textes qui impliquent ou non l'hypothèse sont ensuite sélectionnés à partir des documents retrouvés par les différents moteur de recherche (e.g. Google et Yahoo) pour chaque hypothèse.

2. Dans ces évaluations, des requêtes sont données aux systèmes qui pour chacune d'elles renvoient une liste de documents devant satisfaire au mieux la requête. Les réponses sont ensuite évaluées manuellement par des experts.

Pour la tâche de question-réponse, des questions sont sélectionnées à partir des compétitions de QR, tel que les suites de tests TREC-QA et QA@CLEF. Les réponses correspondantes sont extraites d'internet à l'aide de systèmes de QR. Les couples question-réponse sont ensuite transformés en couple texte-hypothèse comme suit :

- une réponse, correcte ou non, du type de réponse attendue est extraite du passage de texte sélectionné par le système,
- la question est tournée en une phrase affirmative en y insérant la réponse choisie,
- le couple texte-hypothèse est généré, en utilisant la phrase affirmative comme hypothèse et le passage de texte comme texte.

Par exemple, étant donné la question « Who has written "The Persians" ? », et le texte donné pour la tâche QR dans la table 1.1, l'annotateur transformera la question en « "The Persians" was written by Aeschylus. », générant ainsi le couple positif QR de la table 1.1.

L'annotation permettant de savoir si le texte implique l'hypothèse est réalisée par un minimum de trois annotateurs (seulement deux pour l'ensemble de développement permettant d'entrainer les systèmes). Les problèmes pour lesquels les annotateurs ne sont pas unanimes sont éliminés. L'accord moyen sur l'ensemble de test du RTE1 (entre chaque paire d'annotateurs partageant au moins 100 problèmes), est d'environ 0,8 avec un niveau Kappa[Cohen, 1960] moyen de 0,6 correspondant à un accord inter-annotateurs *modéré*. Un des organisateurs a passé en revue les problèmes restant et en a supprimé 13% de plus. Au total 33% des problèmes de l'ensemble de test ont été écartés à cause de désaccord entre les annotateurs. Un des participants [Bos & Markert, 2005] a effectué une autre annotation manuelle du corpus, et a obtenu un accord de 95,25% avec les annotations de base. On peut se rendre compte que la détection d'implications textuelles n'est pas un tâche facile, même pour un humain. Les désaccords de ceux-ci sont dus à l'ambigüité de l'implication quant à la probabilité qu'elle ait lieu et la connaissance nécessaire pour qu'elle soit vraie.

La campagne RTE évoluant avec les années, la tâche et les données à traiter ont évolué d'une année à l'autre.

Le RTE4 [Giampiccolo *et al.*, 2008] a modifié l'annotation permettant de savoir si l'implication textuelle est vrai ou fausse. Les fausses implications textuelles ont été décomposées en deux catégories. La première contenant les problèmes où l'hypothèse contredit le texte comme le problème (7) où il y a une contradiction car « Jennifer Hawkins » ne peut pas avoir 20 et 21 ans en même temps. L'autre contenant le reste des problèmes comme celui de l'exemple (8) où le texte n'implique pas textuellement l'hypothèse et n'est pas non plus contredit par celle-ci (dans cet exemple la non-implication est due à l'ajout d'information dans l'hypothèse).

Tâche	Texte	Hypothèse	implication
EI	At the same time the Italian digital rights group, Electronic Frontiers Italy, has asked the nation's government to investigate Sony over its use of anti-piracy software.	Italy's government investigates Sony.	Oui
EI	Parviz Davudi was representing Iran at a meeting of the Shanghai Co-operation Organisation (SCO), the fledgling association that binds Russia, China and four former Soviet republics of central Asia together to fight terrorism.	China is a member of SCO.	Oui
RI	Between March and June, scientific observers say, up to 300,000 seals are killed. In Canada, seal-hunting means jobs, but opponents say it is vicious and endangers the species, also threatened by global warming.	Hunting endangers seal species.	Oui
RI	The Italian parliament may approve a draft law allowing descendants of the exiled royal family to return home. The family was banished after the Second World War because of the King's collusion with the fascist regime, but moves were introduced this year to allow their return.	Italian royal family returns home.	Non
QR	Aeschylus is often called the father of Greek tragedy ; he wrote the earliest complete plays which survive from ancient Greece. He is known to have written more than 90 plays, though only seven survive. The most famous of these are the trilogy known as Orestia. Also wellknown are The Persians and Prometheus Bound.	"The Persians" was written by Aeschylus.	Oui
SD	A Pentagon committee and the congressionally chartered Iraq Study Group have been preparing reports for Bush, and Iran has asked the presidents of Iraq and Syria to meet in Tehran.	Bush will meet the presidents of Iraq and Syria in Tehran.	Non

Table 1.1 – Exemples de couples d'implication textuelle provenant du RTE3

(7) **T :** « Jennifer Hawkins is the 21-year-old beauty queen from Australia. »
H : « Jennifer Hawkins is Australia's 20-year-old beauty queen. »
Résultat : CONTREDIT

(8) **T :** « Five people were killed in another suicide bomb blast at a police station in the northern city of Mosul. »
H : « Five people were killed and 20 others wounded in a car bomb explosion outside an Iraqi police station south of Baghdad. »
Résultat : INCONNU

Une autre modification faite lors du RTE4 fut de demander aux systèmes une justification du choix effectué. Par exemple, si un système affirme qu'il y a une contradiction dans le problème (7), il devra dire que cela est du au fait qu'une personne ne peut pas avoir 20 et 21 ans en même temps. Cette modification permet de mieux comprendre pourquoi les systèmes se trompent et de vérifier qu'ils répondent juste pour le bon motif.

Au cours des différentes campagnes, des textes plus longs ont été incorporés afin d'avoir des couples nécessitant une analyse du discours pour obtenir la bonne réponse. Par exemple, dans l'exemple (9) il faut que le système arrive à faire la liaison anaphorique entre « she » et « Dr Fiona Wood » pour pouvoir déduire que l'implication est vraie.

(9) **T :** « She has become world renowned for her patented invention of spray on skin for burns victims, a treatment which is continually developing. Via her research, Fiona found that scarring is greatly reduced if replacement skin could be provided within 10 days. As a burns specialist the holy grail for Dr Fiona Wood is 'scarless, woundless healing'. »
H : « Dr Fiona Wood has invented a treatment for burns victims. »
Implication : OUI

Afin de tester les systèmes dans des conditions plus réelles, le RTE5 [Bentivogli *et al.*, 2009] a choisi de ne plus corriger les textes obtenus lors de la création des problèmes, afin d'évaluer si les systèmes arrivent à gérer des erreurs comme les phrases agrammaticales, ou des erreurs typographiques.

Les systèmes étant généralement composés de plusieurs modules, le RTE5 [Bentivogli *et al.*, 2009] propose de faire des tests d'ablation afin de savoir quels modules sont importants ou redondants pour la détection d'implications textuelles. Par exemple un système utilisant des ressources lexicales et du chevauchement de mots, sera testé sur le corpus avec les deux modules activés puis chacun d'eux séparément. Cela permettra en même temps de donner des informations sur le type de ressources nécessaires pour répondre juste à un problème.

La dernière modification a pour but de rendre le campagne encore plus applicative. Pour cela le RTE6 propose d'étendre les couples texte-hypothèse en des couples

textes-hypothèse où l'on a plusieurs textes associés à une hypothèse. Les systèmes doivent alors identifier quels textes impliquent l'hypothèse parmi ceux proposés. Cela revient donc à une sorte d'évaluation de systèmes de recherche d'information où l'on demanderait à des systèmes de trouver des documents sur un thème précis, sauf que dans ce cas le corpus de document est nettement plus petit et que la recherche doit satisfaire des critères plus précis (i.e., elle ne doit pas juste être sur le même thème mais doit permettre d'inférer l'hypothèse).

1.1.2 L'évaluation

Le campagne RTE a réussi à rassembler un grand nombre de groupes de recherche autour de la tâche de reconnaissance d'implications textuelles. Au cours des différentes campagnes, entre 16 et 26 groupes se sont prêtés à l'exercice. Hickl *et al.* [2006] regroupe les différentes approches mises en œuvre en 4 catégories :

1. les systèmes qui extraient des informations linguistiques des couples texte-hypothèse, et transforment le problème de reconnaissance d'implication en un problème de classification,
2. ceux qui évaluent la *probabilité* qu'une implication puisse exister entre la couple texte-hypothèse,
3. ceux qui représentent la connaissance du couple texte-hypothèse dans un langage de représentation pouvant être associé avec un système d'inférence,
4. et enfin ceux qui utilisent la définition classique dans le domaine de l'intelligence artificielle de l'implication, et construisent des modèles du monde dans lesquels l'hypothèse et le texte sont vrais, et vérifie si le modèle associé à l'hypothèse est inclus dans le modèle associé au texte.

Une autre classification des systèmes peut être faite par rapport aux techniques qu'ils utilisent pour détecter l'implication textuelle. Ces techniques sont entre autre la mesure de chevauchement de mots, l'alignement d'arbres syntaxiques ou de graphes de dépendances, l'apprentissage automatique, l'inférence logique, l'étiquetage de rôles sémantiques. Ces techniques sont combinées avec l'utilisation de ressources de connaissances lexicales, syntaxiques ou du monde (e.g., WordNet, Wikipedia, DIRT, PropBank ou NomBank) déjà existantes ou créées spécialement pour la campagne.

Les résultats obtenus se sont améliorés au fur et à mesure des campagnes. Le niveau de référence (baseline) en terme d'exactitude est de 50% pour un système qui répondrait toujours Vrai ou toujours Faux. Lors de la première campagne, les systèmes ont obtenu des exactitudes allant de 50% à 60%, qui sont passées de 53% à 75% lors de la seconde campagne pour atteindre les meilleurs résultats allant de 49% à 80% pour la troisième édition du challenge. Le RTE4 proposant aux systèmes de répondre par { Implique, Inconnu, Contredit } avec une répartition de (50%,

35%, 15%) plutôt que par { Oui, Non } (50%,50%), le niveau des systèmes est descendu atteignant des exactitudes allant de 30,7% à 68,5%. Dans la section 5, nous examinerons une évaluation plus approfondie des résultats du RTE2.

1.2 Les autres suites de test existantes

1.2.1 Suite de tests FraCaS

L'analyse de la sémantique d'un texte est une tâche essentielle du TAL, et nombre de chercheurs se sont intéressés à ce problème. Parmi ces chercheurs, le projet FraCaS (Framework for Computational Semantics [The Fracas Consortium *et al.*, 1996]), s'est attardé sur l'aspect logiciel de cette tâche. Ils ont donc proposé un ensemble de logiciels permettant entre autre le développement de grammaires produisant des sorties sémantiques, la visualisation de certains formalismes sémantiques comme les DRT et l'analyse de textes afin d'obtenir les formules sémantiques correspondantes.

Lors de la création de ces outils, le consortium FraCaS s'est posé la question de comment évaluer adéquatement les compétences sémantiques d'un système de TAL. Pour cela, il a décidé de développer une suite de tests car cela lui permettait en même temps de réfléchir aux différentes capacités sémantiques qu'un système de TAL doit avoir. Plusieurs choix étaient possibles pour le type de problème composant la suite de tests. Il aurait pu par exemple donner des couples (texte, sémantique) et les systèmes auraient été évalués sur la justesse de leur sémantique. Le problème de cette approche est qu'elle implique de choisir un formalisme et qu'elle est coûteuse à mettre en place. Le consortium FraCas a préféré choisir la capacité d'inférence comme évaluation car c'est un fondement des compétences sémantiques, et qu'elle permet d'évaluer des systèmes utilisant des formalismes différents.

Les éléments de la suite de tests sont des sortes de syllogismes représentés sous la forme de couples $< P, Q >$ où P est une liste de phrases représentant les prémisses et Q est une question représentant la conclusion. P est composée d'une ou plusieurs phrases qui sont pertinentes par rapport à la question Q. Les questions sont totales [3]. On a donc une évaluation similaire à l'évaluation du RTE incluant la contradiction. Il existe une version modifiée de la suite de tests [4] réalisée par Bill MacCartney où les questions ont été transformées en hypothèses afin de pouvoir évaluer les systèmes du RTE. Dans l'exemple 022 de la figure 1.1, on a une variation de monotonicité ascendante [5] sur « the report on time » qui est dans un contexte négatif à cause de

3. Pour les questions totales il est juste nécessaire d'infirmer/confirmer la proposition ou de dire que l'on ne connait pas la réponse en répondant par « oui »/« non »/« je ne sais pas », alors que pour les questions partielles il est nécessaire de donner une information non contenue dans la question (e.g., « Qui a découvert l'Amérique ? », « Christophe Colomb »)

4. `http://www-nlp.stanford.edu/~wcmac/downloads/fracas.xml`

5. Un contexte avec une monotonicité descendante implique tout ce qui est plus spécifique alors qu'un contexte avec une monotonicité ascendante implique tout ce qui est plus général.

– Quantificateurs :
  – Conservativité :

001	P1	An Italian became the world's greatest tenor.
	Q	Was there an Italian who became the world's greatest tenor ?
	H	There was an Italian who became the world's greatest tenor.
		[Oui]
014	P1	Neither leading tenor comes cheap.
	P2	One of the leading tenors is Pavarotti.
	Q	Is Pavarotti a leading tenor who comes cheap ?
	H	Pavarotti is a leading tenor who comes cheap.
		[Non]

  – Monotonicité (ascendante sur le second argument) :

022	P1	No delegate finished the report on time.
	Q	Did no delegate finish the report ?
	H	No delegate finished the report.
		[Inconnue]

– Anaphore :
  – Intra-Phrasique :

114	P1	Mary used her workstation.
	Q	was Mary's workstation used ?
	H	Mary's workstation was used.
		[Oui]

  – Inter-Phrasique :

120	P1	Smith attended a meeting.
	P2	She chaired it.
	Q	Did Smith chair a meeting ?
	H	Smith chaired a meeting.
		[Oui]

FIGURE 1.1 – Exemples de la suite de tests FraCaS

« No ». La réponse à la question Q est donc Inconnue car elle ne peut être ni Oui, ni Non.

La suite de tests est composée de 346 problèmes composés en moyenne de 1,55 prémisses. Elle est décomposée en 9 catégories représentant les différents types de phénomènes linguistiques et sémantiques. Ces catégories sont elles-mêmes divisées en sous-catégories afin de savoir exactement quel phénomène est évalué. Par exemple la catégorie Quantificateur est composée des sous-catégories Conservativité, Monotonicité ascendante sur le second argument, Monotonicité descendante sur le second argument, Monotonicité montante sur le premier argument et Monotonicité descendante sur le premier argument. Dans la table 1.2, on peut voir que les problèmes ne sont pas répartis équitablement dans ces catégories. Cela est normal car le but de cette suite de tests est d'avoir des problèmes nécessitant des raisonnements logiques différents, et certains phénomènes (i.e., les Quantificateurs) ont beaucoup plus de va-

type	#	%	prémisse unique
Quantificateur	80	23	50
Pluriel	33	10	24
Anaphore	28	8	6
Ellipse	55	16	25
Adjectif	23	7	15
Comparatif	31	9	16
Temporel	75	22	39
Verbe	8	2	8
Attitude	13	4	9

TABLE 1.2 – Répartition des catégories de FraCaS

riations de raisonnement que d'autres (i.e., les Verbes). Il n'y a pas dans cette suite de tests, deux problèmes nécessitant le même raisonnement mais variant uniquement sur le lexique et la syntaxe.

Cette suite de tests est très différente du RTE dans sa conception. Elle est nettement moins applicative, et utilise des textes artificiels. Elle est néanmoins intéressante et complémentaire à l'approche du RTE. Tout d'abord, la suite de tests FraCaS évalue les capacités des systèmes à gérer des phénomènes linguistiques et sémantiques et non la capacité à pouvoir trouver des solutions à un certain type de problème applicatif. De plus, dans cette suite de tests, chaque problème met en œuvre un phénomène unique, ce qui permet de savoir exactement ce qui pose problème aux différents systèmes. Il aurait pu être intéressant d'avoir en plus des catégories disjointes des croisements entre ces catégories afin de savoir si la combinaison de certains phénomènes pose problème, mais il aurait fallu créer beaucoup plus de problèmes, qui auraient été plus complexes que ceux de base et cela aurait été très couteux, la suite de tests étant créée entièrement manuellement.

1.2.2 Suite de tests AQUAINT

La recherche sur les systèmes de question-réponse, est une tâche complexe à cause de la quantité importante d'informations disponibles mais aussi de la difficulté à modéliser le contenu de la question et des documents dans lesquels on recherche des réponses afin de pouvoir extraire la bonne. Le projet AQUAINT s'est intéressé à ce problème et l'a décomposé en plusieurs parties. Celle qui nous intéresse est la partie inférence où le système teste différents fragments de texte afin de savoir s'ils impliquent la réponse.

Afin d'évaluer leur moteur d'inférence, le projet AQUAINT a développé une suite de tests [Crouch *et al.*, 2005] telle que chaque élément de cette suite est composé des

éléments suivants :

- Passage : texte
- Question : question à propos du texte
- Réponse : réponse à la question
- Polarité :
  - vraie : si la réponse est juste
  - fausse : si la réponse est fausse et peut être contredite à partir du passage
  - inconnue : si on ne peut pas savoir si la réponse est vrai ou fausse
- Force :
  - stricte : si la polarité peut être déduite inéluctablement
  - plausible : si la polarité choisie peut changer selon le contexte
- Source :
  - linguistique : si uniquement des connaissances linguistiques sont nécessaires pour savoir si la réponse est juste ou fausse
  - monde : s'il est nécessaire d'avoir une connaissance du monde pour savoir si la réponse est juste ou fausse
- Cause (optionnelle) : connaissances nécessaires pour justifier la réponse
- Contexte (optionnel) : contexte dans lequel la polarité change

Cette suite de tests est intéressante car elle pose la question de l'ambigüité de l'implication par rapport à la connaissance que le système a du monde. Elle permet donc de répondre au problème d'accord entre les annotateurs. Les exemples 2, 3 et 4 de la figure 1.2 montrent un cas d'implication textuelle où la réponse ainsi que la plausibilité de celle-ci peut changer en fonction des connaissances disponibles. L'exemple 4 n'utilise aucune connaissance autre que syntaxique et conclut donc que la réponse est Inconnue. L'exemple 2, utilise une connaissance générale du monde qui lui permet d'avoir une réponse Fausse mais pour laquelle cette réponse n'est pas sure à 100%. Enfin, l'exemple 3 utilise une connaissance spécialisée du monde, qui permet d'être sûr que la réponse est Fausse. Soulignons que la Cause ne doit pas permettre de déduire la réponse sans le passage, c'est pourquoi dans l'exemple 3 on n'a pas comme Cause « La reine Victoria est née en 1819 ».

Bien qu'intéressante, cette suite de tests est difficilement utilisable car les problèmes sont pour la plupart ambigus. Sachant, que l'annotation est ambigüe, comment dire si un système a bien répondu ou non ? On pourrait imaginer que les systèmes sélectionnent le type de réponse à laquelle ils veulent se comparer, mais il faudrait alors une annotation plus précise permettant par exemple de différencier la connaissance du monde de l'exemple 2, qui est générale de celle de l'exemple 3 qui est spécialisée.

1. PASSAGE : Queen Victoria died in 1901, at Osborne House on the Isle of Wight.
   QUESTION : Who died in 1901 ?
   RÉPONSE : Queen Victoria
   POLARITÉ : Vraie
   FORCE : Stricte
   SOURCE : Linguistique

2. PASSAGE : Queen Victoria died in 1901, at Osborne House on the Isle of Wight.
   QUESTION : Who was born in 1901 ?
   RÉPONSE : Queen Victoria
   POLARITÉ : Fausse
   FORCE : Plausible
   SOURCE : Monde
   CAUSE : La plupart des gens, et plus spécialement les personnes couronnées, ne meurent pas le jour où elles sont nées

3. PASSAGE : Queen Victoria died in 1901, at Osborne House on the Isle of Wight.
   QUESTION : Who was born in 1901 ?
   RÉPONSE : Queen Victoria
   POLARITÉ : Fausse
   FORCE : Stricte
   SOURCE : Monde
   CAUSE : La reine Victoria est morte à l'age de 81 ans

4. PASSAGE : Queen Victoria died in 1901, at Osborne House on the Isle of Wight.
   QUESTION : Who was born in 1901 ?
   RÉPONSE : Queen Victoria
   POLARITÉ : Inconnue
   FORCE : Stricte
   SOURCE : Linguistique

5. PASSAGE : Congressman Smith visited Baghdad.
   QUESTION : Has Smith been to Iraq ?
   RÉPONSE : Oui
   POLARITÉ : Vraie
   FORCE : Stricte
   SOURCE : Monde
   CAUSE : Baghdad est en Iraq.

FIGURE 1.2 – Exemples provenant de la suite de test AQUAINT

1.3 Critiques du RTE et propositions d'alternatives

Chevauchement lexical Un des défauts qu'une suite de tests peut avoir, est de contenir des indices sur les solutions aux différents problèmes n'existant pas normalement. Ce défaut touche la suite de tests du RTE via la corrélation entre la réponse et le score de chevauchement de mots. Pour obtenir cette corrélation on calcule tout d'abord le score de chevauchement de mots à l'aide de la formule :

$$CM = \frac{\#\ lemmes\ en\ commun\ de\ T\ et\ H}{\#\ lemmes\ de\ H}$$

puis on calcule la corrélation :

$$Correlation = \frac{\sum_{couple} \begin{cases} CM & \text{si Implication} = \textsc{Oui} \\ 1 - CM & \text{si Implication} = \textsc{Non} \end{cases}}{\#\ couples}$$

Cette corrélation a été calculée pour les données du RTE2 par [Garoufi, 2007], qui a obtenu un chevauchement de mots de 71,24% en moyenne et une corrélation de 0,56. Cela signifie que plus le chevauchement de mots est important plus l'implication à de chance d'être Vraie, ce qui est inexact vu qu'une négation, ou un quantificateur peut inverser le résultat en modifiant légèrement le chevauchement. Un autre indicateur montrant que la corrélation entre chevauchement de mots et résultat n'a pas bien été réglée dans la campagne du RTE2, que [Zanzotto *et al.*, 2006] a mis en évidence, est qu'un système utilisant seulement l'information sur le chevauchement de mots pouvait atteindre une exactitude de 60% [6].

Comme nous allons le voir, ce problème de corrélation est en grande partie du au fait que la suite de tests du RTE2 a une mauvaise répartition des différents phénomènes linguistiques et sémantiques qu'une implication peut mettre en œuvre.

Annotation Comme nous l'avons vu précédemment, la campagne RTE a pour but d'évaluer les systèmes sur leur capacité à reconnaître des implications textuelles provenant de diverses applications. À cette fin, les problèmes sont annotés afin de savoir si l'implication existe, et quelle application a été utilisée pour créer le problème. Cela permet donc seulement d'évaluer les systèmes sur leurs exactitudes [7] globalement et par application. Il n'est pas possible de savoir quel phénomène pose le plus de problèmes aux systèmes. De plus, comme le fait remarquer Zaenen *et al.* [2005], un corpus n'ayant pas d'annotation sur les phénomènes mis en œuvre ne permet pas d'évaluer un système incomplet, uniquement sur les problèmes qu'il sait traiter. Les

6. Les résultats du RTE2 vont de 53% à 75% avec la plupart des systèmes se situant entre 55% et 61%

7. L'exactitude (accuracy) est le pourcentage de problèmes de DIT bien reconnus

Identité	**Lexique**	**Syntaxe**	**Discours**	**Raisonnement**
Identité	Acronyme	Apposition	Coréférence	Génitif
	Démonyme	Variation d'Arg.	Contre-factif	Modifieur
	Hypernyme	Passivation	Factif	Quantificateur
	Nominalisation	Relative réduite	Implicatif	Raisonnement
	Autres		Négation	
	Synonyme		Factif neutre	
			Anaphore nominal	
			Anaphore pronominal	

TABLE 1.3 – Liste des phénomènes linguistiques et sémantiques des problèmes positifs annotés par ARTE.

–696–Vrai–EI–
« The sources, speaking days before Scheuer's interview appeared July 4 in the [italian] daily La Republica, said those actions were routine in such cases, known as "renditions" – transfers of subjects from one country to another. »
« La Republica publishes [in Italy] »

FIGURE 1.3 – Un alignement avec un Modifieur « Italian » qui dénote une location, le marqueur est aussi une instance de Démonyme.

suites de tests FraCaS et AQUAINT vont dans le sens d'une annotation plus fournie permettant d'identifier les phénomènes qui posent le plus de problèmes aux systèmes, mais l'annotation la plus développée est ARTE [Garoufi, 2007] qui a été appliquée sur les 400 problèmes positifs du RTE2 et sur 100 problèmes négatifs.

L'annotation ARTE propose un ensemble d'annotations pour les problèmes positifs montrant les phénomènes validant l'implication et un autre ensemble d'annotations pour les problèmes négatifs indiquant ce qui invalide l'implication. Les phénomènes annotés pour les problèmes positifs sont regroupés en catégories comme le montre la table 1.3. Pour les problèmes négatifs, les phénomènes sont Contexte, Additionnel, Inadéquation et Contradiction. Les figures 1.3 à 1.11 sont des exemples d'annotations, où l'on peut voir des alignements entres les mots du texte et de l'hypothèse mettant en œuvre un phénomène.

Cette annotation permet d'évaluer les systèmes plus précisément et d'estimer les faiblesses de ces systèmes afin de connaître le type de capacité qu'il faut améliorer

–132–Vrai–RI–
« An avalanche has struck a popular skiing resort in Austria, [killing] at least 11 people. »
« Humans [died] in an avalanche »

FIGURE 1.4 – Un alignement marqué comme Variation d'Arguments, puisque les arguments des deux prédicats sont alignés avec des fonctions syntaxique différentes.

–389–Vrai–QR–
« In Rwanda there were [on average 8,000] victims [per day for about 100 days]. »
« There were [800,000] victims of the massacres in Rwanda »

FIGURE 1.5 – Un alignement marqué avec Quantificateur car il requiert un raisonnement arithmétique.

–701–Vrai–QR–
« From its original 12 member states, CERN has now grown [to 20], including a number of countries (Bulgaria, Hungary, the Czech and Slovak republics, and Poland) of the new Europe »
« CERN has [20 member states] »

FIGURE 1.6 – Un alignement marqué avec Raisonnement, puisqu'il nécessite l'analyse de l'ellipse NP « 20 ».

–631–Vrai–EI–
« Greek coastguard officials [say] they have found a body on a boat that sent out a distress call as it carried 150 would-be immigrants to italy. The identity of the dead man and the circumstances of his death are unclear. »
« Coastguard officials have found a dead man »

FIGURE 1.7 – Dans ce problème, l'information à propos de l'évènement décrit dans la première phrase ment dans le contexte introduit par de prédicat factif neutre « say ». L'implication dépend de la confiance que l'on a en la source de l'information.

–613–Vrai–IE–
« Since [it] saw the light of day in 2004, [Katamari Damacy] has gone on to become one of the biggest cult hits in the history of video games »
« Katamari Damacy was release in 2004. »

FIGURE 1.8 – Un alignement indiquant une Coréférence entre le pronom « it » et son antécédent « Katamari Damacy ».

–567–Faux–EI–
« Jose Reyes scored the winner [for] Arsenal as they ended a three-game league losing streak with a victory over battling Charton. »
« Jose Reyes scored the winner [against] Arsenal »

FIGURE 1.9 – Les prépositions « for » et « against » expriment dans cet exemple des sens diamétralement opposés qui se contredisent.

–472–Faux–EI–
« His old enemies joined calls for his release after Hariri's killing fuelled opposition to Syria »
« Hariri [was buried] in Syria »

FIGURE 1.10 – Dans ce problème, la première phrase ne contient pas d'information liée au prédicat de la seconde phrase « was buried ». Elle est donc marquée comme Additionnelle.

–533–Faux–EI–
« Winkeler, an 18-year veteran employee of the Southen Illinoisan, [is a native] of Beckemeyer, but graduated from Southern Illinois University Carbondale in 1976 with a degree in journalism. »
« Winkeler [resides] in Beckemeyer »

FIGURE 1.11 – Le prédicat de la seconde phrase aligné avec celui de la première phrase ne permet une interprétation sémantique justifiant une implication. Le mauvais alignement est marqué comme Inadéquat.

ou ajouter. L'évaluation peut être faite au niveau des annotations ou sur les groupes d'annotations. Dans l'amélioration du RTE consistant à demander une justification, on pourrait imaginer que l'annotation respecte le schéma ARTE afin de savoir si le système a la bonne réponse pour la bonne raison.

Distribution des différents types d'implications À partir de son annotation des données RTE2, Garoufi [2007] a extrait des statistiques sur la distribution des différents phénomènes. La figure 1.12 permet de voir que cette distribution n'est pas du tout équilibrée. On peut voir que des phénomènes sémantiques permettant d'avoir des non-implications avec un chevauchement de mots élevé, comme les factifs ou la négation ne sont pratiquement pas représentés. Les phénomènes les plus présents sont l'Identité, et le Raisonnement (qui représentent tous les phénomènes non représentés par les autres étiquettes comme la connaissance générale du monde, la modalité, ou la métonymie), qui sont respectivement le phénomène le plus simple et le plus complexe (voir section 5.2). Une telle distribution ne permet pas une bonne analyse des résultats, car lorsque Raisonnement est présent dans un problème, il y a de fortes chances que les systèmes se trompent à cause de ce phénomène plutôt que sur un autre phénomène présent dans le problème. Les autres phénomènes auront donc des résultats biaisés par le Raisonnement.

La figure 1.13 montre la distribution des combinaisons de catégories de phénomènes de manière exclusive. Ainsi, les valeurs représentent le nombre de couples ayant exactement cette combinaison de phénomènes. Cet histogramme permet de voir qu'il y a très peu de problèmes impliquant une seule catégorie de phénomène, et que la répartition des phénomènes n'est pas équilibrée. Il est donc difficile d'identifier clairement quelle catégorie de phénomène est mal gérée par un système.

1.4 Conclusion

Dans ce chapitre nous avons donc vu en quoi consiste la tâche de détection d'implications textuelles, et décrit le challenge RTE, qui représente le plus grand rassemblement sur ce domaine, ainsi que les améliorations qui y ont été apportées au

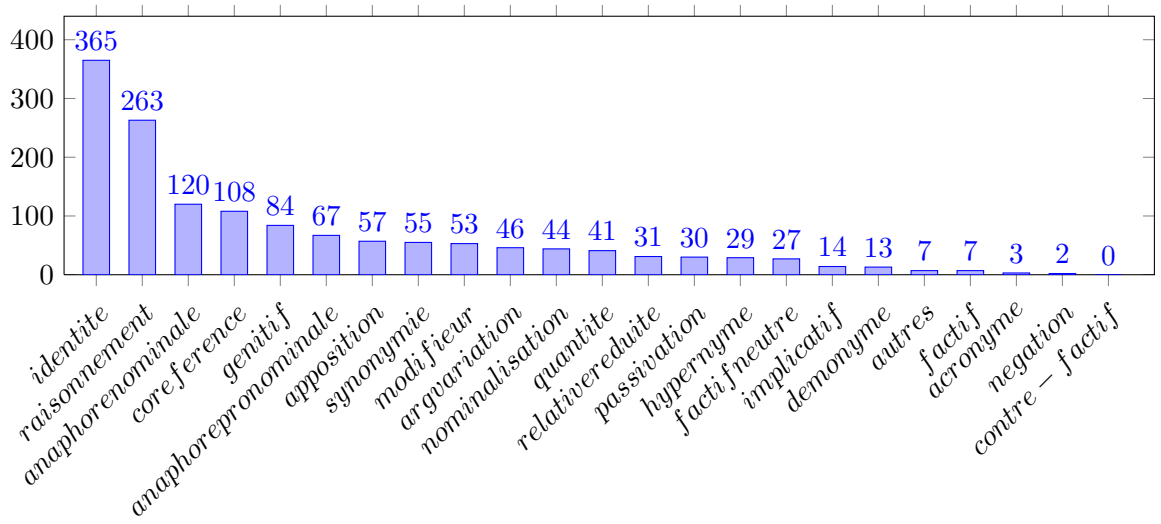


FIGURE 1.12 – Distribution des phénomènes sur le RTE2

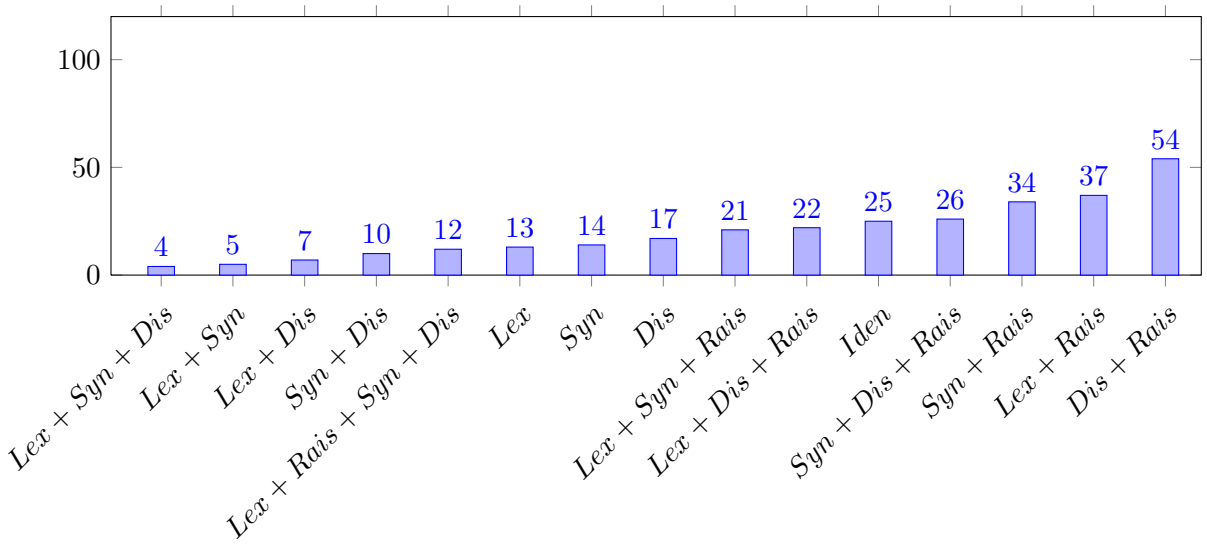


FIGURE 1.13 – Distribution des combinaisons de catégories de phénomènes de façon exclusive sur le RTE2

cours des différentes campagnes. Nous avons ensuite parlé des alternatives existantes : FraCaS proposant de son coté une suite de tests basée uniquement sur la logique et AQUAINT proposant du sien une suite de tests permettant de rendre compte de la variabilité du jugement quant aux ressources utilisées. Enfin, nous avons proposé une analyse critique du challenge RTE concluant que son annotation n'est pas assez abondante et qu'il devrait incorporer plus de problèmes incluant des phénomènes comme la négation et n'utilisant qu'un seul phénomène à la fois.

Dans la suite de ce manuscrit, nous allons présenter le développement d'un système de reconnaissance d'implications textuelles utilisant en cascade des ensembles de règles de réécriture permettant chacun de gérer une tâche particulière comme l'étiquetage de rôles sémantiques sur un verbe ou un nom, et la création de formules sémantiques. Nous exposerons aussi la création de suites de tests générées automatiquement possédant une annotation fournie des phénomènes mis en place, permettant la sélection de la distribution des phénomènes, une évaluation précise ainsi qu'une détection du type d'erreurs les plus probables.

Chapitre 2

Afazio, un système hybride pour la détection d'implications textuelles

Ce chapitre décrit la structure générale de notre système de détection d'implications textuelles appelé Afazio. Son fonctionnement ressemble un peu à celui d'un prouveur de théorème, car il décompose la tâche de détection en deux phases : la première consistant à normaliser la représentation d'un texte, et la seconde visant à vérifier si l'implication est vraie ou non. Afazio est un système hybride combinant un analyseur syntaxique stochastique (Section 2.2) avec des modules symboliques de réécriture et d'inférence logique. Dans la Section 2.3, nous expliquons comment Afazio normalise les analyses syntaxiques en utilisant plusieurs systèmes de réécriture permettant de gérer les variations verbales et nominales ainsi que de générer les formules logiques correspondantes. Dans la Section 2.4 nous présentons le procédé de vérification permettant de savoir si le texte implique l'hypothèse ou non. Enfin, dans la Section 2.5, nous commentons l'utilisation de la réécriture dans les systèmes de RTE.

2.1 Architecture générale d'Afazio

Le système Afazio utilise différents outils afin de détecter les implications textuelles. Il utilise en premier lieu l'analyseur syntaxique de Stanford [Klein & Manning, 2003] afin d'obtenir plusieurs analyses en constituants et en dépendances pouvant être associées au texte et à l'hypothèse. À partir de chaque couple d'analyses de constituants-dépendances, nous construisons un graphe hybride contenant les constituants et les dépendances en même temps afin de combiner les avantages de ces deux types de représentations (plus abstraite pour les dépendances, et plus précise pour les constituants).

Ensuite nous utilisons l'outil de réécriture de graphes GrGen [Kroll & Geiß, 2007]

pour effectuer des transformations sur nos graphes hybrides. Pour commencer, nous normalisons les structures prédicatives verbales à l'aide de règles de réécriture dérivées semi-automatiquement à partir de XTag. Ces règles de réécriture permettent d'ajouter des informations pour lier les prédicats à leurs arguments sémantiques à la manière d'un système d'étiquetage de rôles sémantiques. Puis nous utilisons la ressource NomLex pour créer automatiquement des règles de réécriture permettant de normaliser les structures prédicatives nominales. Ces règles ont pour but de normaliser toutes les variations syntaxiques possibles autour d'un prédicat et utilisent uniquement les dépendances pour cela.

À partir de ces graphes hybrides annotés, nous appliquons un ensemble de règles de réécriture élaborées manuellement permettant d'ajouter à un graphe la structure d'arbre syntaxique de formule de logique du premier ordre lui étant associée. Les règles permettant de créer cette structure se basent sur tout les éléments du graphe (i.e., les constituants, les dépendances, et les annotations prédicatives).

Pour chaque couple de texte-hypothèse, nous prenons les neuf premiers résultats de l'analyseur de Stanford pour maximiser nos chances d'avoir la bonne analyse, puis nous appliquons nos ensembles de règles sur chacune d'elles. Les structures d'arbre syntaxique de formule logique sont ensuite dérivées en formules logiques du premier ordre. Nous utilisons ensuite des outils de preuve pour savoir si une des formules logiques associées au texte implique ou contredit une des formules logiques associées à l'hypothèse.

2.2 Analyse syntaxique

2.2.1 Comparaison des analyseurs syntaxiques pour le RTE

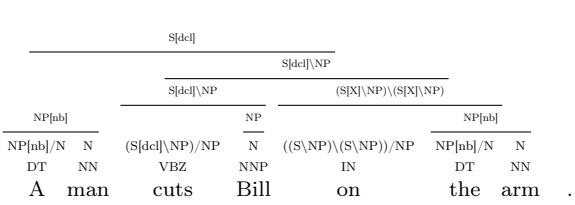
Le cas le plus simple de détection d'implications textuelles consiste à tester si un texte implique une de ses variantes syntaxiques. Pour cela un système doit prendre en compte les variations syntaxiques possibles entre deux phrases véhiculant le même sens de manière à pouvoir faire correspondre les représentations de ces variations. Cette capacité est la plus importante du point de vue de l'analyseur syntaxique pour la détection d'implications textuelles. C'est pourquoi nous avons décidé de comparer plusieurs analyseurs syntaxiques sur cette capacité.

Il existe un grand nombre d'analyseurs syntaxiques basés sur diverses théories. Nous avons décidé de confronter les méthodes formelles et stochastiques, les analyses en constituants et en dépendances, ainsi que les systèmes proposant une ou plusieurs analyses. Pour cela, nous avons sélectionné 5 analyseurs représentant plusieurs des combinaisons méthode d'analyse/type d'analyse/nombre d'analyses possibles comme le montre la table 2.1.

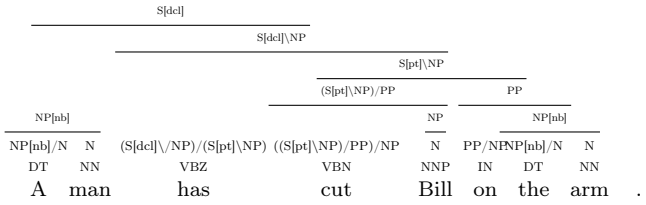
Pour ce qui est de la méthode d'analyse utilisée par les systèmes nous avons choisi

analyseur	méthode	type de sortie	plusieurs analyses
Malt Parser [Nivre *et al.*, 2005]	stochastique	graphe de dépendances	non
Stanford Parser [Klein & Manning, 2003]	stochastique	arbre syntaxique graphe de dépendances	oui
CCG [Clark & Curran, 2007]	stochastique	arbre syntaxique	non
Link Grammar [Sleator & Temperley, 1993]	formelle	graphe de dépendances	oui
Lingo ERG [Copestake & Flickinger, 2000]	formelle	arbre syntaxique	oui

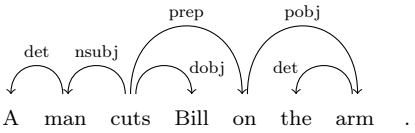
TABLE 2.1 – Comparaison des analyseurs syntaxiques



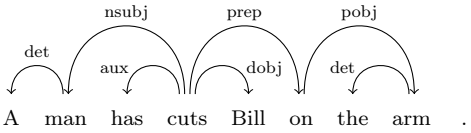
(a) Analyse de CCG pour « A man cuts Bill on the arm »



(b) Analyse de CCG pour « A man has cut Bull on the arm »



(c) Graphe de dépendances de l'analyseur de Stanford pour « A man cuts Bill on the arm »



(d) Graphe de dépendances de l'analyseur de Stanford pour « A man has cut Bill on the arm

FIGURE 2.1 – Analyses syntaxiques de l'analyseur CCG et de l'analyseur de Stanford

la méthode stochastique. Les systèmes symboliques que nous avons examiné donnent des analyses grammaticales fines, mais ont une couverture limitée de la langue et ne sont pas robustes [8]. L'analyseur syntaxique XLE [Maxwell, 1996] aurait pu être une bonne alternative de système symbolique couvrant et robuste, mais son obtention est soumise à des conditions trop restrictives. À l'inverse des analyseurs symboliques, les méthodes stochastiques donnent toujours un résultat même si la phrase contient des mots inconnus et est agrammaticale. Un autre avantage des méthodes stochastiques est leurs rapidités d'analyse, et que leurs complexités par rapport à la longueur de la phrase est en générale inférieure à celles des systèmes symboliques.

Pour que l'implication textuelle soit bien détectée entre un texte et une hypothèse il faut obtenir la bonne analyse pour chacun d'eux. Le problème, est que les analyseurs syntaxiques actuels n'arrivent pas toujours à trouver la bonne analyse. Ce problème est amplifié par le fait que l'on a besoin d'avoir les bonnes analyses pour le texte et l'hypothèse en même temps. Donc si un analyseur syntaxique à une précision de x ($0 \leq x \leq 1$), nous aurons x^2 de chances d'avoir la bonne analyse pour les deux textes. Il est donc important de maximiser les chances d'avoir les deux bonnes analyses, et c'est pourquoi prendre en compte plusieurs analyses plutôt qu'une seule est un bon choix. Cependant il faut faire attention à ne pas prendre trop d'analyses car cela augmente le risque de tomber sur des analyses complètement fausses permettant de considérer des non-implications comme étant des implications.

Le choix de représentation syntaxique n'est pas simple car les deux possibilités (constituants et dépendances) ont chacune des avantages et des inconvénients. La figure 2.1 présente les graphes de dépendances de l'analyseur de Stanford ainsi que les arbres en constituants de l'analyseur Lingo ERG pour les phrases « A man cuts Bill on the arm. » et « A man has cut Bill on the arm. ». Les constituants représentent la structure hiérarchique de la phrase permettant ainsi de connaitre la portée des opérateurs logiques entre eux et sur les prédicats logiques. Les dépendances ont quant à elles l'avantage de normaliser et de simplifier la représentation syntaxique. Par exemple, elles représentent les variations temporelles de façon similaire (i.e., les dépendances liant les arguments au prédicat sont les mêmes dans tous les temps et les auxiliaires sont liés directement au prédicat), contrairement aux constituants où les étiquettes sont différentes et où les auxiliaires coupent le lien entre le premier argument syntaxique et le prédicat. Les informations des constituants sont essentielles pour connaitre les portées (e.g., quantificateurs) et les dépendances permettent une simplification des traitements à effectuer sur la représentation car il y a moins de variations possibles. C'est pourquoi nous avons fait comme choix d'utiliser l'analyseur de Stanford qui satisfait les choix précédents et qui propose à la fois une

8. La robustesse concerne uniquement Lingo ERG car le Link Grammar sait faire des analyses partielles

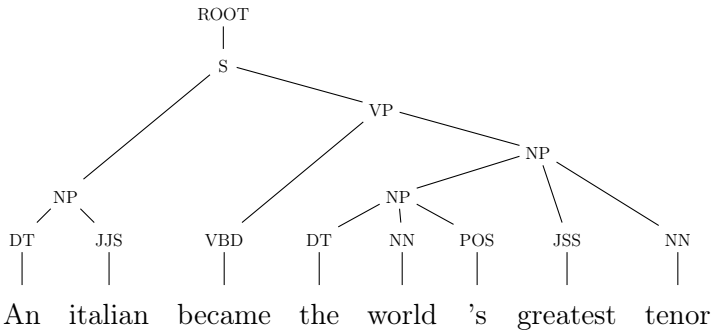


FIGURE 2.2 – Arbre de constituants de l'analyseur Stanford pour « an italian became the world's greatest tenor »

représentation en constituants et en dépendances.

2.2.2 L'analyseur de Stanford

Nous avons déjà vu précédemment que l'analyseur de Stanford est stochastique, et qu'il permet d'obtenir plusieurs analyses en constituants et en dépendances. Pour être plus précis, il utilise une grammaire hors-contexte probabiliste lexicalisée. Il génère avec cette grammaire des analyses en constituants composées de l'ensemble d'étiquettes du Penn Treebank à partir duquel il a été entrainé. Ses analyses en constituants ont une précision de 86,9% et un rappel de 85,7% sur la section 23 du Penn TreeBank. Les graphes de dépendances sont dérivés de ces analyses en constituants ce qui permet d'avoir le graphe de dépendances correspondant à une analyse en constituants [9]. Pour cela des règles de réécriture Tsurgeon [Levy & Andrew, 2006] ont été écrites manuellement comme décrit dans [de Marneffe *et al.*, 2006]. L'analyseur de Stanford met 11 minutes à analyser la section 23 du Penn TreeBank en dépendances, et obtient un F-score [10] de 87,2 pour l'attachement non-étiqueté des dépendances et de 84,2 pour l'attachement étiqueté. La figure 2.2 montre l'analyse en constituants de la phrase « an italian became the world's greatest tenor », et la figure 2.3 le graphe de dépendances obtenu par transformation.

2.3 Transformation des analyses grâce à la réécriture

Comme nous l'avons vu précédemment, nous souhaitons dériver des structures prédicat/arguments puis une représentation sémantique à partir des représentations syntaxiques. Pour cela, nous avons choisi d'utiliser la réécriture et de réaliser des

9. si nous avions choisi de prendre deux analyseurs différents pour les constituants et les dépendances il aurait été difficile voir impossible de combiner les deux types d'analyse.

10. le gold est obtenus en transformant automatiquement, à l'aide de module de conversion de l'analyseur de Stanford, le corpus gold des analyses en constituants.

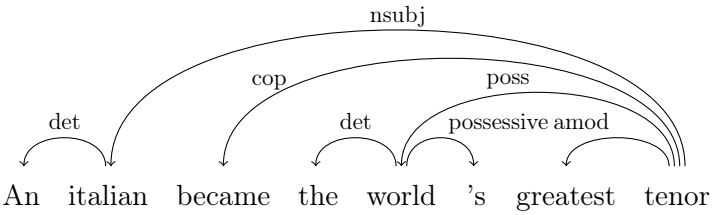


FIGURE 2.3 – Graphe de dépendances de l'analyseur Stanford pour « an italian became the world's greatest tenor »

ensembles de transformations appliqués en cascade pour chacun des modules que nous souhaitons avoir. Nous allons donc dans un premier temps décrire ce qu'est la réécriture de manière théorique puis nous parlerons de GrGen, un outils de réécriture puissant que nous avons utilisé pour développer Afazio.

2.3.1 Théorie

La réécriture est une technique permettant de modéliser des transformations de structures (e.g., graphes, arbres, termes, ...). Elle est utilisée dans de nombreux domaines dont l'algèbre, la logique, ou la linguistique comme nous avons pu le voir précédemment avec la transformation des structures en constituants en graphes de dépendances mise en jeu dans l'approche de Stanford. Un exemple simple permettant de comprendre à quoi peut servir la réécriture et comment elle fonctionne, est le problème de la factorisation de formules mathématiques. On sait que la formule $(a * c) + (b * c)$ est équivalente à la formule $(a + b) * c$ et on veut un moyen de factoriser automatiquement des formules mathématiques. Grâce à la réécriture, il est possible de résoudre ce problème à l'aide de la règle :

$$r1 : (x * y) + (x * z) \rightarrow x * (y + z)$$

représentée graphiquement dans la partie supérieure de la figure 2.4. Cette règle permet entre autres de transformer la formule $((2+3)*4)+(5*4)+5$ en $(((2+3)+5)*)4+5)$, comme le montre la partie inférieure de la figure 2.4. Cette transformation peut être notée :

$$((2 + 3) * 4) + (5 * 4) + 5 \rightarrow_{r1} (((2 + 3) + 5) * 4) + 5)$$

On note $t_0 \rightarrow^*_{r1} t_1$ la transformation d'un terme t_0 en un terme t_1 via zéro ou plus applications de la règle $r1$. Si on ajoutait de nouvelles règles pour gérer plus de cas de factorisation, on obtiendrai un système de réécriture.

Un système de réécriture $\rightarrow_R$ est un ensemble de règles de réécriture de la forme

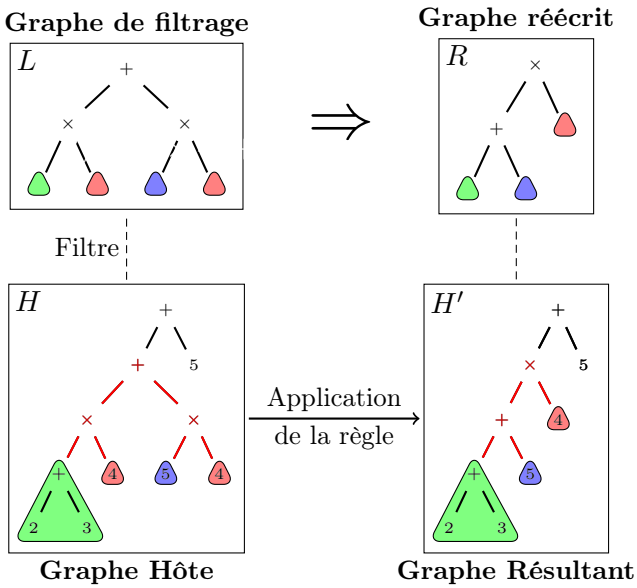


FIGURE 2.4 – Idée générale de la réécriture

$l \to r$, où l et r sont des descriptions avec des parties sous-spécifiées. Une telle règle s'applique à un objet t si celui-ci contient une instance du membre gauche l, c'est-à-dire un sous-objet que l'on peut identifier à l, cette étape s'appelle le filtrage. L'objet t se réécrit alors en un nouvel objet t', obtenu en remplaçant l'instance de l par l'instance réécrite par le membre droit r correspondante. L'ordre d'application des règles ainsi que la technique de parcours des objets représentent la stratégie de réécriture.

Terminaison La terminaison d'un système de réécriture dépend des règles qu'il contient. On dit qu'un système de réécriture est noetherien s'il satisfait la propriété de terminaison. Une façon de démontrer qu'un système de réécriture termine est de construire un ordre de terminaison (i.e., un ordre strict bien fondé) tel que $l \to r$ implique $l > r$. Si le système de réécriture n'est pas noetherien, il faut alors avoir une stratégie permettant au programme de terminer en évitant de tomber dans un cycle infini.

Définition 2.3.1.1 (Terminaison) *Un système de réécriture $\to_R$ termine lorsqu'il n'existe aucune chaîne infinie d'objets $t_0 \to_R t_1 \to_R t_2 \to_R \ldots$*

Dans notre exemple, on aura un tel cycle infini si l'on rajoute la règle :

$$r2 : x * (y + z) \to x * y + x * z$$

représentant l'autre sens de l'équivalence entre les formules $a * c + b * c$ et $(a + b) * c$, et que l'on adopte une stratégie appliquant alternativement les deux règles jusqu'à

que l'une d'elles échoue. Un autre exemple est la règle de réécriture :

$$r3 : x \rightarrow x * 1$$

qui peut s'appliquer à n'importe quel objet en lui rajoutant une multiplication par un. Si on l'applique sur la formule 5, l'algorithme ne terminera pas et créera le terme infini $5 * 1 * 1 \ldots * 1$

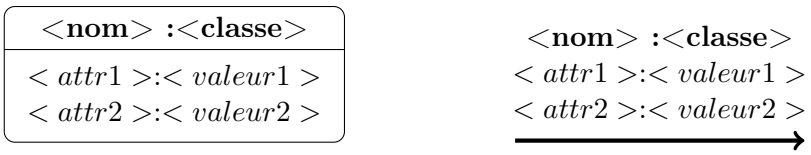
Confluence La confluence est la propriété d'un système de réécriture $\rightarrow_R$ affirmant qu'il est possible d'obtenir la même solution indépendamment de l'ordre d'application des règles.

Définition 2.3.1.2 (Confluence) *Un système de réécriture $\rightarrow_R$ est dit confluent si pour tous termes M, M_1 et M_2 tels que $M \rightarrow_R^* M_1$ et $M \rightarrow_R^* M_2$, il existe M' tel que $M_1 \rightarrow_R^* M'$ et $M_2 \rightarrow_R^* M'$.*

Typiquement, l'application d'un système est régi par une stratégie permettant de définir l'ordre d'application des règles, et le parcours de l'objet permettant de savoir quelle règle est appliquée sur quelle partie de l'objet. La stratégie d'application est très importante car elle peut changer le résultat obtenu. Étant donné le terme $5 * 6 + 5 * 7 + 5 * 8$ et la règle de réécriture $r1$ vu précédemment, selon que l'on parcourt le terme de gauche à droite ou de droite à gauche pour appliquer la règle $r1$ on obtiendra respectivement les termes $5 * ((6 + 7) + 8)$ et $5 * (6 + (7 + 8))$. Comme il n'est plus possible d'appliquer la règle $r1$ sur les termes obtenus et qu'ils ne sont pas identiques, le système de réécriture contenant uniquement la règle $r1$ n'est pas confluent.

2.3.2 GrGen

Nous avons vu précédemment que la création des graphes de dépendances de l'analyseur de Stanford est réalisée grâce à l'outil de réécriture Tsurgeon. Nous avons préféré utiliser GrGen [Kroll & Geiß, 2007] pour trois raisons. Premièrement, GrGen est nettement plus expressif que Tsurgeon, il permet par exemple de définir des règles de réécriture permettant de filtrer un nombre indéterminé de nœuds (i.e., pour recopier tout les arcs entrant dans un nœud sur un autre nœud). Il a aussi un système pour définir les stratégies d'application des règles. Et enfin le langage permettant de définir les règles de filtrage et les transformations est nettement plus lisible pour GrGen, ce qui facilite la maintenance.



(a) Représentation d'un nœud (b) Représentation d'un arc

FIGURE 2.5 – Représentation graphique pour les graphes.

Modèle de graphe

Les graphes utilisés dans GrGen sont des graphes dirigés étiquetés typés. Avant de spécifier des règles de réécriture où de définir un graphe, il est nécessaire de définir un modèle de graphe. Pour cela, il faut définir des classes de nœuds et d'arcs pouvant être instanciés. À chaque classe est associé une liste d'attributs typés (i.e., chaine de caractère, entier, booléen, ...) que chaque instance de la classe possèdera. Une classe peut hériter d'une ou plusieurs classes. Quand une classe hérite d'une autre classe, elle hérite de ses attributs et sera filtrée par un motif cherchant à filtrer la super classe. Par exemple, si on a une classe `forme` et une classe `polygone` héritant de `forme` ; le filtrage d'un nœud de classe `forme` sélectionnera toutes les instances de `forme` et de `polygone`. Pour mieux visualiser nos graphes nous les représentons graphiquement comme le montre la figure 2.5. Cette représentation sera réutilisée dans les règles de réécriture pour définir les motifs de filtrage et de transformation. Le <nom> servira uniquement dans les règles de réécriture.

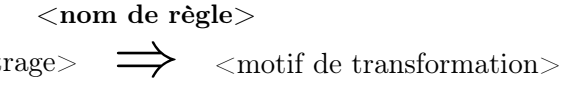
Que cela soit pour représenter un graphe, un motif de filtrage, ou un motif de transformation, nous utilisons la même représentation graphique comme défini ci-dessous, mais il y a cependant quelques différences de sémantique selon le cas. Le <nom> du nœud sert uniquement à avoir une correspondance de nœuds entre le motif de filtrage et le motif de transformation. La classe permet de connaitre la classe du nœud ou de l'arc dans un graphe, la classe du nœud ou de l'arc que l'on veut filtrer dans le motif de filtrage et la classe des nœuds ou des arcs créés pour le motif de transformation (il n'est pas possible de modifier la classe d'un nœud existant). Enfin les couples <attributs> :<valeur> représentent les attributs du nœud ou de l'arc pour un graphe, des contraintes que l'on souhaite vérifier pour le motif de filtrage et enfin des assignations de valeurs pour le motif de transformation.

Règle de réécriture

Une règle de réécriture est composée d'un <nom de règle> permettant de différencier les règles de réécriture, d'un <motif de filtrage> avec des contraintes et d'un <motif de transformation>. Les nœuds et les arcs peuvent être nommés. Deux éléments avec des noms différents ne peuvent pas faire référence au même élément dans

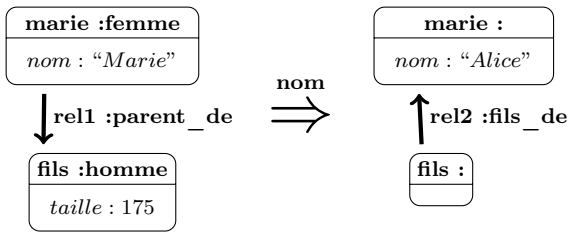
le graphe hôte (à part si on le spécifie avec le prédicat `hom(<liste d'éléments>)`). De base, le motif de graphe cherche juste à filtrer les connexions qu'il définit. Si le graphe hôte à plus de connexions cela ne pose pas de problème. Cependant on peut rendre les motifs plus sélectifs en imposant qu'un nœud n'ait que les connexions définies dans le motif de graphe ou limiter les connexions au sein d'un ensemble de nœuds. Nous définissons une règle de réécriture en terme de code source et de représentation graphique de la façon suivante :

```
rule <nom de règle> {
  <motif filtrage>
  modify{
    <motif de transformation>
  }
}
```



Les motifs de filtrage et de transformation sont composés de nœuds et d'arcs de la même manière qu'un graphe normal. Leurs représentations (c.f., figure 2.5) est la même mais la sémantique change. Le <nom> des arcs et des nœuds sert uniquement à avoir une correspondance entre les éléments du motif de filtrage et le motif de transformation pour savoir quels éléments sont modifiés, supprimés ou ajoutés. La <classe> permet de connaitre la classe du nœud ou de l'arc que l'on veut filtrer dans le motif de filtrage et la classe des nœuds ou des arcs crées pour le motif de transformation (il n'est pas possible de modifier la classe d'un nœud existant). Enfin les couples <attributs> :<valeur> des contraintes que l'on souhaite vérifier pour le motif de filtrage et des assignations de valeurs pour le motif de transformation. Un exemple de règle de réécriture est visible ci-dessous. Il représente une règle qui cherche à filtrer un nœud **marie** de classe **femme** avec la valeur *"Marie"* pour l'attribut *nom*, et un nœud **fils** de classe **homme** avec la valeur *175* pour l'attribut *taille* tel qu'il existe un arc **rel1** de classe **parent_de** connectant le nœud **marie** au nœud **fils**. Une fois ce motif de graphe filtré, il supprime l'arc **rel1**, ajoute l'arc **rel2** de classe **fils_de** entre le nœud **fils** et le nœud **marie**, puis affecte la valeur *"Alice"* à l'attribut *nom* du nœud **marie** (l'attribut *taille* du nœud **fils** n'est pas modifié).

```
rule nom {
  marie:femme;
  fils:homme;
  marie -rel1:parent_de-> fils;
  if (marie.nom=="Marie"};
  if (fils.taille==175};
  modify{
    delete(rel_down);
    fils -rel2:fils_de-> marie;
    eval{marie.nom=="Alice"};
  }
}
```



Les conditions sur les attributs ou la classe des éléments sont intégrées dans le motif de filtrage. Nous représentons les conditions plus complexes, avec des disjonctions

par exemple, en mettant directement le code source correspondant aux conditions dans la représentation graphique. Pour tester que la classe du nœud **jean** appartient à une classe parmi les classes **homme** et **garçon**, nous avons la règle :

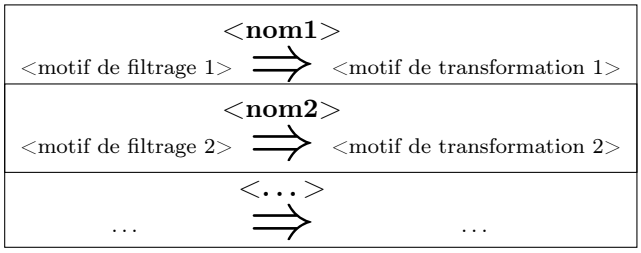
```
if{ classeof(jean) == homme | classeof(jean) == garçon; }
```

La négation permet de définir un motif de graphe qui ne doit pas pouvoir être filtré pour que la règle puisse filtrer. Les variables déclarées dans la négation sont par défaut homomorphiques avec les variables de la porté supérieure (i.e., les nouveaux éléments définis dans la négation ne peuvent pas être les mêmes que ceux définis dans la porté supérieure).

```
negative{
  <motif filtrage>
}
```

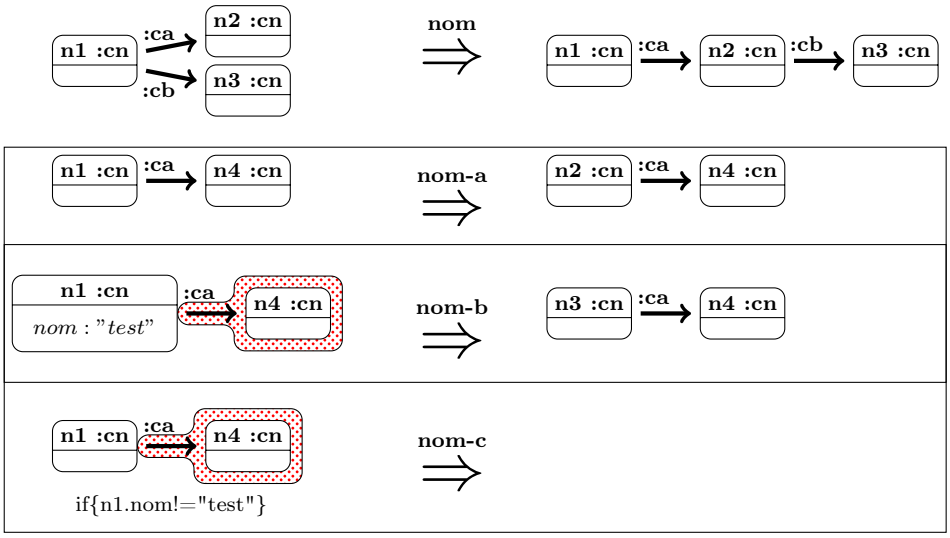
Les alternatives permettent de factoriser les règles de réécriture, ce qui améliore le temps d'application des règles, et facilite leur maintenance. Une des alternatives doit être filtrée pour que la règle principale soit filtrée. Les alternatives représentent une disjonction entre plusieurs sous-motifs de filtrage, et selon le sous-motif filtré, différents sous-motifs de transformation sont appliqués. L'ordre de filtrage des alternatives n'est pas spécifié, et si plusieurs alternatives sont filtrées, l'ordre dans lequel elles le seront ne dépend pas de leur ordre d'apparition. Il est donc important de concevoir les alternatives de façon à ce qu'une seule puisse être filtrée à la fois en utilisant des négations. Il est possible dans une alternative d'accéder aux éléments définis dans la règle principale, mais les éléments déclarés dans une alternative ne sont accessibles que dans cette alternative. Il est possible d'avoir plusieurs blocs d'alternatives imbriqués dans une règle de réécriture. La définition des alternatives sous la forme de code source et de représentation graphique est visible ci-dessous.

```
alternative{
  <alternatve 1>{
    <motif filtrage 1>
    modify{
      <motif de transformation 1>
    }
  }
  <alternative 2>{
    <motif filtrage 2>
    modify{
      <motif de transformatios 2>
    }
  }
  ...
}
```

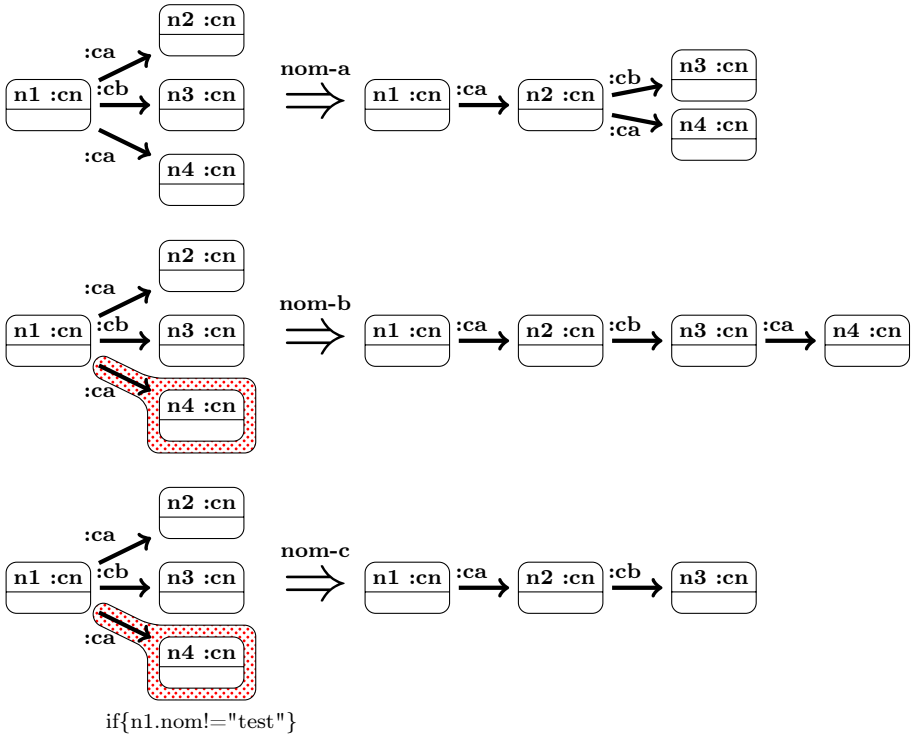


L'exemple ci-dessous permet de mieux comprendre la sémantique des alternatives. Nous avons tout d'abord l'exemple avec l'utilisation des alternatives puis les trois règles correspondantes si on n'utilise pas les alternatives. Nous pouvons voir que nous

utilisons bien dans les alternatives des éléments de la règle principale. Les éléments de motif de transformation d'une alternative peuvent appartenir aux éléments définis dans de motif de filtrage de l'alternative ou dans la règle principale. Dans cette exemple les nouveaux nœuds `n4:cn` définis dans les alternatives représentent des nœuds différents (à la manière d'arguments dans des fonctions en programmation). La négation de la règle *nom-b* permet de ne pas filtrer si *nom-a* a filtré, et la négation et le `if` de *nom-c* de ne pas filtrer si *nom-a* ou *nom-b* ont filtré. Cela permet ainsi d'avoir un ordre d'importance entre les alternatives. L'alternative *nom-a* sert de cas par défaut si aucune des autres alternatives n'a été filtrée on applique ainsi uniquement la règle principale.



Voici les règles correspondantes à la sémantique de l'alternative de la règle ci-dessus :



Les sous-règles permettent de factoriser des motifs de graphe récurrents dans plusieurs règles. Elles peuvent prendre des paramètres de type nœud ou arc en entrée. Elles permettent d'avoir du code plus clair et flexible et permettent aussi en combinaison avec les alternatives de faire des règles récursives pour chercher un connexion via un nombre indéterminé d'arcs entre deux nœuds. Nous représentons ces sous-règles de la même manière que les règles normales mais en rajoutant une signature. Quand elle est appelé, une sous-règle teste le motif de filtrage, et si celui-ci filtre elle renvoie une fonction qui permet d'appliquer les transformations du motif de transformation quand elle est appelée. Les paramètres des sous-règles sont soit des occurrences de nœuds où d'arc, soit des classes de nœuds (`cnode`) ou d'arcs (`cedge`). Pour utiliser la sous-règle ci-dessous il faudra faire les appels suivant dans le motif de filtrage et de transformation de la règle :

$$sr :< nom > (af1, af2, \dots); \qquad sr(at1, at2, \dots);$$

La définition des sous-règles est la suivante :

```
pattern <nom>(pf1:pfc1, pf2:pfc2, ...)
      modify (pt1:ptc1, pt2:ptc2, ...){
  ...
  modify{
    ...
  }
}
```

$$<\textbf{nom}>$$
$$(pt1 : ptc1, pt2 : ptc2, \dots)$$
$$\overrightarrow{(pf1 : pfc1, pf2 : pfc2, \dots)}$$
$$<\text{motif de filtrage}> \Longrightarrow <\text{motif de transformation}>$$

Stratégie de réécriture

Pour déterminer l'ordre d'application des règles, GrGen permet de regrouper les règles en séquences puis d'imposer différents types de contraintes sur l'exécution de ces séquences. Une séquence de règles peut être disjonctive, conjonctive ou itérative. Deux ou plusieurs séquences de règles peuvent être combinées par concaténation (la combinaison réussit si au moins l'une des séquences de règles peut être appliquée avec succès), par une conjonction transactionnelle (pour que la combinaison de séquences soit exécutée, il faut que chacune des séquences spécifiées puisse être appliquée dans l'ordre spécifié) et par une disjonction exclusive (la combinaison de séquences réussit dès que l'une des séquences spécifiées peut être appliquée). Il est par ailleurs possible d'imposer soit une application « globale » (la règle est appliquée à toutes les structures filtrées par son motif dans le graphe réécrit) soit une application « unique » (la règle n'est appliquée qu'à une seule structure).

2.3.3 Description brève de l'utilisation faite de la réécriture avec exemple.

Le système Afazio utilise GrGen pour transformer les analyses du Stanford parser. Ces transformations se font en trois étapes :

- Normalisation de la réalisation des arguments verbaux
- Normalisation de la réalisation des arguments nominaux
- Conversion des graphes syntaxico-sémantiques en formules de logique de premier ordre

Nous utilisons comme structure de départ pour la réécriture, une structure contenant à la fois les analyses en constituants et en dépendances produites par l'analyseur de Stanford. Le graphe de la figure 2.6 représente la structure associée à la phrase « The city was destroyed by Caesar ».

Les structures et règles manipulées par Afazio seront détaillées dans le chapitre 4, l'exemple qui suit illustre les structures produites par les différents modules de réécriture.

Pour identifier les structures prédicatives nous appliquons des règles de réécriture qui ajoutent pour chaque structure prédicative un nœud représentant le prédicat relié au verbe et à ses arguments par des arcs. Nous avons fait le choix d'ajouter de l'information plutôt que de la transformer car un fragment du graphe peut être nécessaire à plusieurs niveaux de transformation (e.g., lors de l'étiquetage sémantique et de la création de la représentation sémantique). Cependant nous voulons éviter que certaines règles s'appliquent sur les mêmes fragments de graphe (e.g., les règles d'étiquetage sémantique de l'actif pour les verbes transitifs et ergatifs), pour cela nous ajoutons des drapeaux (i.e., un booléen appelé *used*) aux mots pour savoir s'ils

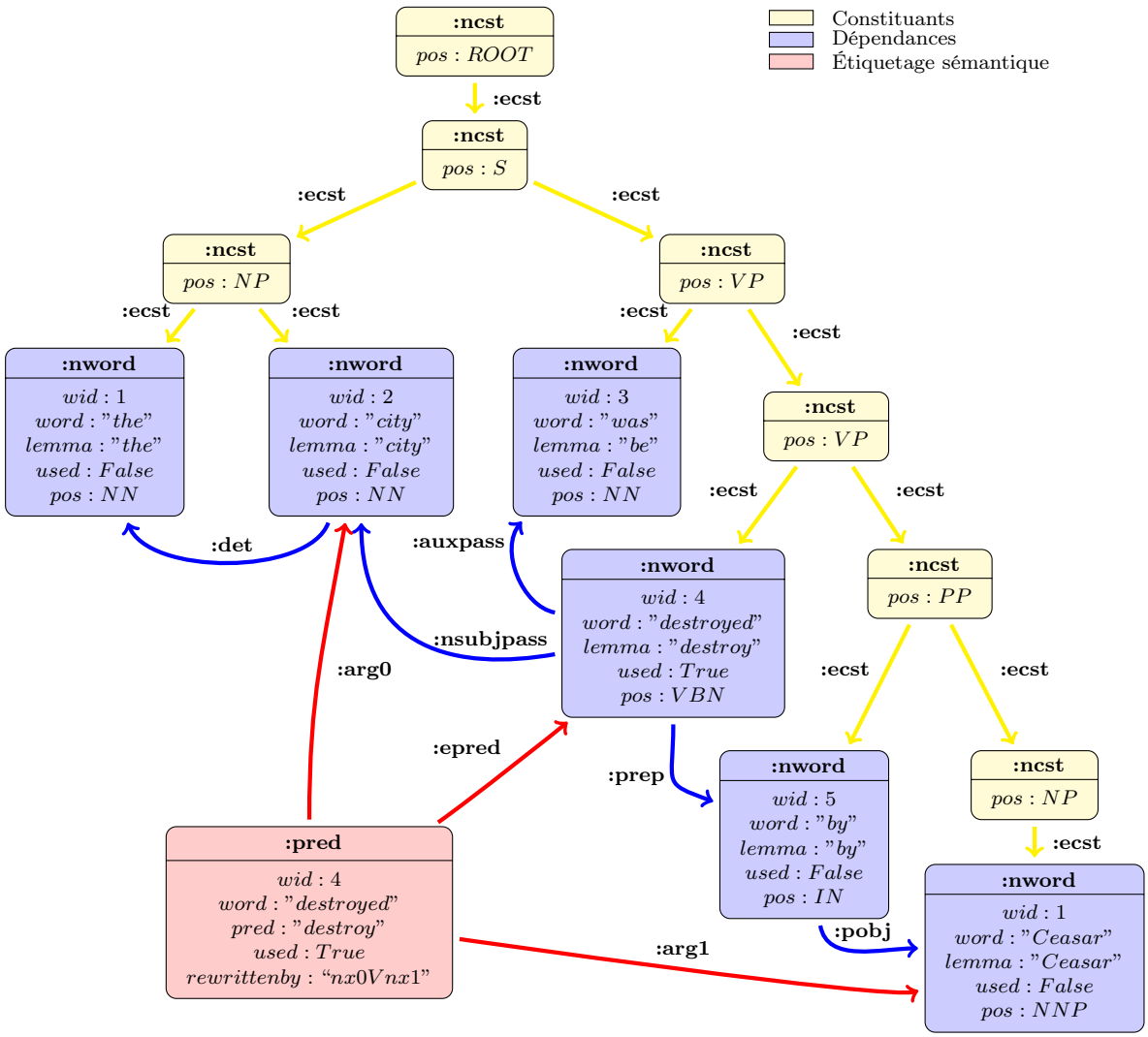


FIGURE 2.6 – Structure syntaxique du Stanford Parser en constituants et en dépendances annotée avec des rôles sémantiques pour la phrase « The city was destroyed by Caesar »

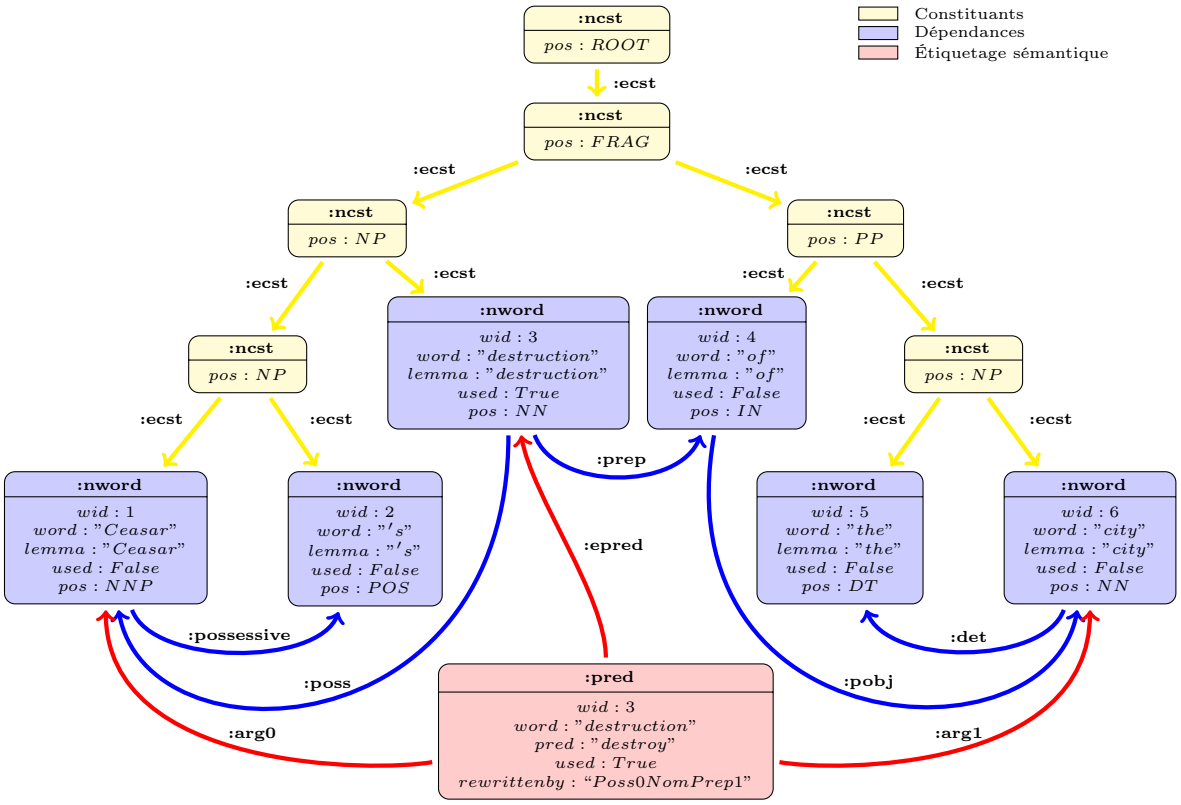


FIGURE 2.7 – Structure annotée pour « Ceasar's destruction of the city »

ont été étiquetés ou non. La figure 2.6 montre l'analyse syntaxique en constituants et en dépendances de la phrase « The city was destroyed by Ceasar » étiquetée sémantiquement. Dans ce graphe l'étiquetage sémantique a été rajouté au graphe de base. Nous pouvons ainsi voir l'analyse en dépendances, en constituants et l'étiquetage sémantique sur le même graphe. Notre objectif est d'obtenir les mêmes annotations sémantiques pour les différentes variations verbales/nominales possibles. Les variations verbales sont développés à partir de XTAG et les variations nominales à partir de NomLexPlus. L'objectif est d'obtenir les graphes des figures 2.8 et 2.7 pour les phrases « Caesar destroyed the city » et « Caesar's destruction of the city » :

Notre système ressemble dans sa phase de normalisation à un d'étiqueteur sémantique, mais contrairement à ceux-ci, il cherche à avoir un étiquetage le plus près de la sémantique. Par exemple dans CoNLL'09 [Hajič *et al.*, 2009], lorsqu'il y a une relative les systèmes lient le prédicat au pronom relatif et pas à l'argument réel comme le fait notre système. Les graphes ainsi annotés sont ensuite augmentés d'une couche représentant l'arbre syntaxique de la formule du premier ordre correspondant à la sémantique de la phrase. Sur l'exemple de « Caesar destroyed the city », on obtient la structure syntaxo-sémantique de la figure 2.9.

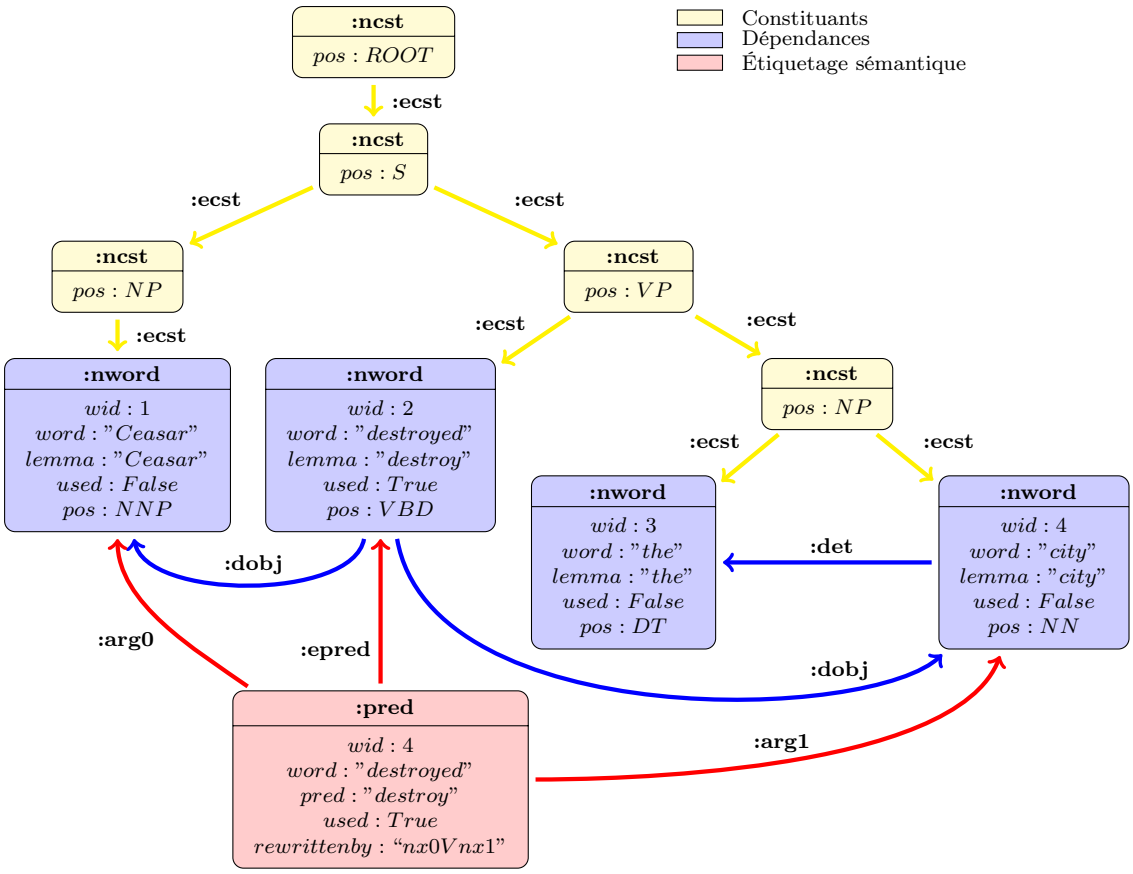


FIGURE 2.8 – Structure annotée pour « Ceasar destroyed the city »

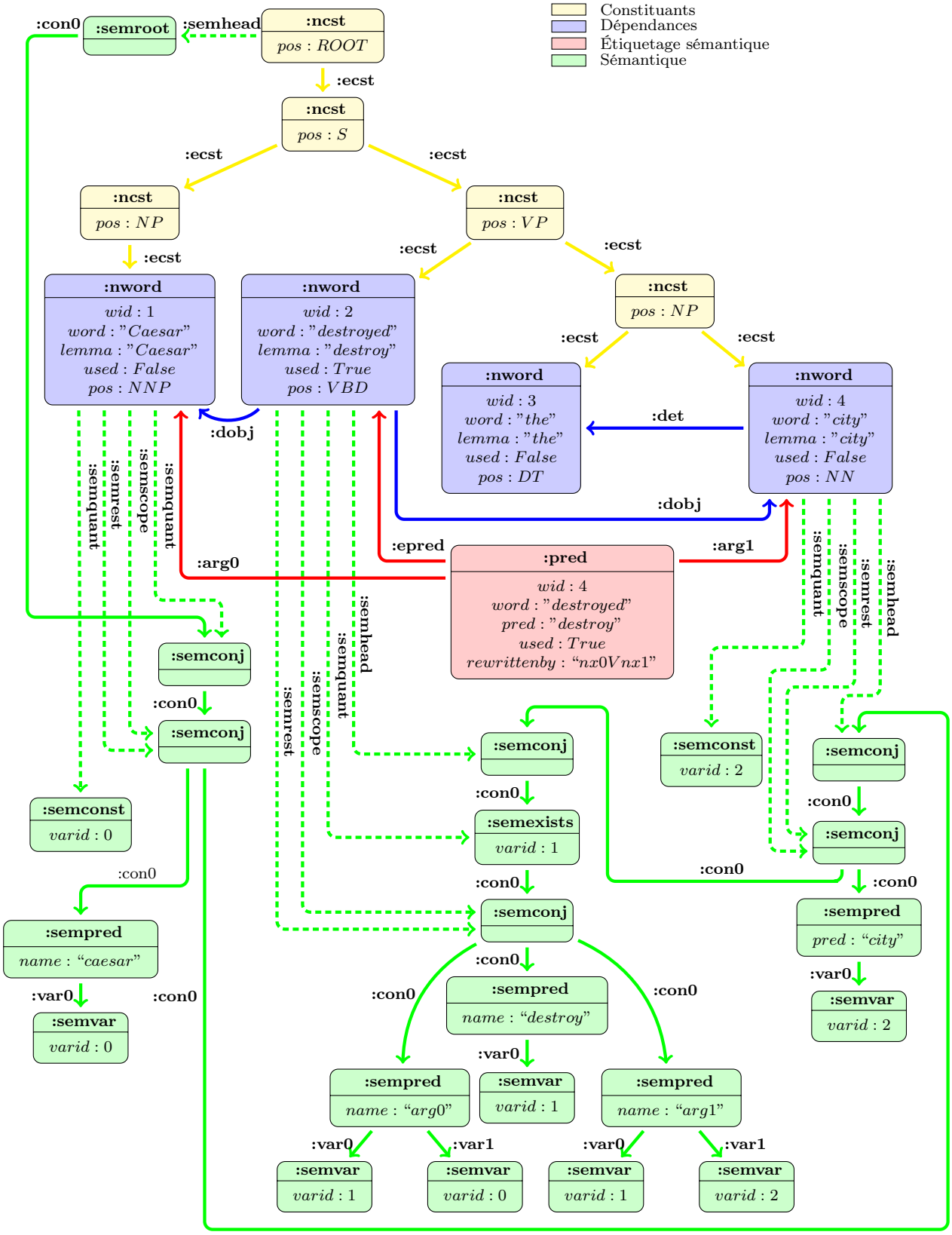


FIGURE 2.9 – Structure finale pour « Caesar destroyed the city »

Les phénomènes comme la négation, ou les implicatifs sont pris en compte lors de la création de ces formules.

2.4 Raisonnement automatique

La dernière étape consiste à vérifier que les structures sémantiques dérivées par réécriture pour T et H sont ou non dans une relation d'implication. Notons que dans notre approche nous ne testons pas l'implication textuelle, mais l'implication logique, donc pour un couple d'implications comme (10) notre système donnera une réponse négative alors que l'implication textuelle est considérée comme vraie car fortement probable.

(10) **T :** « About two weeks before the trial started, I was in Shapiro's office in Century City. »
H : « Shapiro works in Century City. »
Implication : OUI

Pour vérifier que le texte implique ou contredit l'hypothèse, nous prenons les formules de logique du premier ordre T_{lpo} et H_{lpo} qui leur sont respectivement associées, puis nous testons la validité de la formule $T_{lpo} \Rightarrow H_{lpo}$ pour l'implication et de la formule $T_{lpo} \Rightarrow \neg H_{lpo}$ pour la contradiction comme cela est décrit dans [Blackburn & Bos, 2005]. Il est nécessaire de tester la validité et non la satisfaibilité, car mise à part le cas où le texte et l'hypothèse se contredisent, il y aura toujours un modèle dans lequel le texte et l'hypothèse pourront être satisfiables en même temps et satisfaire ainsi l'implication.

Définition 2.4.0.1 (Validité) *Une formule est valide si et seulement si elle est vraie dans toutes les interprétations possibles du langage.*

Définition 2.4.0.2 (Satisfiabilité) *Une formule est satisfiable si et seulement si il existe au moins une interprétation pour laquelle elle est vraie.*

Dans notre approche, le système récupère les 9 premières analyses du texte et de l'hypothèse auxquelles nous appliquons notre système de réécriture pour étiqueter sémantiquement les structures prédicatives. Nous appliquons ensuite un autre système de réécriture permettant de dériver à partir de ces structures des arbres syntaxiques de formules logiques du premier ordre. Ces arbres sont ensuite transformés en formules, et c'est ainsi que nous obtenons pour l'exemple de la figure 2.9 la formule :

$$(caesar(c0) \wedge (city(c2) \wedge \exists V1.(destroy(V1) \wedge arg0(V1, c0) \wedge arg1(V1, c2))))$$

À partir de ces formules nous créons la formule ci-dessous permettant de savoir si une des sémantiques associées au texte implique une des sémantiques associées à l'hypothèse. Cela donne la formule suivante :

$$(t_1 \wedge t_2 \wedge \ldots \wedge t_9) \Rightarrow (h_1 \vee h_2 \vee \ldots \vee h_9) \qquad \text{(Implication)}$$

$$(t_1 \wedge t_2 \wedge \ldots \wedge t_9) \Rightarrow (\neg h_1 \vee \neg h_2 \vee \ldots \vee \neg h_9) \qquad \text{(Contradiction)}$$

Il est aussi possible de tester successivement les implications entre tous les couples formés par le produit cartésien des formules associées au texte et de celles associées à l'hypothèse. Cette solution a l'avantage de permettre de savoir quels couples de sémantiques sont impliqués ou contredits. Le problème de l'utilisation de plusieurs analyses à la fois, est qu'il est possible d'avoir des couples de texte-hypothèse tel que le texte implique et contredit l'hypothèse en même temps. Dans ce cas nous prenons la réponse utilisant les sémantiques associées aux analyses les mieux classées par l'analyseur de Stanford. Ce que nous souhaitons tester maintenant, c'est la validité de ces formules. Comme un raisonneur automatique ne permet que de savoir si une formule est satisfiable ou non, pour savoir si une formule ϕ est valide, nous devons tester si la formule $\neg\phi$ est insatisfiable.

Comme la logique du premier ordre est indécidable, il est possible que le calcul ne termine pas. Il faut donc imposer un temps de calcul maximum à partir duquel nous arrêtons notre calcul. Le problème est que si la durée maximum est atteinte, nous n'obtenons aucune information sur la satisfaisabilité de la formule. C'est pourquoi, afin de maximiser nos chances d'avoir un résultat, nous utilisons deux systèmes complémentaires en parallèle pour avoir notre réponse comme le propose [Blackburn & Bos, 2005]. Le premier, Equinox[11], est un prouveur de théorème qui essaye de trouver une preuve que la formule est insatisfiable. Le second, Paradox[11], est un constructeur de modèles qui cherche une interprétation pour laquelle la formule est vraie afin de montrer sa satisfaisabilité.

Pour prendre en compte des phénomènes comme la synonymie, ou l'antonymie, on extrait de ressources linguistiques (i.e., WordNet [Fellbaum, 1998]) les connaissances pouvant exister entre les lemmes du texte et de l'hypothèse. Par exemple, si le texte contient le mot « cat » et l'hypothèse le mot « animal » qui est un hyperonyme de « cat », on ajoutera alors à la connaissance de base l'axiome $\forall x.(chat(x) \rightarrow animal(x))$ formalisant le fait qu'un chat est un animal.

11. `http://www.cs.chalmers.se/~koen/folkung/`

2.5 La réécriture dans les systèmes de DIT

La réécriture est un outil puissant de transformation de graphes et d'arbres, c'est pourquoi elle est fortement utilisée par la communauté du TAL. Nous allons donc examiner différents systèmes de TAL utilisant cette technique.

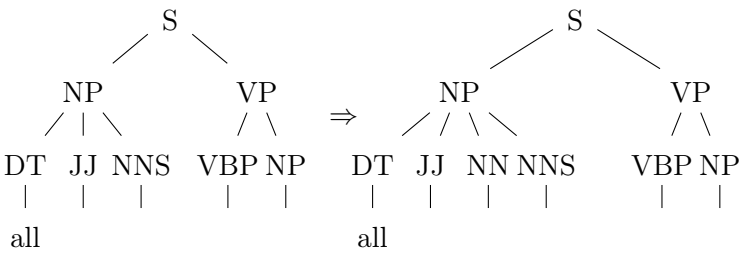
L'approche de Fowler *et al.* [2005] utilise le système COGEX (une version modifiée du prouveur OTTER) pour reconnaitre les implications textuelles. Ils utilisent ensuite des axiomes qui permettent d'exprimer les équivalences syntaxiques et de réduire la complexité des autres formules logiques. Le problème de cette méthode, est qu'une implication en logique du premier ordre (qui est la logique utilisé par OTTER) permet seulement de rajouter de l'information. Cela permet donc à plusieurs axiomes de s'appliquer à un prédicat même s'ils se contredisent (e.g., pour la phrase « John sinks the boat » si l'axiome ergatif s'applique en même temps que le transitif, « John » sera considéré comme étant le premier et le second argument sémantique de « sinks »). Pour éviter que plusieurs règles s'appliquent à un prédicat, il faudrait utiliser la logique linéaire qui permet la consommation de ressources. Mais il resterait encore le problème de l'ordre d'application des règles qui ne pourrait pas être contrôlé, et il serait possible que l'axiome ergatif s'applique sur une verbe transitif avant l'axiome transitif laissant ainsi le second argument non traité.

Jijkoun & de Rijke [2007] proposent de voir la tâche de TAL comme une tâche de transformation de graphes. Ils présentent une méthode permettant d'apprendre les transformations nécessaires pour dériver un corpus annoté à partir de la version non annotée du corpus. Ils ont ainsi crée un système permettant de faire de l'étiquetage de rôles sémantiques à partir de PropBank [Palmer *et al.*, 2005] composé de 320 règles ordonnées. Ce système obtient une précision de 81,0%, un rappel de 70,4% et un f-score de 75,3% (contre une référence de 78,8% pour les systèmes actuels [Pradhan *et al.*, 2005]) Les règles obtenues forment un tout et ne peuvent pas être utilisées indépendamment pour gérer un sous-ensemble de phénomènes. Il est très difficile de corriger manuellement les règles, car si on identifie les règles impliquées dans la gestion d'un phénomène et que l'on corrige l'une d'elles pour améliorer la gestion du phénomène, on modifiera aussi la gestion des autres phénomènes utilisant cette règle.

Le système DIRT (Discovering Inference Rules from Text) [Lin & Pantel, 2001] permet d'extraire des règles d'inférence du genre « X writes Y » implique que « X is the author of Y » à partir d'un corpus de textes. Il utilise pour cela l'hypothèse distributionelle qui déclare que les mots apparaissant dans le même contexte tendent à avoir le même sens [Harris, 1954], qu'ils appliquent à des chemins dans des graphes de dépendances. L'idée est intéressante mais les résultats obtenus sont faibles. Le système pose problème sur les prédicats polysémiques, où un sens prend le dessus, et surtout pour les prédicats ayant des arguments de même type comme « X asks Y »

qu'il associe à lui même dans la configuration inverse « Y asks X ».

L'approche présentée part Zanzotto *et al.* [2006] propose aussi de découvrir automatiquement des règles de réécriture mais cette fois à partir d'exemples d'implications et de non-implications textuelles. L'objectif est d'avoir des règles permettant de dériver l'hypothèse à partir du texte s'il y a implication. Les règles obtenues par le système sont à l'inverse de DIRT, très spécialisées. Elle représentent en général une combinaison de phénomènes apparaissant dans plusieurs couples. Par exemple, la règle ci-dessous permettant d'obtenir les implication comme « All wild animals eat plants » implique « All wild mountain animals eat plants ».

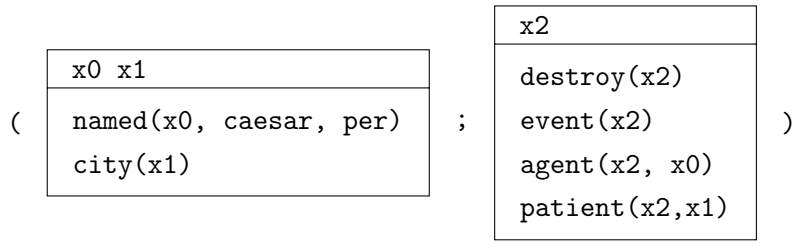


Le problème de cette règle est qu'elle est trop spécialisée car elle gère le cas de monotonicité montante du première argument quand le déterminant de celui-ci est « all », pour un verbe transitif à l'actif, pour une adjonction d'un NN au premier argument. Il faudrait donc un nombre énorme de règles pour pouvoir prendre en compte toutes les combinaisons de phénomènes possibles, ce qui n'est pas envisageable avec un corpus de la taille du RTE. Pour cet exemple il serait préférable d'avoir une règle qui gère l'actif, une les monotonicités du quantificateur, et que le reste soit géré par les outils de preuve.

L'article [Bar-Haim *et al.*, 2007] introduit un système qui recherche des preuves permettant de démontrer que l'hypothèse peut être déduite à partir du texte. Les preuves sont réalisées en cherchant une suite de règles d'inférence à appliquer sur l'arbre syntaxique du texte, pour obtenir celui de l'hypothèse à la manière de la logique naturelle [MacCartney & Manning, 2009]. Les règles d'inférence sont extraites de WordNet ainsi que du premier CD du corpus RCV1 de Reuters à l'aide de l'algorithme utilisé pour créer les règles de DIRT. Il y a donc dans cette approche le même problème que pour les règles de DIRT. Mais il y a aussi le problème de la recherche de preuve, où il faut trouver la suite de transformations permettant de dériver H à partir de T si T implique H. Le choix du système est de faire une recherche en profondeur avec une profondeur maximale de 4, et s'il est nécessaire de faire 5 transformations pour dériver H à partir de T, alors l'implication ne sera pas détectée.

Pour finir, nous allons présenter le système Nutcracker [Bos & Markert, 2005], qui est actuellement le système le plus proche d'Afazio. Il utilise aussi un analyseur syntaxique stochastique puis transforme les arbres syntaxiques en DRS (Discourse

Representation Structure). Il diffère cependant d'Afazio sur certains points. Tout d'abord l'analyseur syntaxique qu'il utilise ne lui donne qu'une seule analyse, ce qui limite ses chances de trouver l'implication comme nous l'avons vu précédemment. Ensuite, la normalisation syntaxique réalisée est moins élaborée et est gérée sous la forme d'axiomes logiques, augmentant ainsi la complexité des formules testées. Enfin, la création de la sémantique est faite en utilisant le lambda-calcul alors qu'Afazio utilise la réécriture. Nous avons utilisé la réécriture plutôt que le lambda-calcul afin d'avoir un système entièrement basé sur ce modèle de calcul. La DRS obtenue pour la phrase « Caesar destroyed the city » est la suivante :



2.6 Conclusion

Ce chapitre nous a permis d'exposer la structure générale de notre système composé de l'analyseur syntaxique de Stanford, choisi pour sa capacité à gérer les variations syntaxiques ; de plusieurs modules de réécriture en cascade permettant de normaliser les variations verbales et nominales puis de transformer les représentations en arbres syntaxiques de formule logique du premier ordre, et d'outils de raisonnement logique permettant de vérifier si une formule obtenue en implique une autre. En conclusion, nous avons brièvement passé en revue les approches similaires existantes afin de montrer les avantages et inconvénients de notre système. Nous allons maintenant décrire les suites de tests que nous avons créées, et les méthodes utilisées pour évaluer d'une façon plus précise les systèmes de reconnaissance d'implication textuelle.

Chapitre 3

Construire des suites de tests pour la DIT

Dans ce chapitre, nous argumentons en faveur de la construction de suites de tests complémentaires à celles de la campagne RTE. Alors que la campagne RTE vise à évaluer la capacité des systèmes à détecter l'implication textuelle sur des données produites dans le cadre d'applications données (systèmes question-réponse, traduction et résumé automatique, recherche et extraction d'information), les suites de tests que nous proposons visent à permettre une évaluation analytique des systèmes à traiter certains types d'implications comme les implications basées sur la syntaxe, la sémantique lexicale ou la sémantique compositionnelle. Nous commençons par motiver cette nouvelle approche puis nous décrivons le processus de construction des suites de tests. Enfin, nous proposons un algorithme permettant de sélectionner un échantillon d'une suite de tests satisfaisant au mieux un certain nombre de contraintes.

3.1 Méthodologie

Dans le chapitre 1, nous avons fait ressortir que la campagne d'évaluation RTE ne permettait pas une évaluation précise des systèmes du fait de sa distribution inégale des phénomènes linguistiques et sémantiques représentés et de son schéma d'annotation minimal. Le problème de cette évaluation est qu'elle vise uniquement une comparaison des systèmes d'un point de vue applicatif. Nous avons ensuite présenté des alternatives (FraCaS et AQUAINT) proposant une meilleure répartition et annotation des problèmes, mais de taille trop réduite pour permettre une réelle évaluation. Pour pallier ces lacunes, nous proposons ici une évaluation analytique des systèmes grâce à des suites de tests construites semi-automatiquement.

La création semi-automatique d'une suite de tests de RTE permet d'avoir une annotation détaillée des couples de phrases ; d'éviter les problèmes trop simples de

non-implication ; d'avoir un nombre conséquents de problèmes ; et de contrôler la distribution des types d'implications présents dans les données. Plus généralement, cela permet de pouvoir réaliser une analyse sophistiquée des erreurs que les systèmes font au moyen de techniques de recherche d'erreur mises au point dans le domaine de l'analyse syntaxique. Il est ainsi possible d'identifier de façon précise quels phénomènes ou combinaisons de phénomènes font qu'un système se trompe. Les suites de tests proposées permettent d'évaluer les systèmes sur des implications basées uniquement sur la syntaxe, la sémantique lexicale et la sémantique compositionnelle, ou sur une combinaison de ces classes de phénomènes.

Pour générer nos suites de tests, nous utilisons dans un premier temps un réalisateur de surface qui permet de créer à partir d'une sémantique S des triplets (S, P, A), où S est la sémantique choisie, P est une phrase véhiculant cette sémantique et A un ensemble d'annotations syntaxiques qui liste les constructions syntaxiques utilisées pour créer la phrase P (section 3.2). Nous donnons au réalisateur syntaxique un ensemble de formules sémantiques et récupérons l'ensemble de triplets (S, P, A) correspondant. Pour chaque combinaison de triplets $((T_S, T_P, T_A), (H_S, H_P, H_A))$, tel que $T_P \neq H_P$, nous testons si $T_S \Rightarrow H_S$. Nous filtrons les cas de non-implication trop simples du genre « John eats a chicken » $\not\Rightarrow$ « Mary sends a mail ». Enfin, nous choisissons une taille de corpus ainsi que des contraintes de distribution des phénomènes et sélectionnons automatiquement un ensemble de problèmes permettant de respecter au mieux ces contraintes grâce à un algorithme de sélection itératif avec ajustement par pivot que nous avons développé. La méthode d'évaluation sera décrite dans le chapitre 5 qui lui est consacré.

3.2 Génération de phrases avec GenI

Pour générer notre banque de phrases, nous utilisons GenI [Gardent & Kow, 2007b], un réalisateur de surface permettant de générer des phrases à partir de formules sémantiques. Il utilise pour cela une grammaire d'arbres adjoints basée sur les traits (FTAG), augmentée avec un calcul compositionnel de la sémantique basé sur l'unification comme proposé dans [Gardent & Kallmeyer, 2003], permettant ainsi de dériver les phrases associées à une sémantique. Par exemple, à partir de l'entrée donnée en (11), GenI produira les variantes listées en (12).

(11) $l_0{:}a(B, l_1, h_0)$, $l_1{:}give(B, C, D, E)$, $l_1{:}passive(B)$, $l_2{:}the(C)$, $l_2{:}man(C)$, $l_3{:}the(D)$, $l_3{:}woman(D)$, $l_4{:}the(E)$, $l_4{:}flower(E)$, $l_5{:}a(M, l_6, l_0)$, $l_6{:}smell(M, E, N)$, $l_7{:}nice(N)$

(12) a. « The flower which smells nice is given to the woman by the man »
give :`(n0Vn1Pn2,Passive,{CanSubj, ToObj, ByAgt}),`
smell :`(n0V,active,{OvertSubjectRelative)})`

b. « The flower which smells nice is given the woman by the man »
give :`(n0Vn2n1,Passive,{})`,
smell :`(n0V,active,{OvertSubjectRelative})`

c. « The flower which is given the woman by the man smells nice »
give :`(n0Vn2n1,Passive,{CovertSubjectRelative})`,
smell :`(n0V,active,{})`

d. « The flower which is given to the woman by the man smells nice »
give :`(n0Vn1Pn2,Passive,{OvertSubjectRelative})`,
smell :`(n0V,active,{})}`

e. « The flower that smells nice is given to the woman by the man »
give :`(n0Vn1Pn2,Passive,{})`,
smell :`(n0V,Active,{CovertSubjectRelative})`

f. « The flower that smells nice is given the woman by the man »
give :`(n0Vn2n1,Passive,{})`,
smell :`(n0V,CovertSubjectRelative,{})`

g. « The flower that is given the woman by the man smells nice »
give :`(n0Vn2n1,Passive,{CovertSubjectRelative})`,
smell :`(n0V,active,{})`

h. « The flower that is given to the woman by the man smells nice »
give :`(n0Vn1Pn2,Passive,{CovertSubjectRelative})`,
smell :`(n0V,active,{})`

Comme la grammaire utilisée par GenI est factorisée, il est possible de connaitre les morceaux d'arbres utilisés pendant la génération d'une phrase afin d'utiliser ces informations comme annotations syntaxiques.

3.2.1 Grammaire Tag [12]

La grammaire utilisée dans GenI est une grammaire d'arbres adjoints lexicalisés basée sur l'unification (FLTAG, [Shanker & Joshi, 1988]). Une FLTAG comprend un ensemble d'arbres élémentaires et deux opérations permettant de combiner ces arbres entre eux, l'opération de substitution et l'opération d'adjonction. Les arbres résultant d'une de ces opérations sont appelés « arbres dérivés ».

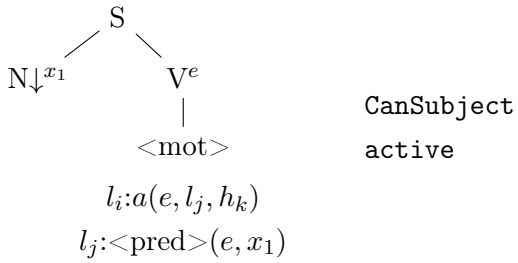
Les arbres élémentaires sont lexicalisés, c'est-à-dire qu'ils sont explicitement associés avec un lemme ou une forme fléchie. Leurs nœuds sont étiquetés par deux structures de traits appelées Top et Bottom. Un arbre élémentaire est soit initial, soit auxiliaire. Un arbre initial est un arbre dont les nœuds feuilles sont soit des nœuds terminaux, soit des nœuds dit de substitution (marqués par ↓). Un arbre

12. Description reprise en partie de l'article[Gardent & Kow, 2007a]

auxiliaire est un arbre dont l'un des nœuds feuilles est un nœud «pied» (marqué par $\star$) étiqueté par la même catégorie que le nœud racine.

L'opération de substitution permet d'insérer un arbre élémentaire ou dérivé τ_δ dans un arbre initial τ_α : le nœud racine de τ_δ est alors identifié avec un nœud de substitution dans τ_α et les traits Top sont unifiés ($\text{Top}_{\tau_\alpha} = \text{Top}_{\tau_\delta}$). L'opération d'adjonction permet d'insérer un arbre auxiliaire τ_β dans un arbre quelconque τ_α à un nœud n : les traits Top_n et Bottom_n du nœud n où se fait l'adjonction sont alors unifiés avec les traits Top du nœud racine de l'arbre auxiliaire et les traits Bottom de son nœud pied respectivement ($\text{Top}_n = \text{Top}_{Root_{\tau_\beta}}$ et $\text{Bottom}_n = \text{Bottom}_{Foot_{\tau_\beta}}$). En fin de dérivation, les traits Top et Bottom de chaque nœud de l'arbre dérivé produit sont unifiés.

Grammaire et méta-grammaire. La grammaire TAG utilisée est une grammaire produite par compilation à partir d'une spécification plus abstraite appelée, méta-grammaire. Dans cette méta-grammaire, un arbre élémentaire est défini par la combinaison d'un ou de plusieurs fragments d'arbre et chaque fragment d'arbre encapsule une caractéristique linguistique spécifique. Par exemple, tout arbre verbal dont le sujet est un sujet nominal canonique fera intervenir le fragment d'arbre `CanSubject`. Plus généralement, chaque arbre élémentaire est associé par le processus de compilation à un profil listant l'ensemble des noms de fragments d'arbre ayant participé à sa construction. Comme l'illustre le profil de l'arbre ci-dessous appartenant à la famille des arbres intransitifs (`n0V`), le profil de chaque arbre donne ainsi des informations sur ses caractéristiques linguistiques. Nous pouvons remarquer que cet arbre contient des trous au niveau de la lexicalisation (<mot>) et de la sémantique <pred>(x_1).



Les familles d'arbres existantes dans la grammaire de Alahverdzhieva [2008] sont les verbes transitifs (`n0Vn1`), intransitifs (`n0V`), bitransitifs (`n0Vn2n1`), transitifs avec complément prépositionnel (`n0Vn1Pn2`), intransitifs à complément phrastique (`n0Vs1`), les intransitifs à complément adjectivale (`n0Va1`) et les auxiliaires (`betaVv`). Les arbres appartenant à ces familles peuvent être sous l'une de ces trois formes : `active`, `betaVn` (gérondif), `passive`. Enfin les réalisations possibles des arguments sont les suivantes :`RelativeCovertSubject`, `RelativeOvertSubject`, `NPGerundSubjectPro`, `ProSubject`, `CanObject`, `NPGerundSubjectCan`,

`RelativeOvertIObject`, `CanIObject`, `CanByAgent`, `RelativeCovertIObject`, `CanSubject`, `RelativeCovertObject`, `RelativeOvertObject`.

Ces informations sont utilisées pour annoter les phrases générées. Nous annotons chaque prédicat de la phrase afin de connaitre sa famille, sa forme, et comment il réalise ses arguments. C'est ainsi que l'on obtient les annotations associées aux phrases générées dans l'exemple (12).

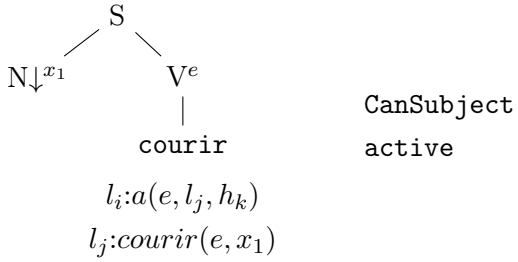
Lexiques de lemmes et morphologique Afin de permettre la génération de phrases, un lexique de lemmes et un lexique morphologique sont associés à la grammaire. Le lexique de lemmes est constitué d'entrées comprenant entre autres un lemme, un ensemble de familles d'arbres syntaxiques et des informations sémantiques permettant de remplir les trous des sémantiques associés aux arbres syntaxiques. Par exemple, l'entrée pour « courir » est :

```
lemme: courir, familles: n0V, sémantique: courir
```

Le lexique morphologique est constitué d'entrées comprenant un lemme, une structure de traits et la forme fléchie correspondante. La grammaire sera d'abord associée au lexique de lemmes afin d'obtenir des arbres lexicalisés, les arbres seront alors combinés pour satisfaire les contraintes de la grammaire et de la sémantique pour laquelle nous souhaitons générer des phrases. Pour finir les lemmes seront fléchis grâce au lexique morphologique. L'entrée pour la flexion est la suivante :

```
court, courir, [pos = v ; mode = ind ; pers = 3 ; num = sg]
```

Étant donné l'arbre vu précédemment et l'entrée du lexique de lemmes pour « courir », nous obtenons l'arbre suivant pour la grammaire lexicalisée :



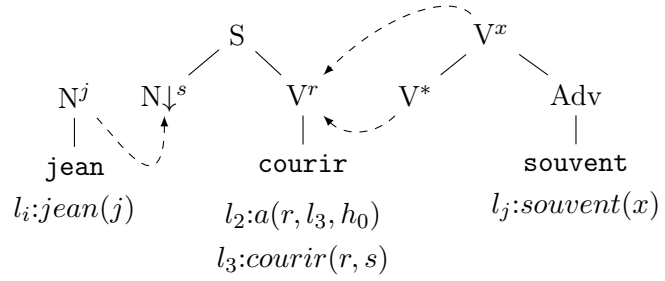
Le prédicat $a(e, l_j, h_k)$ permet de quantifier existentiellement l'évènement, car nous utilisons une représentation Davidsonienne.

3.2.2 Algorithme de génération

L'algorithme de base est un algorithme tabulaire ascendant [Kay, 1996] optimisé pour les grammaires d'arbres adjoints. Une spécification détaillée de l'algorithme et des optimisations déployées est donnée dans (Gardent & Kow, 2005). Nous nous contentons ici d'illustrer son fonctionnement par un exemple.

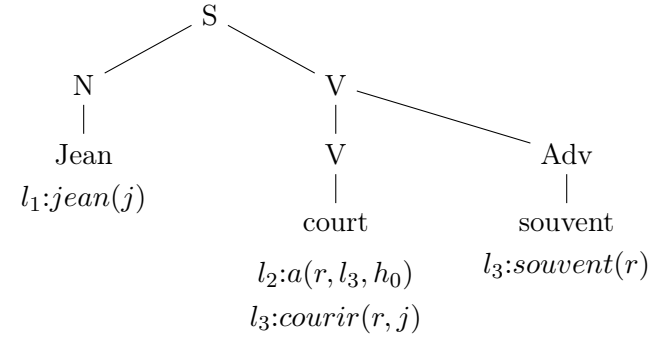
Supposons que la sémantique donnée en entrée soit $l_1{:}jean(j),\ l_2{:}a(r, l_3, h_0),\ l_3{:}courir(r, j),\ l_3{:}souvent(r)$. L'algorithme procède de la façon suivante. Dans un premier temps (phase de sélection lexicale), les arbres élémentaires dont la sémantique subsume une partie de l'entrée sont sélectionnés. Pour notre exemple, les arbres sélectionnés seront (entre autres) les arbres de `Jean`, `courir` et `souvent` (cf. Figure ci-dessous). La deuxième étape (phase de substitution) consiste à explorer systématiquement les possibilités de combinaisons par substitution. Pour l'exemple considéré, cette exploration permettra de substituer l'arbre pour `Jean` dans l'arbre pour `courir` (cf. Figure ci-dessous). La troisième étape (phase d'adjonction) permet de combiner les arbres produits par adjonction. C'est à ce stade que l'arbre pour `souvent` sera adjoint à l'arbre dérivé pour `Jean courir`. En dernier ressort (phase d'extraction), les chaînes étiquetant les items couvrant la sémantique donnée en entrée sont produites, en l'occurrence : `Jean courir souvent`.

$l_1{:}jean(j),\ l_2{:}a(r, l_3, h_0),\ l_3{:}courir(r, j),\ l_3{:}souvent(r) \Rightarrow$



Suit la phase de réalisation morphologique qui, en utilisant les entrées du lexique morphologique qui donnent pour les lemmes `courir`,`jean` et `souvent` les flexions « court », « Jean » et « souvent », génère l'arbre syntaxique ci-dessous correspondant à la phrase « Jean court souvent ».

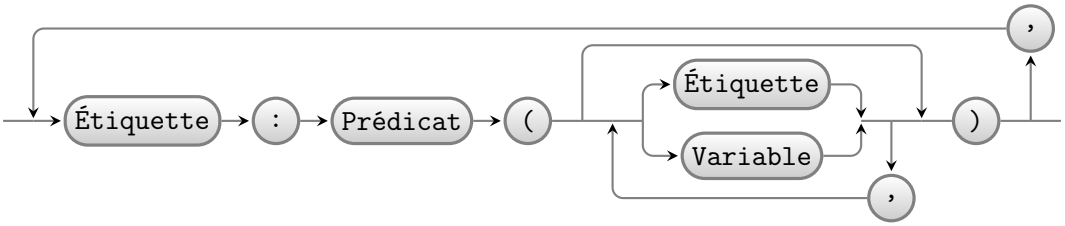
$l_1{:}jean(j),\ l_2{:}a(r, l_3, h_0),\ l_3{:}courir(r, j),\ l_3{:}souvent(r) \Rightarrow$



3.2.3 La sémantique plate

La représentation sémantique utilisée est une version simplifiée de la sémantique plate, comme défini dans [Copestake *et al.*, 2005]. Les formules contiennent en plus de

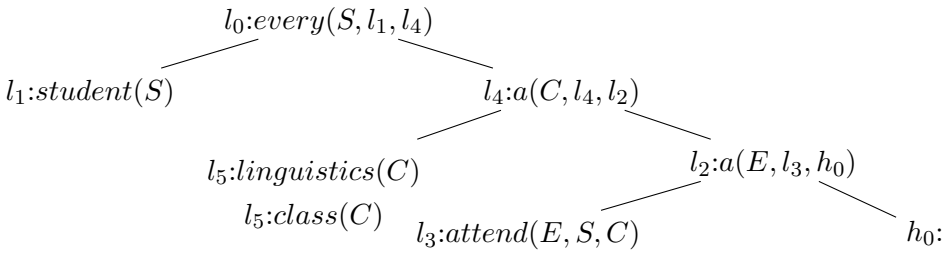
l'information sémantique un certain nombre de prédicats permettant de spécifier des informations de nature plus syntaxique comme la forme d'un verbe (e.g., actif/passif). La sémantique plate permet d'associer un ou plusieurs prédicats à un identifiant et d'avoir des prédicats ayant comme arguments des variables ou des identifiants, permettant ainsi de représenter les phénomènes de portée. La sémantique plate a la grammaire suivante :



Les étiquettes sont représentées par des chaines de caractères en minuscule et les variables par des chaines de caractères en majuscule. La sémantique plate a été utilisée car elle prévient des problèmes combinatoires apparaissant lors de la génération des structures récursives des lambda termes et des formules de logique du premier ordre. L'exemple de sémantique ci-dessous peut être associée à la phrase « Every student attends a linguistics class ».

$$l_0{:}every(S, l_1, l_4),\ l_1{:}student(S),\ l_2{:}a(E, l_3, h_0),\ l_3{:}attend(E, S, C),$$
$$l_4{:}a(C, l_5, l_2),\ l_5{:}linguistics(C),\ l_5{:}class(C)$$

Les étiquettes prises en argument d'un prédicat permettent de représenter la portée de ce prédicat sur d'autres prédicats. Avec cette information de portée entre les prédicats, il est alors possible de créer l'arbre de portée suivant pour la formule ci-dessus :



Pour générer la formule de logique du premier ordre correspondante à une formule de sémantique plate, il faut remplacer chaque étiquette en argument par l'ensemble de prédicats portant cette étiquette :

$$l_0{:}\{every(S, l_1{:}\{student(S)\},$$
$$l_3{:}\{a(C, l_4{:}\{linguistics(C), class(C)\},$$
$$l_2{:}\{a(E, l_3{:}\{attend(E, S, C)\}, h_0{:}\{\})\})\})\}$$

puis il faut appliquer la fonction de transformation τ définie ci-dessous :

$$
\begin{array}{lrcl}
1: \tau(& l{:}\{p_0,\ \ldots,\ p_n\} &) = & \tau(p_0) \wedge \ldots \wedge \tau(p_n) \\
2: \tau(& a(V,\ l_1{:}s1,\ l_2{:}s2)\} &) = & \exists V.\ \tau(l_1{:}f1) \ \wedge \ \tau(l_2{:}f2) \\
3: \tau(& a(V,\ l_1{:}s1,\ h_1{:}\{\})\} &) = & \exists V.\ \tau(l_1{:}f1) \\
4: \tau(& every(V,\ l_1{:}s1,\ l_2{:}s2) &) = & \forall V.\ \tau(l_1{:}s1) \ \Rightarrow \ \tau(l_2{:}s2) \\
5: \tau(& no(V,\ l_1{:}s1,\ l_2{:}s2) &) = & \forall V.\ \tau(l_1{:}s1) \ \Rightarrow \ \neg\tau(l_2{:}s2) \\
6: \tau(& notevery(V,\ l_1{:}s1,\ l_2{:}s2) &) = & \neg\exists V.\ \tau(l_1{:}s1) \ \Rightarrow \ \tau(l_2{:}s2) \\
7: \tau(& pred(arg_0,\ \ldots,\ arg_n) &) = & pred(arg_0,\ \ldots,\ arg_n)
\end{array}
$$

En appliquant la fonction de transformation τ sur la formule vue ci-dessus nous obtenons la suite de transformations suivantes :

$$
\begin{array}{ll}
 & \tau(l_0{:}\{every(S, l_1{:}\{student(S)\}, l_3{:}\{a(C, l_4{:}\{linguistics(C), class(C)\}, l_2{:}\{a(E, l_3{:}\{attend(E,S,C)\}, h_0{:}\{\})\})\})\}) \\
\vdash_1 & \tau(every(S, l_1{:}\{student(S)\}, l_3{:}\{a(C, l_4{:}\{linguistics(C), class(C)\}, l_2{:}\{a(E, l_3{:}\{attend(E,S,C)\}, h_0{:}\{\})\})\})\}) \\
\vdash_4 & \forall S.(\tau(l_1{:}\{student(S)\}) \Rightarrow, \tau(l_3{:}\{a(C, l_4{:}\{linguistics(C), class(C)\}, l_2{:}\{a(E, l_3{:}\{attend(E,S,C)\}, h_0{:}\{\})\})\})) \\
\vdash_1 & \forall S.(\tau(student(S)) \Rightarrow, \tau(l_3{:}\{a(C, l_4{:}\{linguistics(C), class(C)\}, l_2{:}\{a(E, l_3{:}\{attend(E,S,C)\}, h_0{:}\{\})\})\})) \\
\vdash_7 & \forall S.(student(S) \Rightarrow, \tau(l_3{:}\{a(C, l_4{:}\{linguistics(C), class(C)\}, l_2{:}\{a(E, l_3{:}\{attend(E,S,C)\}, h_0{:}\{\})\})\})) \\
\vdash_1 & \forall S.(student(S) \Rightarrow, \tau(a(C, l_4{:}\{linguistics(C), class(C)\}, l_2{:}\{a(E, l_3{:}\{attend(E,S,C)\}, h_0{:}\{\})\})) \\
\vdash_2 & \forall S.(student(S) \Rightarrow, \exists C.(\tau(l_4{:}\{linguistics(C), class(C)\}) \wedge \tau(l_2{:}\{a(E, l_3{:}\{attend(E,S,C)\}, h_0{:}\{\})\}))) \\
\vdash_1 & \forall S.(student(S) \Rightarrow, \exists C.((\tau(linguistics(C)) \wedge \tau(class(C))) \wedge \tau(l_2{:}\{a(E, l_3{:}\{attend(E,S,C)\}, h_0{:}\{\})\}))) \\
\vdash_7 & \forall S.(student(S) \Rightarrow, \exists C.((linguistics(C) \wedge \tau(class(C))) \wedge \tau(l_2{:}\{a(E, l_3{:}\{attend(E,S,C)\}, h_0{:}\{\})\}))) \\
\vdash_7 & \forall S.(student(S) \Rightarrow, \exists C.((linguistics(C) \wedge class(C)) \wedge \tau(l_2{:}\{a(E, l_3{:}\{attend(E,S,C)\}, h_0{:}\{\})\}))) \\
\vdash_1 & \forall S.(student(S) \Rightarrow, \exists C.((linguistics(C) \wedge class(C)) \wedge \tau(a(E, l_3{:}\{attend(E,S,C), h_0{:}\{\})\}))) \\
\vdash_3 & \forall S.(student(S) \Rightarrow, \exists C.((linguistics(C) \wedge class(C)) \wedge \exists E.(\tau(l_3{:}\{attend(E,S,C)\})))) \\
\vdash_1 & \forall S.(student(S) \Rightarrow, \exists C.((linguistics(C) \wedge class(C)) \wedge \exists E.(\tau(attend(E,S,C))))) \\
\vdash_7 & \forall S.(student(S) \Rightarrow, \exists C.((linguistics(C) \wedge class(C)) \wedge \exists E.(attend(E,S,C))))
\end{array}
$$

Nous pouvons ainsi appliquer des outils de preuves permettant de tester l'implication logique entre la traduction en logique du premier ordre des sémantiques plates associées à deux phrases.

3.3 Génération de problèmes de RTE

Nous venons de présenter comment générer à partir d'une formule en sémantique plate un ensemble de paraphrases et la formule de logique du premier ordre correspondante. Nous allons maintenant voir comment nous utilisons cela pour générer des problèmes de détection d'implications textuelles annotés automatiquement avec les variations de propriétés syntaxique et sémantique existantes entre les deux phrases de chaque problème. Dans un premier temps nous allons montrer comment générer des problèmes reproduisant uniquement des variations syntaxiques, puis nous ajouterons des variations basées sur la sémantique lexicale, les quantificateurs, et enfin les prédicats implicatifs.

3.3.1 Implications basées sur la syntaxe

Pour générer une suites de tests fondée sur les variations syntaxiques, nous avons spécifié manuellement un ensemble de formules permettant de couvrir des variations sémantiques ayant des effets spécifiques lors de la réalisation syntaxique. Par exemple, nous avons créé des formules contenant un prédicat avec un, deux ou trois de ses arguments pour obtenir des phrases contenant ce prédicat à l'intransifs, au transitifs, et au ditransitif. Nous avons choisi des sémantiques où l'on inversait les arguments d'un prédicat (e.g., « Jean aime Marie » et « Marie aime Jean » pour leurrer les systèmes utilisant les techniques de chevauchement de mots, ainsi que des prédicats ayant des particularités de réalisation syntaxique comme les ergatifs. Chaque formule contient un maximum de deux propositions.

Nous avons conçu 81 formules qui ont généré 226 phrases annotées. Nous avons ensuite créé tous les couples possibles de phrases que nous avons annotés en tant qu'implication ou que non-implication selon que la sémantique associée à l'une implique la sémantique associée à l'autre. Nous avons ainsi obtenu un ensemble de 2720 implications (e.g, le problème (13)) et de 3666 non-implications (e.g, le problème (14)). La suite de tests ainsi obtenue contient un très grand nombre de non-implications dont la majeure partie sont triviales à résoudre (e.g., « John send Mary a book » $\not\Rightarrow$ « Kevin loves Mary »). Nous avons donc filtré les couples de non-implications ayant un chevauchement de mots trop faible car nous les considérons trop simples.

(13) **T :** « John sends Kevin a mail »
SP : $l_0{:}john(J),\ l_1{:}a(S,l_1,h_0),\ l_2{:}send(S,J,M,K),\ l_3{:}kevin(K),\ l_4{:}a(M,l_5,l_1),\ l_5{:}mail(M)$
LPO : $john(J) \wedge kevin(K) \wedge \exists M.(mail(m) \wedge \exists S.send(S,J,M,K))$
ANNOTATIONS SYNTAXIQUES :
send :{(`n0Vn2n1`, `active`, `CanSubject`),
(`n0Vn2n1`, `active`, `CanObject`)}
H : « A mail is sent to Kevin »
SP : $l_0{:}a(M,l_1,h_0),\ l_1{:}mail(M),\ l_2{:}a(S,l_3,l_4),\ l_3{:}send(S,\_,M,K),\ l_5{:}kevin(K)$
LPO : $kevin(K) \wedge \exists M.(mail(m) \wedge \exists S.(\exists U.send(S,U,M,K)))$
ANNOTATIONS SYNTAXIQUES :
send :{(`n0Vn2n1`, `active`, `CanSubject`),
(`n0Vn2n1`, `active`, `CanObject`)}
Implication : OUI

(14) **T :** « John sinks a boat »
SP : $l_0{:}john(J),\ l_1{:}a(S,l_2,h_0),\ l_2{:}sink(S,J,B),\ l_3{:}a(B,l_4,l_1),\ l_4{:}boat(B)$
LPO : $john(J) \wedge \exists B.(boat(B) \wedge \exists S.sink(S,J,B))$
ANNOTATIONS SYNTAXIQUES :

sink :{(n0Vn1, active, CanSubject),
(n0Vn1, active, CanObject)}

H : « John sinks »

SP : l_0:$john(J)$, l_1:$a(S, l_2, h_0)$, l_2:$sink(S, \_, J)$

LPO : $john(J) \wedge \exists S.(\exists U.sink(S, U, J))$

ANNOTATIONS SYNTAXIQUES :

sink :{(n0Vn1, active, CanSubject),
(n0Vn1, active, CanObject)}

Implication : NON

(15) **T :** « John loves a woman who dances »

SP : l_0:$john(J)$, l_1:$a(L, l_2, h_0)$, l_2:$love(L, J, W)$, l_3:$a(W, l_4, l_5)$, l_4:$woman(W)$, l_5:$a(D, l_6, l_1)$, l_6:$dance(D, W)$

LPO : $john(J) \wedge \exists W.(woman(w) \wedge \exists D.dance(D, W) \wedge \exists L.love(L, J, W))$

ANNOTATIONS SYNTAXIQUES :

love :{(n0Vn1, active, CanSubject),
(n0Vn1, active, CanObject)}

dance :{(n0Vn1, active, CanSubject),
{(n0Vn1, active, CanObject)}

H : « A woman who dances is loved by John »

SP : l_0:$john(J)$, l_1:$a(L, l_2, h_0)$, l_2:$love(L, J, W)$, l_2:$passive(L)$, l_3:$a(W, l_4, l_5)$, l_4:$woman(W)$, l_5:$a(D, l_6, l_1)$, l_6:$dance(D, W)$

LPO : $john(J) \wedge \exists W.(woman(w) \wedge \exists D.dance(D, W) \wedge \exists L.love(L, J, W))$

ANNOTATIONS SYNTAXIQUES :

love :{(n0Vn1, active, CanSubject),
(n0Vn1, active, CanObject)}

dance :{(n0Vn1, active, CanSubject),
(n0Vn1, active, CanObject)}

Implication : NON

3.3.2 Implications basées sur la sémantique lexicale

Il faut distinguer deux types de relation pour la synonymie et l'antonymie binaire, les relations lexicales (*violon/fiddle*) et les relations phrastiques (*To die/ To pass away/ To kick the bucket*). Le premier type de relation est traité par la partie morphologique du processus de réalisation de surface, et le second est traité par le processus de combinaison syntaxique.

Synonymie phrastique Comme nous l'avons vu précédemment, la première phase de GenI est la sélection lexicale qui choisit des arbres élémentaires dont la sémantique subsume une partie de la sémantique en entrée. Pour pouvoir générer

des variations contenant des synonymies phrastiques, il suffit donc d'associer la même sémantique aux expressions ayant le même sens. Par exemple, nous associons aux expressions *« pass away »*/ *« kick the bucket »* les entrées *(**pass_away**,* $l_i{:}a(E, l_j, h_0)$, $l_j{:}die(E, X)$, *n0V)*, *(**kick_the_bucket**,* $l_i{:}a(E, l_j, h_0)$, $l_j{:}die(E, X)$, *n0V)* ayant la même sémantique que l'entrée de *die* : *(**die**,* $l_i{:}a(E, l_j, h_0)$, $l_j{:}die(E)$, *n0V)*. Ainsi la sémantique (16) sera associée entre autre par GenI aux trois phrases (17).

(16) $l_1{:}a(e, l_2, h_0)$, $l_2{:}die(e, j)$, $l_2{:}past(e)$, $l_3{:}john(j)$

(17) a. « John died »

b. « John passed away »

c. « John kicked the bucket »

Synonymie lexicale Bien qu'il permette de produire des variations synonymiques, le traitement des synonymes phrastiques augmente la complexité du calcul. Comme la réalisation de surface à partir de grammaire réversible est connue pour être NP complète, il est préférable d'éviter d'augmenter la complexité quand cela est possible. C'est pourquoi dans GenI, la synonymie lexicale est gérée dans la phase de réalisation morphologique plutôt que dans la phase de réalisation syntaxique. À cette étape, les lemmes désambigüisés étiquetant les feuilles de l'arbre d'une phrase et leur structure de traits sont fléchis par le réalisateur morphologique qui va extraire du lexique morphologique, la (ou les) combinaisons *(lemme, structure de trait)* adéquates pour produire la forme correspondante. Par exemple, étant donné le lemme ***die*** et la structure de traits `[cat = v; mode=ppart]`, le réalisateur morphologique va remplacer le lemme ***die*** par la forme *« died »*. Il est important de noter que le lemme stocké dans le lexique morphologique est un lemme désambigüisé. Par exemple, il y a deux entrées `bank_1` et `bank_2` pour le mot *« bank »*, une pour le sens d'institution financière et l'autre pour la rive.

En associant un lemme désambigüisé à plusieurs synonymes lexicaux, les variantes basées sur la synonymie lexicale peuvent être générées. Par exemple, si le lexique morphologique contient les entrée suivantes :

(`die`, `[cat = v; mode=ppart]`) : « died »
(`die`, `[cat = v; mode=ppart]`) : « perished »

la sémantique $l_1{:}a(e, l_2, h_0)$, $l_2{:}die(e, j)$, $l_2{:}past(e)$, $l_3{:}john(j)$, mise en entrée de GenI, donnera les phrases suivantes :

(18) a. « John died »

b. « John perished »

Antonymie lexicale Pour les antonymes binaires, nous associons la même sémantique modulo une négation. Cela permet, par exemple, de générer *« The earth is dry »/« The earth is not wet »* à partir de la même sémantique et d'obtenir ainsi les deux couples d'implication suivantes :

(19) **T :** « The earth is dry »
SP : l_1:$the(X)$, l_1:$earth(X)$, l_2:$dry(X)$
H : « The earth is wet »
SP : l_1:$the(X)$, l_1:$earth(X)$, l_2:$dry(X)$, l_3:$not(l_2)$
Implication : NON
ANNOTATIONS LEXICALES : {antonyme lexical}

(20) **T :** « The earth is dry »
SP : l_1:$the(X)$, l_1:$earth(X)$, l_2:$dry(X)$
H : « The earth is not wet »
SP : l_1:$the(X)$, l_1:$earth(X)$, l_2:$dry(X)$, l_3:$not(l_2)$, $l_4 : not(l_3)$
Implication : OUI
ANNOTATIONS LEXICALES : {antonyme lexical}

Nous ajoutons comme annotations de sémantique lexicale, le type de relation lexicale pouvant exister entre le texte et l'hypothèse. Si deux entrées morphologiques ont la même sémantique, et la même structure de trait mais diffèrent sur la flexion, on ajoute une annotation signifiant qu'il peut y avoir une relation de synonymie lexicale. Si deux arbres utilisés on la même sémantique associée mais utilisent des lemmes différents, on ajoute une annotation signifiant qu'il peut y avoir une relation de synonymie phrastique. Le problème de cette méthode d'annotation est qu'il n'est pas possible de savoir si la relation lexicale qui existe entre le texte et l'hypothèse est la cause de la réussite ou de l'échec de l'implication.

Par exemple, dans l'implication (21), il y a deux relations lexicales entre T et H, un lien d'hyperonymie entre « domestic animal » et « mouse », ainsi qu'un lien de synonymie entre « pet » et « domestic animal », mais seule la relation de synonymie participe à l'implication. Il est tout de même intéressant de savoir quelles relations lexicales apparaissent dans un problème sans être nécessaires à la reconnaissance de l'implication textuelle, pour permettre de savoir si ces occurrences de relations lexicales perturbent le système.

(21) **T :** « John feeds his pet with a mouse »
H : « John feeds his domestic animal »
Implication : OUI
ANNOTATIONS LEXICALES : {synonymie lexicale, hyperonymie}

Une solution possible pour trouver l'ensemble de connaissances minimal permettant d'obtenir la bonne réponse peut être trouvé dans les systèmes d'ontologie. En

effet, ces systèmes qui permettent de représenter des données à l'aide de logiques de descriptions ont intégré un algorithme [Horridge *et al.*, 2008] permettant au prouveur de justifier (c.f., définition 3.3.2.1) sa réponse.

Définition 3.3.2.1 (Justification) *Une justification pour une implication dans une ontologie est un ensemble minimal d'axiomes de l'ontologie qui sont suffisant pour qu'il y ait implication. L'ensemble est minimal s'il n'existe pas de sous-ensemble de la justification permettant de valider l'implication. Il peut y avoir plusieurs solutions, pouvant se chevaucher, pour une implication donnée.*

Nous pourrions ainsi savoir quelles connaissances ont été nécessaires à la validation d'une implication ou d'une contradiction. Notons que pour le cas des non-implications qui ne sont pas des contradictions, il n'est pas possible de justifier pourquoi car une infinité de justifications seraient possibles. En utilisant la justification, il serait imaginable de vérifier si les systèmes répondent bien pour la bonne raison.

Pour le moment nous annotons les problèmes avec toutes les relations lexicales existantes entre les mots des deux phrases qu'elles soient nécessaires pour trouver la réponse ou non.

3.3.3 Implications basées sur la sémantique computationnelle

Les quantificateurs

Les quantificateurs donnent une information sur la quantité de l'entité associée et influent sur la monotonicité de celle-ci ainsi que de la monotonicité du morceau de phrase sur lequel il a porté. Cela permet ainsi d'obtenir des implications comme (22) ainsi que des non-implications comme (23).

(22) **T :** « Every student attends a linguistic class »
SP : $l_0{:}a(E, l_1, h_0),\ l_1{:}attend(E, S, C),\ l_2{:}every(S, l_3, l_4),\ l_3{:}student(S),$
$l_4{:}a(C, l_5, l_0),\ l_5{:}linguistics(C),\ l_5{:}class(C)$
LPO : $\forall S.(student(S) \Rightarrow (\exists C.((linguistics(C) \wedge class(C)) \wedge (\exists E.(attend(E, S, C))))))$
ANNOTATIONS QUANTIFICATEURS :
{`every:rest:bottom`, `every:scope:a`, `a:rest:bottom`, `a:scope:bottom`}
H : « Every tall student attends a linguistic class »
SP : $l_0{:}a(E, l_1, h_0),\ l_1{:}attend(E, S, C),\ l_2{:}every(S, l_3, l_4),\ l_3{:}tall(S),$
$l_3{:}student(S),\ l_4{:}a(C, l_5, l_0),\ l_5{:}linguistics(C),\ l_5{:}class(C)$
LPO : $\forall S.((tall(S) \wedge student(S)) \Rightarrow (\exists C.((linguistics(C) \wedge class(C)) \wedge (\exists E.(attend(E, S, C))))))$
ANNOTATIONS QUANTIFICATEURS :
{`every:rest:bottom`, `every:scope:a`, `a:rest:bottom`, `a:scope:bottom`}
Implication : Oui

(23) **T :** « A student attends a linguistic class »

SP : $l_0{:}a(E, l_1, h_0),\ l_1{:}attend(E, S, C),\ l_2{:}a(S, l_3, l_4),\ l_3{:}student(S),$
$l_4{:}a(C, l_5, l_0),\ l_5{:}linguistics(C),\ l_5{:}class(C)$

LPO : $\exists S.(student(S) \Rightarrow (\exists C.((linguistics(C) \wedge class(C)) \wedge (\exists E.(attend(E, S, C))))))$

ANNOTATIONS QUANTIFICATEURS :

{`a:rest:bottom`, `a:scope:a`, `a:rest:bottom`, `a:scope:bottom`}

H : « A tall student attends a linguistic class »

SP : $l_0{:}a(E, l_1, h_0),\ l_1{:}attend(E, S, C),\ l_2{:}a(S, l_3, l_4),\ l_3{:}tall(S),\ l_3{:}student(S),$
$l_4{:}a(C, l_5, l_0),\ l_5{:}linguistics(C),\ l_5{:}class(C)$

LPO : $\exists S.((tall(S) \wedge student(S)) \Rightarrow (\exists C.((linguistics(C) \wedge class(C)) \wedge (\exists E.(attend(E, S, C))))))$

ANNOTATIONS QUANTIFICATEURS :

{`a:rest:bottom`, `a:scope:a`, `a:rest:bottom`, `a:scope:bottom`}

Implication : NON

(24) **T :** « A researcher presended a linguistic class which was attended by every student »

SP : $l_0{:}a(R, l_1, l_4),\ l_1{:}researcher(R),\ l_2{:}a(P, l_3, h_0),\ l_3{:}present(P, R, C),$
$l_4{:}a(C, l_5, l_0),\ l_5{:}relative(l_6, l_9),\ l_6{:}linguistics(C),\ l_6{:}class(C),$
$l_7{:}a(P, l_8, l_2),\ l_8{:}attended(A, S, C),\ l_9{:}every(S, l_{10}, l_7),\ l_{10}{:}student(S)$

LPO : $\exists R.(researcher(S) \wedge \exists C.((linguistics(C) \wedge class(C)) \wedge \forall S.(student(S) \Rightarrow \exists A.attend(A, S, C)) \wedge \exists P.attend(P, R, C)))$

ANNOTATIONS QUANTIFICATEURS :

{`a:rest:bottom`, `a:scope:a`, `a:rest:every`, `a:scope:bottom`,
{`every:rest:bottom`, `every:scope:bottom`}

H : « Every student attended a linguistic class which was presented by a researcher »

SP : $l_0{:}every(S, l_1, l_4),\ l_1{:}student(S),\ l_2{:}a(A, l_3, h_0),\ l_3{:}attend(A, S, C),$
$l_4{:}a(C, l_5, l_0),\ l_5{:}relative(l_6, l_9),\ l_6{:}linguistics(C),\ l_6{:}class(C),$
$l_7{:}a(P, l_8, l_2),\ l_8{:}present(P, R, C),\ l_9{:}a(R, l_{10}, l_7),\ l_{10}{:}researcher(R)$

LPO : $\forall S.(student(S) \Rightarrow \exists C.((linguistics(C) \wedge class(C)) \wedge \exists R.(researcher(R) \wedge \exists P.present(P, R, C)) \wedge \exists A.attend(A, S, C)))$

ANNOTATIONS QUANTIFICATEURS :

{`every:rest:bottom`, `every:scope:a`, `a:rest:a`, `a:scope:bottom`,
{`a:rest:bottom`, `a:scope:bottom`}

Implication : OUI

Comme nous l'avons vu dans la section précédente, nous représentons les quantificateurs sous la forme de prédicat à trois arguments *quant(variable, restriction, portée)*. Le premier argument représente la variable quantifiée, le second sa restriction et le troisième sa portée. Nous ne gérons pour le moment que les quantificateurs

« a », « every », « no », « not every » et leurs synonymes. La transformation définie par la fonction τ est suffisante pour gérer l'implication avec les quantificateurs, car les monotonicités de la restriction et de la portée sont les mêmes dans les formules de logique du premier ordre obtenues. Un fragment de texte a une monotonicité descendante quand il implique tout ce qui est plus spécifique et une monotonicité ascendante quand il implique tout ce qui est plus général.

Par exemple, le quantificateur « every » a une monotonicité descendante sur sa restriction et une monotonicité ascendante sur sa portée. C'est pour cela que la phrase (25) implique la phrase (26) car la restriction est plus spécifique et la phrase (29) car la portée est plus générale, mais n'implique pas les phrases (28) et (27).

(25) « Every big cat eats a big mouse »

SP : $l_0{:}every(C, l_1, l_4),\ l_1{:}big(C),\ l_1{:}cat(C),\ l_2{:}a(E, l_3, h_0),\ l_3{:}eat(E, C, M),$ $l_4{:}a(M, l_5, l_2),\ l_5{:}big(M),\ l_5{:}mouse(M)$

FOL : $\forall C.((big(C) \wedge cat(C)) \Rightarrow (\exists M.((big(M) \wedge mouse(M)) \wedge \exists E.eat(E, C, M))))$

(26) « Every big ugly cat eats a big mouse »

SP : $l_0{:}every(C, l_1, l_4),\ l_1{:}big(C),\ l_1{:}ugly(C),\ l_1{:}cat(C),\ l_2{:}a(E, l_3, h_0),$ $l_3{:}eat(E, C, M),\ l_4{:}a(M, l_5, l_2),\ l_5{:}big(M),\ l_5{:}mouse(M)$

FOL : $\forall C.((big(C) \wedge ugly(C) \wedge cat(C)) \Rightarrow (\exists M.((big(M) \wedge mouse(M)) \wedge \exists E.eat(E, C, M))))$

(27) « Every big cat eats a big ugly mouse »

SP : $l_0{:}every(C, l_1, l_4),\ l_1{:}big(C),\ l_1{:}cat(C),\ l_2{:}a(E, l_3, h_0),\ l_3{:}eat(E, C, M),$ $l_4{:}a(M, l_5, l_2),\ l_5{:}big(M),\ l_5{:}ugly(M),\ l_5{:}mouse(M)$

FOL : $\forall C.((big(C) \wedge cat(C)) \Rightarrow (\exists M.((big(M) \wedge ugly(M) \wedge mouse(M)) \wedge \exists E.eat(E, C, M))))$

(28) « Every cat eats a big mouse »

SP : $l_0{:}every(C, l_1, l_4),\ l_1{:}cat(C),\ l_2{:}a(E, l_3, h_0),\ l_3{:}eat(E, C, M),$ $l_4{:}a(M, l_5, l_2),\ l_5{:}big(M),\ l_5{:}mouse(M)$

FOL : $\forall C.(cat(C) \Rightarrow (\exists M.((big(M) \wedge mouse(M)) \wedge \exists E.eat(E, C, M))))$

(29) « Every big cat eats a mouse »

SP : $l_0{:}every(C, l_1, l_4),\ l_1{:}big(C),\ l_1{:}cat(C),\ l_2{:}a(E, l_3, h_0),\ l_3{:}eat(E, C, M),$ $l_4{:}a(M, l_5, l_2),\ l_5{:}mouse(M)$

FOL : $\forall C.((big(C) \wedge cat(C)) \Rightarrow (\exists M.(mouse(M) \wedge \exists E.eat(E, C, M))))$

Il est difficilement possible de savoir étant donné deux phrases si la monotonicité est nécessaire pour détecter l'implication. En effet, il faudrait aligner les fragments d'information quantifiés des deux phrases pour savoir si le quantificateur change et s'il existe une relation d'implication ou de contradiction entre les fragments d'information alignés. Une solution pour avoir ce type d'information serait de générer automatiquement des couples de formules sémantiques alignés. Dans un premier temps

nous avons décidé de garder comme annotation les enchâssements de quantificateurs. C'est ainsi que l'on obtient les annotations des problèmes (22), (23) et (24), nous permettant par exemple de savoir qu'un quantificateur existentiel est enchâssé dans la portée d'un quantificateur universel.

Les prédicats implicatifs

Les prédicats implicatifs sont des verbes à complément phrastique qui impliquent ou non selon le contexte, la véridicité de ce complément. Nairn *et al.* [2006] ont modélisé ce phénomène dans le but de détecter les implications textuelles. Pour bien comprendre ce phénomène le mieux est de regarder plusieurs exemples. Dans les exemples (30) et (31), le prédicat implicatif « manage » implique son complément indépendamment du contexte. Dans l'exemple (30), le prédicat « manage » implique son complément dans un contexte positif et dans l'exemple (31) le même prédicat implique son complément dans un contexte négatif. Les exemples (32) et (33), montrent que le prédicat « refuse » implique son complément dans un contexte positif, mais devient neutre [13] dans un contexte négatif. Le dernier exemple (36) montre que le prédicat « want » est neutre dans tous les contextes. Le prédicat l_i:$pol(P)$ permet d'associé la polarité P à l'ensemble des prédicats portant l'étiquette l_i.

(30) **T :** « Bill manages to leave »
SP: l_0:$equals(PR, true)$, l_1:$bill(C)$, l_2:$a(R, l_3, h_0)$, l_3:$manage(R, C, L)$, l_3:$pol(PR)$, l_3:$equals(PL, PR)$, l_4:$comp(L)$, l_5:$a(L, l_6, l_1)$, l_6:$leave(L, C)$, l_6:$pol(PL)$
ANNOTATIONS IMPLICATIFS :
{`top:both-equal`, `both-equal:bottom`}
H : « Bill leaves »
SP: l_0:$equals(PL, true)$, l_1:$bill(C)$, l_2:$a(L, l_3, h_0)$, l_3:$leave(L, C)$, l_3:$pol(PL)$
ANNOTATIONS IMPLICATIFS :
{`top:bottom`}
Implication : OUI

(31) **T :** « Bill did not manage to leave »
SP: l_0:$equals(CO, true)$, l_1:$bill(C)$, l_2:$inv(PR, CO)$, l_3:$a(R, l_4, h_0)$, l_4:$past(R)$, l_4:$manage(R, C, L)$, l_4:$pol(PR)$, l_4:$equals(PL, PR)$, l_5:$comp(L)$, l_5:$a(L, l_7, l_1)$, l_7:$leave(L, C)$, l_7:$pol(PL)$
ANNOTATIONS IMPLICATIFS :
{`top:both-inv`, `both-inv:both-equal`, `both-equal:bottom`}
H : « Bill did not leave »

13. Être neutre signifie qu'il n'implique ni le complément, ni la négation de celui-ci, ainsi « Bill did not refuse to leave » n'implique ni « Bill leaves », ni « Bill did not leave ».

SP: l_0:$equals(CO, true)$, l_1:$bill(C)$, l_2:$inv(PL, CO)$, l_3:$a(L, l_4, h_0)$, l_4:$past(L)$, l_4:$leave(L, C)$, l_4:$pol(PL)$

ANNOTATIONS IMPLICATIFS :

{`top:both-inv`, `both-inv:bottom`}

Implication : OUI

(32) **T :** « Bill refused to leave »

SP: l_0:$equals(PR, true)$, l_1:$bill(C)$, l_2:$a(R, l_3, h_0)$, l_3:$refuse(R, C, L)$, l_3:$pol(PR)$, l_3:$t2t$:(PT, PR), l_3:inv:(PL, PT), l_4:$comp(L)$, $l5$:$a(L, l_6, l_2)$, l_6:$leave(L, C)$, l_6:$pol(PL)$

ANNOTATIONS IMPLICATIFS :

{`top:pos-inv`, `pos-inv:bottom`}

H : « Bill did not leave »

SP: l_0:$equals(CO, true)$, l_1:$bill(C)$, l_2:$inv(PL, CO)$, l_3:$a(L, l_4, h_0)$, l_4:$past(L)$, l_4:$leave(L, C)$, l_4:$pol(PL)$

ANNOTATIONS IMPLICATIFS :

{`top:both-inv`, `both-inv:bottom`}

Implication : OUI

(33) **T :** « Bill did not refuse to leave »

SP: l_0:$equals(CO, true)$, l_1:$bill(C)$, l_2:$inv(PR, CO)$, l_3:$a(R, l_4, h_0)$, l_4:$refuse(R, C, L)$, l_4:$pol(PR)$, l_4:$t2t$:(PT, PR), l_4:inv:(PL, PT), l_5:$comp(L)$, $l6$:$a(L, l_7, l_3)$, l_7:$leave(L, C)$, l_7:$pol(PL)$

ANNOTATIONS IMPLICATIFS :

{`top:both-inv`, `both-inv:pos-inv`, `pos-inv:bottom`}

H : « Bill leaves »

SP: l_0:$equals(PL, true)$, l_1:$bill(C)$, l_2:$a(L, l_3, h_0)$, l_3:$leave(L, C)$, l_3:$pol(PL)$

ANNOTATIONS IMPLICATIFS :

{`top:bottom`}

Implication : NON

(34) **T :** « Bill didn't forget that the door was open »

SP: l_0:$equals(CO, true)$, l_1:$bill(C)$, l_2:$inv(PR, CO)$, l_3:$a(R, l_4, h_0)$, l_4:$forget(R, C, L)$, l_4:$pol(PR)$, l_4:$equals$:$(PL, true)$, l_5:$comp(L)$, l_6:$the(D)$, l_6:$door(D)$, l_7:$a(L, l_8, l_3)$, l_8:$open(L, D)$, l_8:$pol(PL)$

ANNOTATIONS IMPLICATIFS :

{`top:both-inv`, `both-inv:both-pos`, `both-pos:bottom`}

H : « The door was open »

SP: l_0:$equals(PL, true)$, l_1:$the(D)$, l_1:$door(D)$ l_2:$a(L, l_3, h_0)$, l_3:$open(L, P)$, l_3:$pol(PL)$

ANNOTATIONS IMPLICATIFS :

{`top:bottom`}

Implication : OUI

(35) **T :** « Bill wants to forget that Ed left »

SP: $l_0{:}equals(PW, true),\ l_1{:}bill(C),\ l_2{:}a(W, l_3, h_0),\ l_3{:}want(W, C, F),$ $l_3{:}pol(PW),\ l_3{:}equals{:}(PF, indef),\ l_4{:}comp(F),\ l_5{:}a(F, l_6, l_2),$ $l_6{:}forget(F, C, O),\ l_6{:}pol(PF),\ l_6{:}equals{:}(PO, true),\ l_7{:}comp(O),$ $l_8{:}ed(E),\ l_9{:}a(O, l_{10}, l_5),\ l_{10}{:}open(O, E),\ l_{10}{:}pol(PO)$

ANNOTATIONS IMPLICATIFS :

{`top:both-indef`, `both-indef:both-pos`, `both-pos:bottom`}

H : « Ed left »

SP: $l_0{:}equals(PO, true),\ l_1{:}ed(E)\ l_2{:}a(O, l_3, h_0),\ l_3{:}open(O, E),\ l_3{:}pol(PO)$

ANNOTATIONS IMPLICATIFS :

{`top:bottom`}

Implication : OUI

(36) **T :** « Bill wants to leave »

SP: $l_0{:}equals(PR, true),\ l_1{:}bill(C),\ l_2{:}a(R, l_3, h_0),\ l_3{:}want(R, C, L),$ $l_3{:}pol(PR),\ l_3{:}equals{:}(PL, indef),\ l_4{:}comp(L),\ l_5{:}a(L, l_6, l_2),$ $l_6{:}leave(L, C),\ l_6{:}pol(PL)$

ANNOTATIONS IMPLICATIFS :

{`top:both-indef`, `both-indef:bottom`}

H : « Bill leaves »

SP: $l_0{:}equals(PL, true),\ l_1{:}bill(C),\ l_2{:}a(L, l_3, h_0),\ l_3{:}leave(L, C),\ l_3{:}pol(PL)$

ANNOTATIONS IMPLICATIFS :

{`top:bottom`}

Implication : NON

Nairn *et al.* [2006] décomposent l'ensemble des prédicats implicatifs en neuf classes selon qu'ils gardent, inversent, ou neutralisent la sémantique du complément dans un contexte positif ou négatif. Le contexte neutre neutralise tout le temps la sémantique.

	type de la classe	prédicat dans la classe	Polarité Relative			
			(+)	positif	(-)	négatif
			Implication			
Implicatif dans	`both-equal`	« manage to »	(+)	positif	(-)	négatif
les deux sens	`both-inv`	« forget to »	(-)	négatif	(+)	positif
Implicatif dans	`pos-equal`	« force to »	(+)	positif	($\sim$)	indéfini
le sens positif	`pos-inv`	« refuse to »	(-)	négatif	($\sim$)	indéfini
Implicatif dans	`neg-equal`	« attempt to »	($\sim$)	indéfini	(-)	négatif
le sens négatif	`neg-inv`	« hesitate to »	($\sim$)	indéfini	(+)	positif
			Présupposition			
Factif	`both-pos`	« forget that »	(+)	positif	(+)	positif
Contre-factifs	`both-neg`	« pretend that »	(-)	négatif	(-)	négatif
			Implication/Présupposition			
Neutre	`both-indef`	« want to »	($\sim$)	indéfini	($\sim$)	indéfini

L'algorithme de Nairn *et al.* [2006] permet de dire pour chaque proposition, si elle est vraie, fausse ou neutre dans chacun des prédicats ayant porté sur lui. Dans l'exemple (33), nous saurons que pour la phrase T « not » est faux à la racine, « refuse » est vrai au niveau de « not » et indéfini à la racine et enfin que « leave » est vrai au niveau de « refuse » et de « not » et indéfini à la racine. C'est pour cela que la phrase T ne peut pas impliquer la phrase H. Cette méthode fonctionne dans la plupart des cas, mais il existe cependant une combinaison de prédicats posant problème. En effet, si nous combinons un prédicat neutre avec un factif, le prédicat complément ne sera pas impliqué. Ainsi, l'exemple (35) qui devrait être considéré comme vrai ne le sera pas car la définition des factifs ne permet pas de rendre vrai au niveau de la racine un factif dans un contexte neutre.

Pour intégrer cette approche dans notre système nous avons décidé de garder uniquement l'information permettant de savoir si un prédicat est vrai, faux ou indéfini au niveau de la racine (i.e., dans le monde « réel »). Nous avons intégré un ensemble de prédicats à la sémantique plate permettant d'associer une polarité à un prédicat et de calculer une polarité en fonction d'une autre polarité. Si un prédicat à une polarité *true* il est donc vrai à la racine, si la polarité est $false$ il est faux à la racine et enfin si la polarité est $indef$ cela signifie qu'on ne peut pas savoir si le prédicat est vrai ou faux à la racine. Un prédicat $< polpred > (X, Y)$ peut être vu comme une affectation $X =< polpred > (Y)$ avec les fonctions $< polpred > (Y)$ définies comme suit :

$< polpred >$	Y		
	$true$	$false$	$indef$
$equals$	$true$	$false$	$indef$
inv	$false$	$true$	$indef$
$t2t$	$true$	$indef$	$indef$

Le prédicat *equals* renvoie la même même valeur qu'il reçoit (c'est l'identité), le prédicat *inv* inverse les valeurs *true* et $false$ et garde $indef$ à l'identique, et enfin le prédicat $t2t$ garde la valeur *true* à l'identique et renvoie $indef$ pour les autres valeurs. À partir de ces trois prédicats, nous pouvons représenter les 9 classes possibles. Étant donné P la polarité d'un prédicat et C la polarité de son complément, nous avons pour les neufs classes représentées comme suit :

`both-equal`	$equals(C, P)$
`both-inv`	$inv(C, P)$
`pos-equal`	$t2t(C, P)$
`pos-inv`	$t2t(C, T)\ inv(T, S)$
`neg-equal`	$inv(C, U)\ t2t(U, T)\ inv(T, S)$
`neg-inv`	$inv(U, T)\ t2t(T, S)$
`both-pos`	$equals(C, true)$
`both-neg`	$equals(C, false)$
`both-indef`	$equals(C, indef)$

Avant de transformer la sémantique plate en logique du premier ordre, il est nécessaire d'effectuer le calcul des polarités pour savoir si un prédicat est vrai, faux ou indéfini. Ce calcul doit être fait dans un certain ordre, car pour un prédicat $polpred(X, Y)$ il est nécessaire de connaitre la valeur de Y pour avoir celle de X. Dans l'exemple (32), nous obtenons donc la suite d'affectations suivante : $PR = true$, $PT = true$, $PL = false$. Cela signifie donc que la conjonction des prédicats associés à l'étiquette l_6 (e.g. : le prédicat $leave(L, C)$ uniquement) est fausse à la racine car la polarité PL de cette étiquette est $false$.

Pour la transformation des formules en logique du premier ordre, il faut tout d'abord extraire les prédicats de polarité de la formule et calculer les polarités. Puis nous appliquons une version modifiée de la fonction de transformation τ où nous remplaçons la règle 1 :

$$\tau(l{:}\{p_0, \ldots, p_n\}) = \tau(p_0) \wedge \ldots \wedge \tau(p_n)$$

par la règle

$$\tau(l{:}\{p_0, \ldots, p_n\}) = \begin{cases} \tau(p_0) \wedge \ldots \wedge \tau(p_n) & \text{si la polarité de } l \text{ est } true \\ \neg(\tau(p_0) \wedge \ldots \wedge \tau(p_n)) & \text{si la polarité de } l \text{ est } false \\ \exists neutral.(neutral \vee (\tau(p_0) \wedge \ldots \wedge \tau(p_n))) & \text{si la polarité de } l \text{ est } indef \end{cases}$$

Pour l'exemple (32), on aura donc la formule de logique du premier ordre suivante :

$$bill(C) \wedge \exists L.(\neg(leave(L, C)) \wedge \exists M.refuse(R, C, L))$$

Nous associons la polarité séparément à chaque ensemble de prédicats avec la même étiquette plutôt qu'à l'ensemble de la portée car cela empêcherait de détecter l'implication (31) à cause de la disjonction que l'on obtiendrait en appliquant la négation sur $\exists L.leave(L, C) \wedge \neg\exists M.refuse(R, C, L)$. De plus, cette méthode permet aussi de détecter l'implication (34) où il faudrait extraire le complément de toutes les portées si l'on avait appliqué les polarités sur l'intégralité de chaque portée. Cette méthode n'est cependant pas exempte de problème, car si la vérification d'implication fonctionne comme nous le souhaitons, il n'en est pas de même pour la contradiction parce que les polarités ne modifient pas les quantificateurs. Pour que les deux fonctionnent il faudrait que les négations ne transforme pas les conjonctions en disjonctions mais gardent la conjonction, on aurait alors $\neg(a \wedge b)$ qui donne $\neg a \wedge \neg b$.

La formule associée à la polarité indéfinie permet de neutraliser le prédicat de façon à ce que la formule n'implique ni le prédicat ni sa négation. Pour l'exemple de « Bill didn't manage to leave » (c.f., (31), nous obtenons donc :

$$\exists b.(bill(b) \wedge \exists l.(\neg leave(l, c) \wedge \exists r.\neg manage(r, c, l)))$$

De la même manière que pour les quantificateurs, nous ajoutons des annotations sur les enchâssements de verbes à complément phrastique en utilisant le type [14]. L'exemple (34) aura par exemple les annotations (`root`,`both-inv`), (`both-inv`, `both-pos`) et (`both-pos`,`normal`).

3.4 Génération d'une suite de tests équilibrée

Pour qu'une suite de tests permette d'évaluer au mieux et le plus équitablement les systèmes, il est nécessaire que sa distribution soit équilibrée selon divers critères. Par exemple, une suite de tests d'implications textuelles ayant une distribution avec plus d'implications que de non-implications favorisera les systèmes répondant plus souvent Oui que Non. Il est aussi important d'avoir une bonne représentation de tous les phénomènes étudiés ainsi que de leur combinaisons. Dans la section 1.3 nous avons vu l'exemple du RTE2 et de l'analyse des résultats de [Garoufi, 2007] qui montre bien que le corpus de RTE2 n'est pas bien distribué et nous verrons dans la section 5.2 qu'une analyse simple de l'évaluation donne des résultats erronés. Nous allons maintenant présenter la solution que nous avons développé pour extraire à

14. le type étant une des neufs classes vues précédemment plus le type `normal` pour les verbes sans complément et `root` pour la racine

partir d'un ensemble d'éléments un sous-ensemble d'une taille donnée satisfaisant au mieux des contraintes de distribution.

Pour illustrer notre approche nous allons utiliser comme exemple une suite de tests d'implications textuelles avec deux contraintes de distribution. La première est la contrainte d'implication CI sur l'ensemble d'annotations d'implication $AI = \{\text{Oui}, \text{Non}\}$, qui impose d'avoir autant de problèmes d'implication et de non-implication. La seconde est la contrainte CV sur l'ensemble d'annotations sur les types de verbes $AV = \{tran, intran, ditran\}$ qui impose qu'il y ait 50% de transitifs ($tran$), 30% d'intransitifs ($intran$) et 20% de ditransitifs ($ditran$). Nous souhaitons que ces contraintes soient respectées, globalement (i.e., sur le corpus entier), mais aussi localement (e.g., qu'il y ait autant d'implications et de non-implications dans les sous-corpus contenant uniquement les transitif, ditransitif et intransitif) dans la mesure du possible. Nous commençons par définir formellement les différents concepts nécessaires à la description de notre approche.

Définition 3.4.0.1 (Ensemble partitionné avec un ensemble d'annotations)
Soit $\boldsymbol{E}$ un ensemble d'éléments, et $\boldsymbol{A = \{a_1, \ldots, a_m\}}$ un ensemble d'annotations qui associe à chaque annotation $\boldsymbol{a_i}$, un sous-ensemble $\boldsymbol{p_{(a_i,E)}}$ de $\boldsymbol{E}$ représentant l'ensemble des éléments de $\boldsymbol{E}$ annotés avec $\boldsymbol{a_i}$. L'ensemble $\boldsymbol{E}$ est partitionné avec l'ensemble d'annotations $\boldsymbol{A}$, si l'ensemble des sous-ensembles $\boldsymbol{p_{(a_i,E)}}$ associés à chaque $\boldsymbol{a_i}$ de $\boldsymbol{A}$ forme une partition de $\boldsymbol{E}$.

Dans notre exemple l'ensemble d'annotations AI partitionnera notre ensemble de couples car l'implication dans un couple peut être soit vraie, soit fausse, mais pas les deux à la fois. Pour l'ensemble d'annotations AV nous considérons que chaque couple ne prend en compte qu'un seul type de prédicat. Si nous souhaitons pouvoir avoir plusieurs types de prédicat associés à un couple il suffit de diviser l'ensemble d'annotations $\{tran, intran, ditran\}$ en trois ensembles d'annotations $\{tran, no\text{-}tran\}$, $\{intran, no\text{-}intran\}$ et $\{ditran, no\text{-}ditran\}$. Pour notre exemple nous nous contenterons de la première version plus simple à expliquer.

Définition 3.4.0.2 (Ensemble globalement contraint)
Soit $\boldsymbol{E}$ un ensemble partitionné avec un ensemble d'annotations $\boldsymbol{A = \{a_1, \ldots, a_m\}}$. Une contrainte de distribution $\boldsymbol{C}$ associe à chaque annotation $\boldsymbol{a_i}$, un ratio $\boldsymbol{c_{a_i}}$. L'ensemble est globalement contraint par cette contrainte si pour chaque annotation $\boldsymbol{a_i}$ le ratio $\boldsymbol{|p_{(a_i,E)}|/|E|}$ est égal au ratio $\boldsymbol{c_{a_i}}$.

Définition 3.4.0.3 (Fonction d'assignation)
Soit $\boldsymbol{\mathcal{A}}$ un ensemble d'ensembles d'annotations $\boldsymbol{A_i = \{a_{(i,0)}, \ldots, a_{(i,n)}\}(n = |A_i|)}$. Une fonction d'assignation $\boldsymbol{s_\mathcal{A}}$ sur $\boldsymbol{\mathcal{A}}$ est une fonction qui pour chaque ensemble d'annotation $\boldsymbol{A_i}$, retourne une de ces annotations $\boldsymbol{a_{(i,j)}}$. $\boldsymbol{\mathcal{S}_\mathcal{A}}$ est l'ensemble des fonctions d'assignations possibles sur $\boldsymbol{\mathcal{A}}$.

Définition 3.4.0.4 (Ensemble localement contraint)

Soit $\boldsymbol{E}$ *un ensemble partitionné avec chacun des ensembles d'annotations* $\boldsymbol{A_i} = \{\boldsymbol{a_{(i,0)}}, \ldots, \boldsymbol{a_{(i,n)}}\}$ *de l'ensemble* $\boldsymbol{\mathcal{A}}$*. Soit des contraintes* $\boldsymbol{C_i}$ *associées aux ensembles d'annotations* $\boldsymbol{A_i}$*. L'ensemble* $\boldsymbol{E}$ *est localement contraint par un ensemble de contraintes* $\boldsymbol{\mathcal{C}} = \{\boldsymbol{C_0}, \ldots, \boldsymbol{C_m}\}$ *, si chaque contrainte* $\boldsymbol{C_i}$ *contraint globalement chacun des ensembles :*

$$\left\{ \bigcap_{\boldsymbol{A_j} \in \boldsymbol{\mathcal{A}} | \boldsymbol{A_j} \neq \boldsymbol{A_i}} \boldsymbol{p_{(s_{\mathcal{A}}(A_j),E)}} \mid \boldsymbol{s_{\mathcal{A}}} \in \boldsymbol{\mathcal{S}_{\mathcal{A}}} \right\}$$

Théorème 3.4.0.1

Soit $\boldsymbol{E}$ *un ensemble partitionné avec les ensembles d'annotations* $\boldsymbol{A_i} = \{\boldsymbol{a_{(i,0)}}, \ldots, \boldsymbol{a_{(i,n)}}\}$*. Soit des contraintes* $\boldsymbol{C_i}$ *associées aux ensembles d'annotations* $\boldsymbol{A_i}$*. Si l'ensemble* $\boldsymbol{E}$ *est localement contraint par les contraintes* $\boldsymbol{C_i}$*, alors chaque contrainte* $\boldsymbol{C_i}$ *contraint globalement* $\boldsymbol{E}$*.*

Notre objectif est donc d'obtenir à partir d'un ensemble d'éléments $\boldsymbol{E_b}$, un sous-ensemble $\boldsymbol{E_s}$ de taille $\boldsymbol{n}$ ($\boldsymbol{n} \leq |\boldsymbol{E_c}|$) localement contraint par un ensemble de contraintes $\boldsymbol{\mathcal{C}}$. Le problème est qu'il est tout à fait possible qu'aucun sous-ensemble de $\boldsymbol{E_b}$ de taille $\boldsymbol{n}$ satisfasse l'ensemble de contraintes $\boldsymbol{\mathcal{C}}$. Dans notre exemple, nous allons prendre l'ensemble de base $\boldsymbol{E_b}$, partitionné de la façon suivante (les nombres représentent les cardinalités des sous-ensembles, le choix des éléments de cet ensemble pouvant être fait plus tard) :

$\boldsymbol{E_b}$		*AV*		
		tran	*intranb*	*bitran*
AI	Oui	9	8	9
	Non	15	6	2

Disons que nous souhaitions prendre 40 éléments dans $\boldsymbol{E_b}$ et respecter les contraintes de distribution $\boldsymbol{CI}$ et $\boldsymbol{CV}$. Dans le meilleur des cas nous aurions l'ensemble parfait $\boldsymbol{E_p}$ représenté par le tableau ci-dessous à gauche, mais étant donné que $\boldsymbol{E_b}$ n'a pas assez d'éléments dans les intersections Oui $\cap$ *tran* et Oui $\cap$ *bitran*, la meilleure solution $\boldsymbol{E_{ms}}$ possible est celle de droite.

$\boldsymbol{E_p}$		AV		
		tran	*intranb*	*bitran*
AI	Oui	10	6	4
	Non	10	6	4

$\boldsymbol{E_{ms}}$		AV		
		tran	*intranb*	*bitran*
AI	Oui	9	5	6
	Non	11	7	2

Pour obtenir cette solution nous utilisons la propriété suivante :

Propriété 3.4.0.1 ()

Soit $\boldsymbol{E}$ *un ensemble partitionné avec chacun des ensembles d'annotations* $\boldsymbol{A_i} = \{\boldsymbol{a_{(i,0)}}, \ldots, \boldsymbol{a_{(i,n)}}\} (\boldsymbol{n} = |\boldsymbol{A_i}|)$ *de l'ensemble* $\boldsymbol{\mathcal{A}}$*. Soit l'ensemble de contraintes* $\boldsymbol{\mathcal{C}}$ *composé des contraintes de distribution* $\boldsymbol{C_i}$ *associées aux ensembles d'annotations* $\boldsymbol{A_i}$ *qui contraignent globalement* $\boldsymbol{E}$*.*

Pour toute couple d'ensembles d'annotations $(\boldsymbol{A_i}, \boldsymbol{A_j})$*, pour toute couple de fonctions d'assignations* $(\boldsymbol{s'_{\mathcal{A}}}, \boldsymbol{s''_{\mathcal{A}}})$*, si nous ôtons un élément aux ensembles :*

$$\bigcap_{\boldsymbol{A_k} \in \boldsymbol{\mathcal{A}}} \boldsymbol{p_{(s'_{\mathcal{A}}(A_k),E)}} \quad \boldsymbol{et} \quad \left(\bigcap_{\boldsymbol{A_k} \in \boldsymbol{\mathcal{A}} | \boldsymbol{k} \notin \{\boldsymbol{i},\boldsymbol{j}\}} \boldsymbol{p_{(s'_{\mathcal{A}}(A_k),E)}} \right) \cap \boldsymbol{p_{(s''_{\mathcal{A}}(A_i),E)}} \cap \boldsymbol{p_{(s''_{\mathcal{A}}(A_j),E)}}$$

et ajoutons un élément aux ensembles

$$\left(\bigcap_{\boldsymbol{A_k} \in \boldsymbol{\mathcal{A}} | \boldsymbol{k} \neq \boldsymbol{i}} \boldsymbol{p_{(s'_{\mathcal{A}}(A_k),E)}} \right) \cap \boldsymbol{p_{(s''_{\mathcal{A}}(A_i),E)}} \quad \boldsymbol{et} \quad \left(\bigcap_{\boldsymbol{A_k} \in \boldsymbol{\mathcal{A}} | \boldsymbol{k} \neq \boldsymbol{j}} \boldsymbol{p_{(s'_{\mathcal{A}}(A_k),E)}} \right) \cap \boldsymbol{p_{(s''_{\mathcal{A}}(A_j),E)}}$$

alors l'ensemble $\boldsymbol{E}$ *sera toujours globalement contraint par les contraintes* $\boldsymbol{C_i}$*. Nous appelons cette opération un pivot de* $\boldsymbol{s'_{\mathcal{A}}}$ *vers* $\boldsymbol{s''_{\mathcal{A}}}$ *par* $(\boldsymbol{A_i}, \boldsymbol{A_j})$*.*

Cela est facilement prouvable en regardant les ajouts et suppressions que nous effectuons aux niveaux de chaque sous-ensemble $\boldsymbol{p_{(a_{(m,n)},E)}}$*, car il y a autant d'ajout que de suppression pour chacun d'eux.*

L'algorithme est composé de 3 étapes : nous calculons dans un premier temps la solution parfaite $\boldsymbol{E_p}$ en rajoutant des éléments factices si nécessaire, puis nous appliquons des pivots pour obtenir un cas possible pour notre ensemble $\boldsymbol{E_b}$, et enfin nous lissons ensuite la solution pour obtenir le meilleur cas possible. Nous définissons une fonction $\boldsymbol{f}$ permettant de donner un score à une solution $\boldsymbol{E_f}$ (cette fonction fait la somme de toutes les différences locales entre la solution courante et la solution parfaite multiplié par l'inverse du pourcentage voulu pour ce local) :

$$\boldsymbol{f(E_s)} = \sum_{\forall \boldsymbol{s_{\mathcal{A}}} \in \boldsymbol{\mathcal{S}_{\mathcal{A}}}} \left((\boldsymbol{e}^{\boldsymbol{abs}(|\bigcap_{\boldsymbol{A_i} \in \boldsymbol{\mathcal{A}}} \boldsymbol{p_{(s_{\mathcal{A}}(A_i),E_p)}}| - |\bigcap_{\boldsymbol{A_i} \in \boldsymbol{\mathcal{A}}} \boldsymbol{p_{(s_{\mathcal{A}}(A_i),E_f)}}|)} - \boldsymbol{1}) \div \left(\prod_{\boldsymbol{A_i} \in \boldsymbol{\mathcal{A}}} \boldsymbol{c_{s_{\mathcal{A}}(A_i)}} \right) \right)$$

Le score de la répartition parfaite est de 0. L'exponentiel permet de favoriser une répartition ayant plusieurs petites différences plutôt qu'une grosse différence par rapport à la répartition parfaite. La division permet de favoriser les locaux ayant de gros ratios pour qu'ils acceptent plus d'éléments. Cette formule est un élément très important de l'algorithme car c'est elle qui permet de définir le choix de la meilleur forme possible.

La première étape est triviale, il suffit de multiplier les ratios, et de rajouter des éléments factices pour obtenir l'ensemble $\boldsymbol{E_p}$. Pour la seconde étape, nous commençons avec l'ensemble $\boldsymbol{E_f}$ qui est le même que $\boldsymbol{E_p}$ puis nous cherchons la fonction

d'assignation $s_{\mathcal{A}}$ pour laquelle il existe le plus d'éléments factices dans E_f :

$$max(\{(|\bigcap_{A_i \in \mathcal{A}} p_{(s_{\mathcal{A}}(A_i),E_f)}| - |\bigcap_{A_i \in \mathcal{A}} p_{(s_{\mathcal{A}}(A_i),E_b)}|) \mid s_{\mathcal{A}} \in \mathcal{S}_{\mathcal{A}}\})$$

Nous cherchons ensuite la fonction d'assignation $s'_{\mathcal{A}}$ ainsi que les deux ensembles d'annotations A_i et A_j permettant d'avoir le pivot de $s_{\mathcal{A}}$ vers $s'_{\mathcal{A}}$ par (A_i, A_j) qui appliqué à E_f donne l'ensemble E'_f ayant le plus petit score tout en diminuant le nombre d'éléments factices. Nous appliquons, cette méthode jusqu'à obtenir un ensemble E_f ne contenant que des éléments de E_b (i.e. : qui ne contient aucun éléments factice). Pour finir, nous regardons si il existe un pivot permettant d'obtenir à partir de l'ensemble E_f un nouvel ensemble E'_f avec un meilleur score pour la fonction f tout en ne créant pas d'élément factice.

Dans notre exemple, les fonctions d'assignations qui posent problèmes sont :

$s_1(\text{AI}) = \text{Oui}$ $\quad$ $s_2(\text{AI}) = \text{Non}$
$s_1(\text{AV}) = tran$ $\quad$ $s_2(\text{AV}) = bitran$

Pour s_1 il y a un élément factice, et pour s_2 il y en a deux. Nous commençons par faire un pivot sur s_2 car il contient plus d'éléments factices. Nous avons que deux ensemble d'annotations donc le choix du pivot se fait uniquement au niveau de la fonction d'assignation. Avec s_2 nous pouvons uniquement appliquer le pivot vers s_{21} ou s_{22}, car les autres fonctions d'assignation ne diminuent pas le nombre d'éléments factices. L'ensemble obtenu par le pivot de s_2 vers s_{21} par (AI, AV) obtient un score de 48,1 et celui par le pivot de s_2 vers s_{22} par (AI, AV) un score de 57,3. Nous choisissons donc l'ensemble obtenu par le pivot vers s_{21} représenté par le tableau E_{f_1} ci-dessous. Il ne reste plus qu'un seul élément factice en s_2. L'ensemble obtenu par le pivot de s_2 vers s_{21} par (AI, AV) obtient un score de 178,9 et celui par le pivot de s_2 vers s_{22} par (AI, AV) un score de 164,4. Nous choisissons donc l'ensemble obtenu par le pivot vers s_{21} représenté par le tableau E_{f_2} ci-dessous. Il n'y a plus d'élément factice et la fonction de lissage ne pourra pas améliorer le résultat, le résultat E_{f_2} est donc le meilleur résultat.

$s_1(\text{AI}) = \text{Oui}$ $\quad$ $s_2(\text{AI}) = \text{Oui}$
$s_1(\text{AV}) = tran$ $\quad$ $s_2(\text{AV}) = intran$

E_{f_1}		AV		
		tran	*intran*	*bitran*
AI	Oui	9	6	5
	Non	11	6	3

E_{f_2}		AV		
		tran	*intran*	*bitran*
AI	Oui	9	5	6
	Non	11	7	2

Dans les tests que nous avons effectué pour évaluer l'algorithme nous avons toujours obtenu la meilleure réponse. Il est possible qu'il existe une exemple où il soit

nécessaire de faire un pivot qui n'est pas le meilleur pour obtenir le meilleur résultat final. Il suffirait dans ce cas de combiner notre approche avec un algorithme génétique pour obtenir la meilleure solution. Une autre amélioration pourrait être de trouver un moyen de trouver le sous-ensemble qui arrive à avoir le meilleur équilibre entre la taille et la satisfaction des contraintes.

Un autre exemple plus complexe de ce que notre algorithme peut faire, est de vouloir extraire 100 éléments de l'ensemble avec la répartition ci-dessous à gauche, ce qui dans le meilleur des cas doit donner le sous-ensemble du tableau ci-dessous à droite.

A	B	C
$a_1 : 50\%$	$b_1 : 40\%$	$c_1 : 60\%$
$a_2 : 50\%$	$b_2 : 50\%$	$c_2 : 20\%$
	$b_3 : 10\%$	$c_3 : 20\%$

		A					
		a_1			a_2		
B	b_1	12	4	4	12	4	4
	b_2	15	5	5	15	5	5
	b_2	3	1	1	3	1	1
		c_1	c_2	c_3	c_1	c_2	c_3
		C					

Nous obtenons en prenant comme corpus de base le corpus ci-dessous à gauche, la sélection ci-dessous à droite.

		A					
		a_1			a_2		
B	b_1	10	100	100	100	100	100
	b_2	12	100	100	100	100	2
	b_2	100	100	100	1	100	100
		c_1	c_2	c_3	c_1	c_2	c_3
		C					

		A					
		a_1			a_2		
B	b_1	10	4	4	14	3	5
	b_2	12	6	7	18	5	2
	b_2	5	1	1	1	1	1
		c_1	c_2	c_3	c_1	c_2	c_3
		C					

3.5 Conclusion

Ce chapitre nous a donc permis de décrire notre méthodologie pour obtenir une suite de tests annotée et équilibrée. Dans un premier temps nous avons décrit le système GenI qui permet de générer à partir d'une formule en sémantique plate un ensemble de phrases véhiculant cette sémantique. Nous avons ensuite proposé une manière de gérer la sémantique lexicale à moindre coût en intégrant certains phénomènes dans le lexique de lemmes et d'autres dans le lexique morphologique. Nous avons proposé des améliorations de la sémantique afin de pouvoir gérer les verbes implicatifs. Nous avons vu comment tester l'implication entre deux formules de sémantique plate en les transformant en logique du premier ordre, permettant ainsi de générer des couples de textes s'impliquant ou non. Nous avons proposé en parallèle, un moyen d'annoter les problèmes obtenus avec les variations syntaxiques

existantes entre les deux textes de chaque problème. Pour finir, nous avons décrit notre algorithme d'extraction de suite de tests équilibrée d'une certaine taille à partir d'un ensemble de problèmes. Nous avons ainsi pu obtenir une suite de tests équilibrée et finement annotée.

Chapitre 4

Normalisation par la réécriture

Dans ce chapitre nous commençons par décrire les structures qu'Afazio utilise ainsi que la nomenclature graphique que nous avons défini pour les représenter (section 4.1). Ensuite, nous présentons la méthodologie utilisée pour obtenir les règles de réécriture permettant d'étiqueter sémantiquement les propositions verbales sur les structures syntaxiques (section 4.2). Nous avons dans un premier temps développé des règles manuellement puis nous avons mis en place une méthodologie permettant de dériver semi-automatiquement de nouvelles règles à partir de celles que nous avions. Par la suite nous décrivons comment nous avons dérivé de nouvelles règles de réécriture pour étiqueter sémantiquement les propositions nominales à partir de la ressource linguistique NomLexPlus (section 4.3). Enfin, nous détaillons les différents traitements que nous appliquons aux structures syntaxiques étiquetées, pour dériver les arbres syntaxiques des formules de logique du premier ordre leur correspondant (section 4.4).

4.1 Les structures utilisées

Nous allons décrire dans cette section les structures qu'Afazio utilise ainsi que la nomenclature graphique que nous avons défini pour les représenter. Les graphes que nous utilisons sont composés de nœuds et d'arcs dirigés ayant chacun une classe spécifique (à la manière des classes dans un langage orienté objet) appartenant à une hiérarchie de classes que nous avons défini. Pour la définition des règles de filtrage, nous rappelons qu'un élément de classe `A` filtre tout les éléments de classe `A` ou d'une de ses sous-classes. À une classe de nœuds/d'arcs est associée une liste d'attributs qui est associée aux instances de cette classe. Les classes héritent des d'attributs de leurs ancêtres. Ci-dessous nous avons les descriptions des classes de bases que nous utilisons pour représenter nos graphes. L'attribut *used* permet d'empêcher plusieurs règles de s'appliquer sur les mêmes structures. Ainsi la règle pour le passif long marquera le nœud représentant le verbe comme étant utilisé empêchant ainsi la règle

associée au passif court de s'appliquer.

- Classe de nœuds génériques :
  - **gnode**
    - *gid* : identifiant du graphe auquel le nœud appartient
    - *used* : booléen permettant de savoir si le nœud a été utilisé
- Classe d'arcs génériques :
  - **gedge**
    - *gid* : identifiant du graphe auquel l'arc appartient

Nous avons divisé le reste des classes en 4 groupes représentant chacun un niveau d'information (constituants, dépendances, étiquetage sémantique, représentation sémantique). Pour que les niveaux soient plus facile à distinguer dans les représentations graphiques de nos graphes, nous avons associé une couleur à chacun d'eux. Nous avons donc un niveau pour représenter les constituants (●), un niveau pour représenter les dépendances (●), un niveau pour représenter l'étiquetage sémantique (●) et enfin un niveau de représenter la sémantique (●).

4.1.1 Le niveau de constituants

Ce niveau est composé des classes de nœuds et d'arcs ci-dessous. L'indentation représente la hiérarchie entre les différentes classes, la puce colorée indique à quelle couche appartient la classe, et les éléments avec des « – » représentent les attributs d'une classe.

- Classes de nœuds :
  - **gnode**
    - ● **ncst** : un constituant
      - cst : type de constituant (e.g., NNP, VP, . . .)
    - ●● **nword** : un mot
      - wid : identifiant du mot représentant sa position dans la phrase
      - word : le mot tel qu'il apparait dans le texte
      - lemma : le lemme associé au mot
      - verb : le verbe pouvant être associé au mot en cas de nominalisation d'un verbe
- Classe d'arcs :
  - **gedge**
    - ● **ecst** : relie les constituants et les mots entre eux

La classe de nœuds `nword` est représentée avec les couleurs des niveaux de constituants (●) et de dépendances (●) car elle appartient aux deux niveaux. Les instances de la classe `nword` sont représentées avec la couleur du niveau de dépendances (●) car les règles de réécriture sont majoritairement basées sur des motifs de graphe de dépendances. Il y a une contrainte qui oblige les instances de `ecst` à avoir comme source et comme destination une instance de `ncst` (et donc aussi de `nword` qui en

hérite). De plus, un nœud `ncst` ne peut pas avoir plus d'un arc `ecst` entrant.

4.1.2 Le niveau de dépendances

Il existe une hiérarchie dans les dépendances de l'analyseur syntaxique de Stanford, mais nous avons décidé de mettre toutes les classes de dépendances au même niveau afin d'avoir des règles strictes sur ces classes de dépendances. Par exemple, dans les dépendances de Stanford `nsubj` hérite de `dep`, mais nous ne voulons pas que `nsubj` soit filtré avec `dep` comme l'aurait permis la modélisation respectant la hiérarchie de dépendances de Stanford. Il y a en tout 54 dépendances possibles. Le niveau de dépendances est composé des classes suivantes :

- Classe de nœuds :
  - **nword**
- Classes d'arcs :
  - **gedge**
    - **edep** : classe générique pour les dépendances
      - **dep** : connexion générique entre deux mots
      - **nsubj** : connexion entre un verbe et son sujet
      - **dobj** : connexion entre un verbe et son objet
      - ...

Toutes les dépendances sont contraintes d'avoir comme source et destination des nœuds de classe `nword`.

4.1.3 Le niveau d'étiquetage sémantique

Ce niveau sert à étiqueter les structures existantes de manière à savoir quels sont les arguments sémantiques d'une proposition. Pour cela, on ajoute un nœud de classe `nprop` représentant le prédicat de la proposition, et on le relie au prédicat et aux arguments via des arcs de classes diverses. Ce niveau est composé des classes suivantes :

- Classes de nœuds :
  - **nword**
    - **nprop** : proposition sémantique pour l'étiquetage
      - rewrittenby : pour savoir quelle règle a créée l'annotation
- Classes d'arcs :
  - **gedge**
    - **eprop** : classe générique pour les arcs d'étiquetage
      - **epred** : liaison de la proposition au mot représentant le prédicat
      - **earg** : classe générique pour les arguments
        - **arg0** : liaison de la proposition au mot représentant le premier argument
        - **arg1** : liaison de la proposition au mot représentant le second argument

- ...

Tous les arcs ont comme source un `nprop` et comme destination un `nword`. Un `nprop` ne peut avoir qu'un arc sortant de chaque type et un `nword` peut avoir plusieurs `arg` entrant mais un seul `epred` entrant.

4.1.4 Le niveau sémantique

Ce niveau sert à représenter une structure syntaxique de formule logique, mais il contient aussi des arcs servant à la construction de cette structure que nous représentons en pointillé. Le niveau sémantique est composé des classes suivantes :

- Classe de nœuds :
  - **gnode**
    - **nsem** : classe générique pour les nœuds sémantiques
      - **semroot** : racine de la formule sémantique
      - **sempred** : prédicat logique
        - name : nom du prédicat logique
      - **semvar** : variable associée à un prédicat logique
        - varid : identifiant de la variable
      - **semcon** : classe générique pour les connecteurs logiques
        - **semconj** : conjonction
        - **semdisj** : disjonction
        - **semneg** : négation
        - **semimpl** : implication
      - **semquant** : classe générique pour les quantificateurs
        - varid : identifiant de la variable associée au quantificateur
        - **semexists** : quantificateur existentiel
        - **semforall** : quantificateur universel
        - **semconst** : constante
- Classes d'arcs :
  - **gedge**
    - **esem** : classe générique pour les arcs sémantiques
    - **con** : classe générique pour lier les connecteurs logiques à leurs arguments
      - **con0** : lie un connecteur logique à son premier argument ou à tous ces arguments s'il est commutatifs
      - **con1** : lie une implication à sa partie droite
    - **var** : classe générique pour lier les prédicats logiques à leurs arguments
      - **var0** : lie le prédicat à son premier argument
      - **var1** : lie le prédicat à son second argument
- Classes d'arcs de construction :
  - **esem**

- **semhead** : lie un mot à la tête sémantique qui lui est associée
- **semrest** : lie un mot à la restriction sémantique qui lui est associée
- **semscope** : lie un mot à la portée sémantique qui lui est associée
- **semquant** : lie un mot au quantificateur qui lui est associé

Les restrictions appliquées à ce niveau sont les suivantes. Les arcs **con0** peuvent connecter des nœuds **semcon** ou **semquant** à des nœuds **semcon**, **semquant** ou **sempred**. Les arcs **con1** sont comme les arcs **con0**, mais ils peuvent uniquement sortir d'un nœud **semimpl**. Un nœud ne peut avoir qu'un connecteur **con0** ou **con1** entrant. Un nœud **semimpl** doit avoir exactement un connecteur **con0** et **con1** sortant. Les nœuds **semneg** ou **semquant** doivent avoir exactement un connecteur **con0** sortant. Les nœuds **semconj** et **semdisj** peuvent avoir un ou plus connecteurs **con0** sortant. Les nœuds **sempred** doivent avoir un arc **var0** sortant et peuvent avoir un arc **var1** sortant.

4.2 Normalisation de la réalisation des arguments verbaux

Nous allons maintenant présenter notre méthodologie pour obtenir la couche d'étiquetage sémantique à partir des couches de constituants et de dépendances. Nous avons utilisé la grammaire de l'anglais XTag [Group, 2001] pour avoir une énumération précise des variations syntaxiques possibles afin de pouvoir étiqueter de la même façon toutes les phrases (37). Nous avons dans un premier temps généré une base de règles de réécriture permettant d'annoter les structures prédicatives pour les types de verbes les plus simples (e.g., intransitif, transitif), puis nous avons mis en place une méthodologie permettant de semi-automatiser la création d'autres règles à partir de cette base. Pour finir nous avons utilisé PropBank afin d'améliorer le lexique associé.

(37) a. « The flower which smells nice is given to the woman by the man. »

b. « The flower which smells nice is given the woman by the man. »

c. « The flower which is given the woman by the man smells nice. »

d. « The flower which is given to the woman by the man smells nice. »

e. « The flower that smells nice is given to the woman by the man. »

f. « The flower that smells nice is given the woman by the man. »

g. « The flower that is given the woman by the man smells nice. »

h. « The flower that is given to the woman by the man smells nice. »

4.2.1 Préparation des données

La grammaire XTag est composée de familles d'arbres dont celles pour les verbes transitifs `Tnx0Vnx1`, intransitif `Tnx0V`, ditransitif `Tnx0Vnx2nx1`, ditransitif avec préposition `Tnx0Vnx1Pnx2`. Chaque famille est composée d'un ensemble d'arbres décrivant les variations syntaxiques possibles pour la classe de verbes associée. Par exemple, pour la famille des verbes transitifs `Tnx0Vnx1` nous avons les variations :

- `nx0Vnx1` (i.e. : « John loves Mary »)
- `nx1Vbynx0` (i.e. : « Mary is loved by John »)
- `nx1V` (i.e. : « Mary is loved »)
- `Np0nx0Vnx1` (i.e. : « John who loves Mary »)
- `Np1nx0Vnx1` (i.e. : « Mary whom John loves »)
- `Np1nx1Vbynx0` (i.e. : « Mary who is loved by John »)
- `DnxOVnx1` (i.e. : « the loving of John by Mary »)
- ...

Afin de faciliter la compréhension de cette section nous allons décrire la convention de dénomination des familles et des arbres XTag. Tout d'abord les familles se différentient des arbres par le fait que leur nom commence par la lettre "T". Ensuite le nom peut contenir un des éléments ci-dessous, les numéros correspondent à l'indice sémantique de l'argument ou de l'élément déplacé. Dans `Nc` et `Npx` l'absence d'indice signifie la relativisation d'un adjoint.

`nx0`, `nx1`, `nx2`	:	groupe nominal
`s0`, `s1`, `s2`	:	syntagme
`V`	:	verbe
`by`	:	« by »
`p`	:	préposition
`I`	:	impératif
`W0`, `W1`, `W2`	:	extraction wh-NP
`pW0`, `pW1`, `pW2`	:	extraction wh-PP
`N0`, `N1`, `N2`	:	proposition relative, argument NP relativisé avec un mot du type « wh- »
`Nc`, `Nc0`, `Nc1`, `Nc2`	:	proposition relative, argument NP relativisé avec un mot qui n'est pas du type « wh- »
`Npx`, `Npx1`, `Npx2`	:	proposition relative, argument PP relativisé

Nous voulons obtenir à partir de ces arbres les motifs de graphes de dépendances correspondant. La première méthode que nous avons envisagée fut de trouver un moyen permettant de convertir les arbres XTag en fragments d'arbre syntaxique (au

format de Stanford) puis de transformer ces fragments d'arbres en graphe de dépendances grâce à l'outil de dérivation de dépendances intégré dans l'analyseur de Stanford. Le problème de cette méthode est que premièrement la transformation de XTag en constituants de Stanford n'est pas simple, mais surtout que la transformation en dépendances nécessite des arbres entiers. C'est pourquoi nous avons décidé d'utiliser une autre approche consistant à associer à chaque arbre XTag plusieurs phrases réalisant la variation syntaxique puis à extraire manuellement les motifs de dépendances correspondant aux variations syntaxiques observées à partir des analyses de ces phrases.

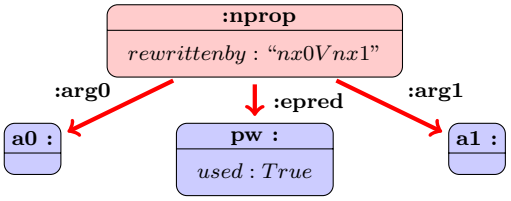
Nous avons produit les phrases de manière à avoir un vocabulaire varié, avec l'utilisation d'un maximum de combinaisons temps/personne/nombre, afin de pouvoir identifier le motif le plus général possible. Par exemple, nous avons associé à l'arbre `nx0Vnx1` les phrases :

- « John loves Mary »
- « I'm sending a mail »
- « They didn't take the train »

4.2.2 Création des règles de base

La création des règles de base associées à une famille se déroule en trois étapes. Dans un premier temps, nous associons un motif de transformation à la famille, puis nous associons des motifs de filtrage à chaque arbre, et pour finir nous construisons les règles de réécriture à partir de ces données.

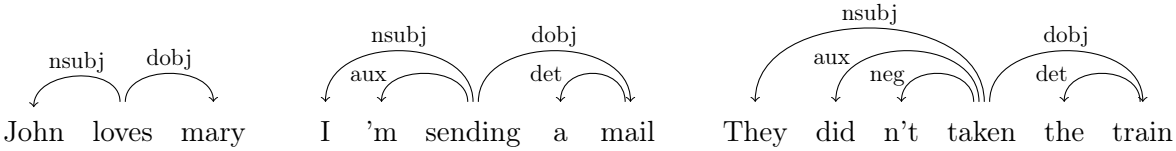
La première étape consiste donc à définir le motif de transformation que nous souhaitons utiliser pour avoir un étiquetage sémantique normalisé de la famille. Il suffit d'identifier le nombre d'arguments sémantiques que la famille a, et de créer un motif ajoutant un nœud de classe **nprop** en le liant au prédicat et à ses arguments avec les bonnes relations (e.g., **arg0**, **arg1**, ...). Nous utilisons une convention de dénomination pour les nœuds afin de pouvoir lier le motif de filtrage au motif de transformation. Le nœud représentant le prédicat s'appelle **pw**, ceux représentant ses arguments s'appellent **a0**, **a1**, ... ; **an**, les prépositions s'appellent **pp0**, **pp1** ; et les pronoms relatifs s'appellent **pr**. La famille des verbes transitifs `nx0Vnx1` possède deux arguments et aura donc le motif de transformation suivant :



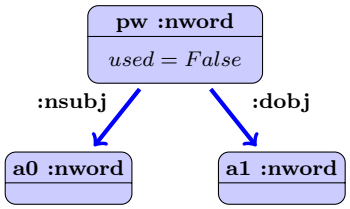
Ce motif de transformation représente ce que l'on souhaite rajouter au motif de filtrage de chaque arbre. Pour obtenir le motif de transformation d'un arbre, il faudra

donc combiner le motif de filtrage et le motif de transformation de la famille pour que la transformation ne supprime aucun nœud ni arc filtré.

Pour créer le motif de filtrage d'un arbre, nous récupérons tout d'abord pour chaque phrase d'exemple les 5 premières analyses correspondantes en constituants et en dépendances. Nous créons des motifs de filtrage basés uniquement sur les dépendances, mais les constituants nous permettent d'identifier plus facilement les bonnes analyses. Nous cherchons donc la bonne analyse pour chaque phrase puis essayons d'extraire le motif de filtrage permettant de capturer les liens entre le prédicat et ses arguments. Par exemple à partir des trois graphes de dépendances :



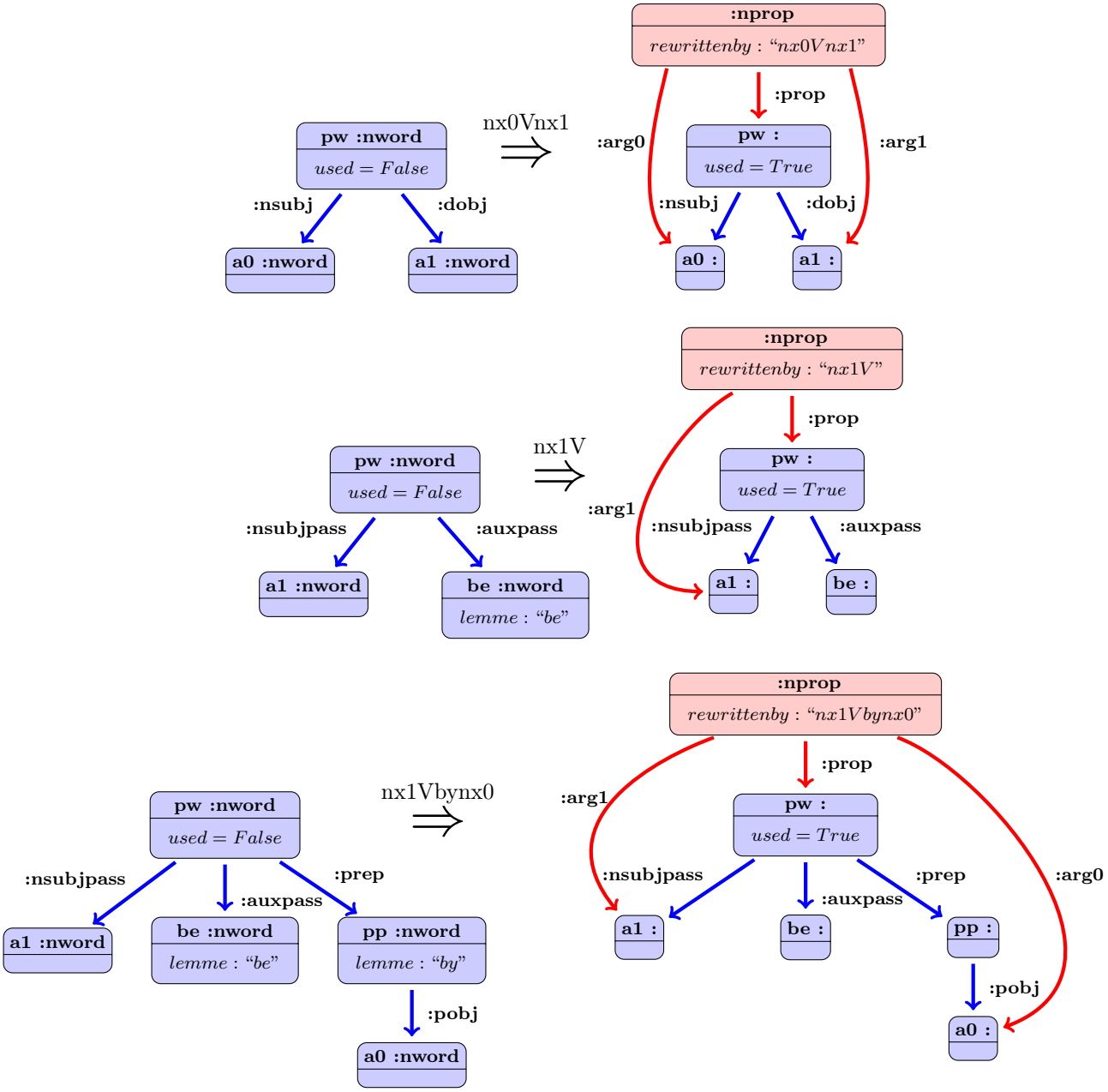
correspondant aux bonnes analyses des phrases associées à l'arbre `nx0Vnx1`, nous extrayons le motif de graphe :



Ce motif filtre les triplets de nœuds **pw**, **a0** et **a1** de type **nword** tel que **pw** ai un arc sortant de type **nsubj** allant vers **a0**, et un arc sortant de type **dobj** allant vers **a1**. De plus, nous ajoutons le fait que le prédicat n'a pas déjà été utilisé pour éviter par exemple que la règle pour le passif court ne réétiquète un prédicat déjà étiqueté par le passif long. Il est important de noter que nous utilisons les même noms que pour le motif de transformation de la famille (i.e., **pw**, **a0**, **a1**, ...) pour le prédicat et ses arguments, car le motif de transformation doit faire référence à des éléments du motif de filtrage. Nous ne prenons pas en compte les auxiliaires et la négation car ils sont optionnels. Nous pouvons remarquer, que le motif de filtrage obtenu est très simple alors que si l'on avait réalisé un motif à partir des constituants cela aurait été nettement plus compliqué car les étiquettes auraient varié selon la conjugaison du verbe, et nous aurions été obligé de prendre en compte les auxiliaires ainsi que la négation car ils sont intégrés dans la structure en constituants.

Une fois tout cela effectué nous pouvons générer les règles de réécriture de chaque arbre. Le motif de filtrage est gardé tel quel. Le motif de transformation correspond au motif de transformation de la famille auquel nous rajoutons tous les nœuds et arcs existants dans le motif de filtrage afin de ne supprimer aucun élément. De plus, pour les arbres qui ne réalisent pas tous les arguments de la famille (e.g., le passif

court), les nœuds du motif de transformation n'existant pas dans le motif de filtrage, sont retirés de ce motif ainsi que tous les arcs qui y sont connectés. Nous obtenons donc pour la famille `Tnx0Vnx1` les règles de réécriture suivantes pour l'actif, le passif court et le passif long.

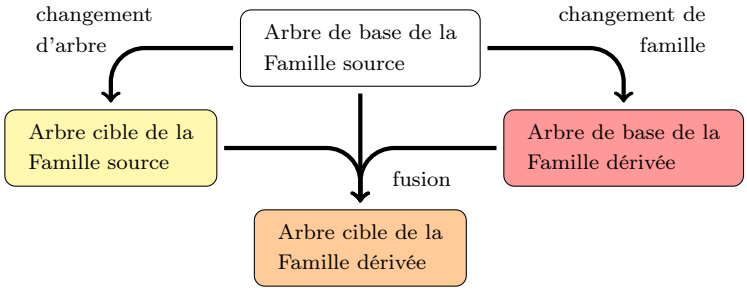


Nous avons ainsi crée 56 règles de réécriture pour les familles de verbes transitifs, intransitifs, et ditransitifs. Lors de la création de ces règles il est apparut que les familles de règles que nous avions créé avaient de grandes ressemblances. C'est pourquoi nous avons décidé de dériver les autres familles à partir de celles que nous avions déjà réalisées.

4.2.3 Dérivation de nouvelles règles

La création manuelle de règles est couteuse en temps et a tendance à augmenter le nombre d'erreurs. De plus, si l'on souhaite modifier le comportement général des règles il est nécessaire de modifier toutes les règles à la main. Nous avons donc élaboré une méthode semi-automatique de dérivation de règle basées sur les algorithmes de fusion de listes triés utilisé entre autre en informatique dans les gestionnaires de version pour fusionner plusieurs modifications faites sur un fichier. Les avantages d'une méthode semi-automatique sont que l'on a une meilleure cohérence des données, et que la correction de bug récurrent est plus facile.

Pour bien comprendre notre approche, considérons une famille source, une famille dérivée, un arbre de base et une arbre cible. La famille source est une famille pour laquelle nous avons déjà crée les motifs de filtrage correspondant à chacun de ses arbres. L'arbre de base est un arbre parmi ceux représentant l'actif, le passif et le passif court à partir duquel nous allons dériver le motif de filtrage de l'arbre cible. L'arbre cible est l'arbre pour lequel nous souhaitons dériver un motif de filtrage. Et enfin la famille dérivée est la famille pour laquelle nous voulons dériver les motifs de filtrages de ses arbres. Il est nécessaire de créer manuellement tous les motifs des arbres de base (actif, passif, passif court) de la famille dérivée pour dériver les autres motifs de filtrage. Comme nous le montre la figure ci-dessous, notre approche consiste à considérer l'arbre de base de la famille source comme élément de base, et l'arbre cible de la famille source et l'arbre de base de la famille dérivée comme des modifications de l'élément de base, puis de dériver l'arbre cible de la famille dérivée grâce à un algorithme de fusion [15].



Nous allons voir maintenant comment nous procédons exactement. Étant donné une famille F_i, nous notons $\mathcal{A}_i$ l'ensemble de ses arbres A_i^j et $\mathcal{A}_i^{\mathcal{B}} = \{A_i^j | j \in \mathcal{B} = \{actif, passif, passif\_court\}\}$ l'ensemble de ses arbres de base. Soit F_s la famille source pour laquelle tous les arbres de $\mathcal{A}_s$ ont un motif de filtrage associé et F_d la famille que l'on souhaite dériver pour laquelle seul les arbres de $\mathcal{A}_d^{\mathcal{B}}$ ont un motif de

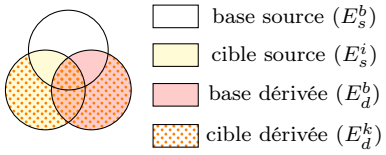
15. L'algorithme de fusion est celui utilisé par les gestionnaires de version pour fusionner deux changements qui ont été faits à un même fichier par deux personnes en parallèle.

filtrage associé. Nous souhaitons donc associer un motif de filtrage pour un maximum d'arbres de $\mathcal{A}_d - \mathcal{A}_d^{\mathcal{B}}$.

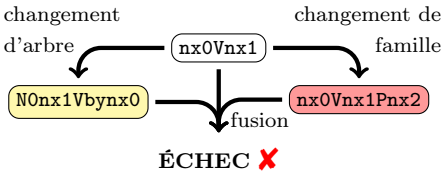
Pour chaque arbre $A_s^i \in \mathcal{A}_s - \mathcal{A}_s^{\mathcal{B}}$, nous testons pour chaque base $b \in \mathcal{B}$, si la fusion des noms de A_s^i et de A_d^b avec comme base le nom de A_s^b fonctionne et si c'est le cas nous regardons si il y a un arbre $A_d^k \in \mathcal{A}_d$ ayant ce nom. Si c'est le cas, cela signifie que les arbres A_s^i et A_d^k représentent des variations syntaxiques équivalentes dans les deux familles (e.g., les arbres `Nc0nx1Vbynx0` et `Nc0nx1Vpnx2bynx0` représentent des variations syntaxiques équivalentes car ils représentent une relativisation de l'agent dans une phrase au passif).

Pour obtenir le motif de filtrage de l'arbre A_d^k, nous appliquons la fusion entre les motifs de filtrage de A_s^i et de A_d^b avec comme base A_s^b. On procède alors comme précédemment pour obtenir les règles de réécriture associées aux arbres. Pour être sur que la fusion des motifs de filtrage se passe sans conflit, il est nécessaire que ces motifs soient définis avec les mêmes noms ce qui est assuré par l'utilisation de la convention de dénomination des nœuds. Les motifs de filtrage étant un ensemble de déclarations de contraintes pour lequel l'ordre n'est pas important, nous utilisons une version modifiée de l'algorithme de fusion fonctionnant sur des ensemble plutôt que sur des listes triées. Étant donné les ensembles de déclarations E_s^b, E_s^i, et E_d^b du motif de base de la famille source, du motif cible de la famille source et du motif de base de la famille dérivée, on a alors la formule suivante pour obtenir l'ensemble de déclarations E_d^k du motif cible de la famille dérivée :

$$E_d^k = ((E_s^i \cup E_d^b) - E_s^b) \cup (E_s^b \cap E_s^i \cap E_d^b)$$

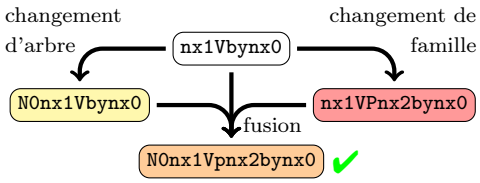


Prenons l'exemple de dérivation de la famille `Tnx0Vnx1Pnx2` à partir de la famille `Tnx0Vnx1`. Soit l'arbre `N0nx1Vbynx0` de `Tnx0Vnx1`. Nous essayons tous d'abord de faire une fusion en prenant comme base l'actif, mais la fusion ne fonctionne pas car les modifications sont en conflits comme nous le montre le schéma ci-dessous :



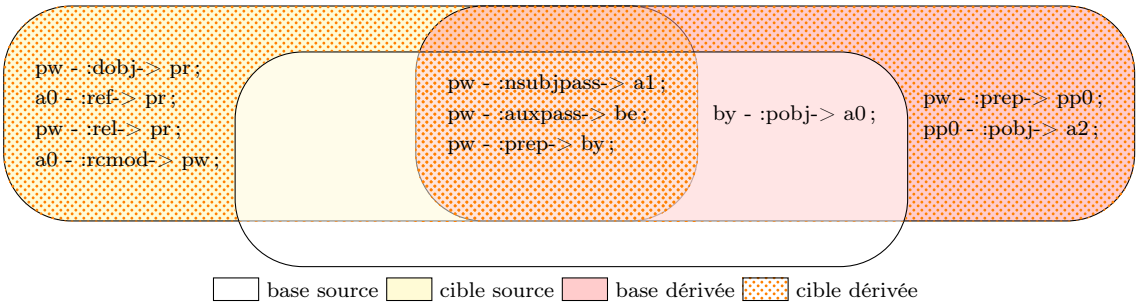
cible source	base source	base dérivée
N0	nx0	nx0
nx1	V	V
V	nx1	nx1
by		P
nx0		nx2

Nous essayons alors de prendre comme base le passif et là nous obtenons une fusion qui fonctionne et qui nous donne le nom d'arbre `N0Nx1Vpnx2bynx0`, comme nous montre la figure suivante :

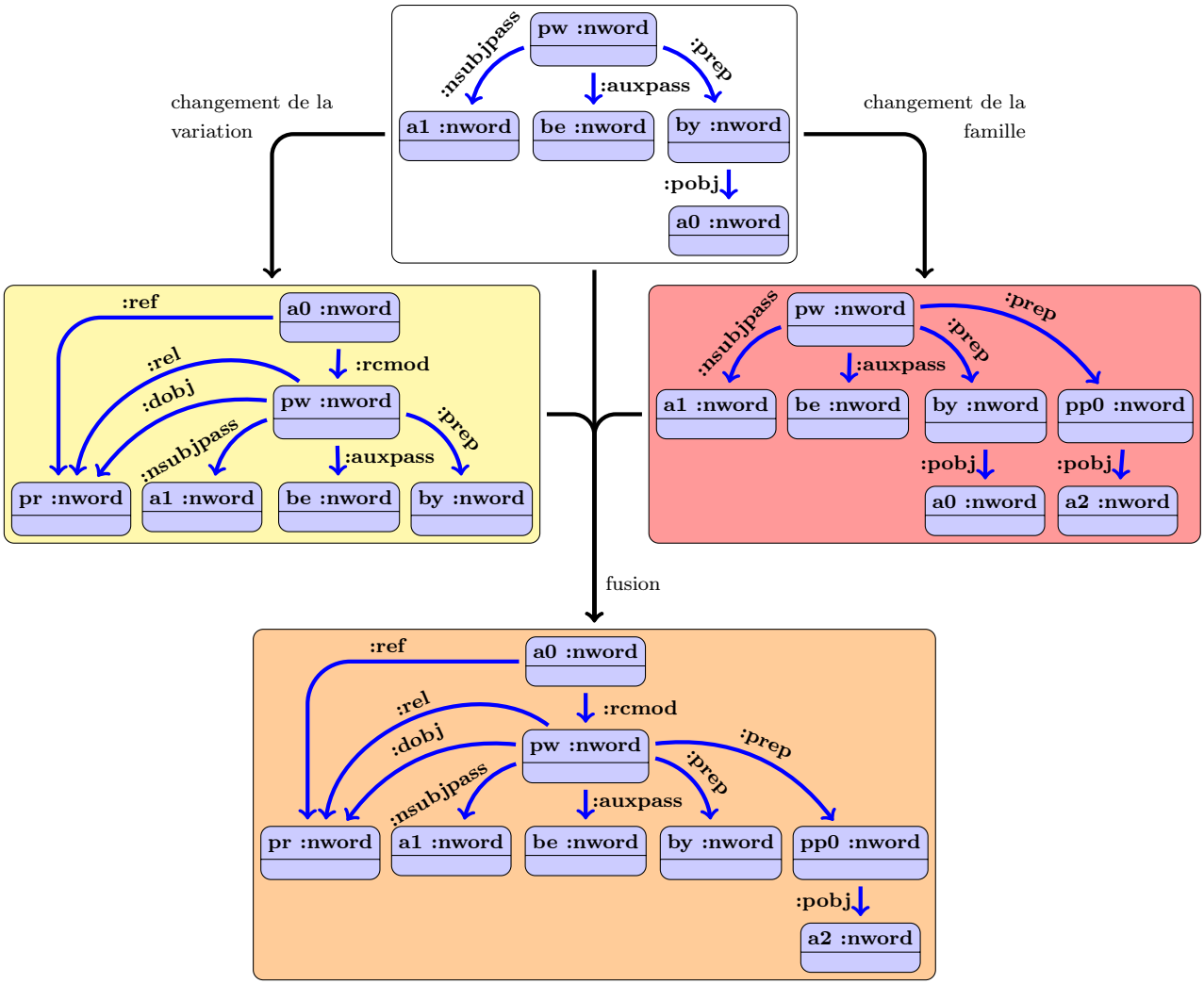


cible source	base source	base dérivée
N0	nx1	nx1
nx1	V	V
V	by	P
by	nx0	nx2
nx0		by
		nx0

L'arbre `N0nx1Vpnx2bynx0` existe bien dans la famille `TnxOVnx1Pnx2` et il parait clair que les variations `N0nx1Vbynx0` et `N0Nx1Vpnx2bynx0` représentent bien le même phénomène (i.e., une relativisation de l'agent au passif) dans les familles de verbes transitifs et ditransitifs avec préposition. Nous avons les ensembles de déclarations suivants pour les motifs de filtrage des différents arbres.



Ce schéma nous montre que pour le motif de filtrage de l'arbre cible de la famille dérivée nous gardons toutes les déclarations des motifs de filtrage des trois arbres utilisés pour la fusion sauf la déclaration « by - :pobj-> a0 ; » car elle est dans l'ensemble de déclarations du motif de filtrage de l'arbre de base de la famille source mais pas dans celui de l'arbre cible de la famille source. La figure ci-dessous permet de mieux visualiser les motifs de filtrage que nous avions et celui que nous avons dérivé.



Une fois nos motifs de filtrage obtenus nous les testons automatiquement pour voir s'ils filtrent bien une des 10 premières analyses de chaque exemple associé à la variation. Les motifs ne filtrant pas les exemples ont été analysés puis modifiés manuellement afin de bien représenter la variation. Par exemple, lors de la dérivation de la famille `Tnx0Vpnx1` à partir de la famille `Tnx0Vnx1`, le motif obtenu pour la variation `Nc1nx0VPnx1` (relativisation du second argument à l'actif) ne filtre pas les exemples car le motif veut que la préposition soit entre le verbe et l'argument prépositionnel alors que dans cette variation le verbe est directement lié à l'argument prépositionnel et que la préposition est lié au verbe. Il y a aussi certaines variations pour lesquelles il n'existe pas de correspondance dans la famille source. Par exemple lorsque l'on dérive la famille `Tnx0Vnx1Pnx2` à partir de la famille `Tnx0Vnx1` il n'y a pas de correspondance pour la variation `Nc2nx0Vnx1pnx2` (relativisation de l'argument prépositionnel à l'actif).

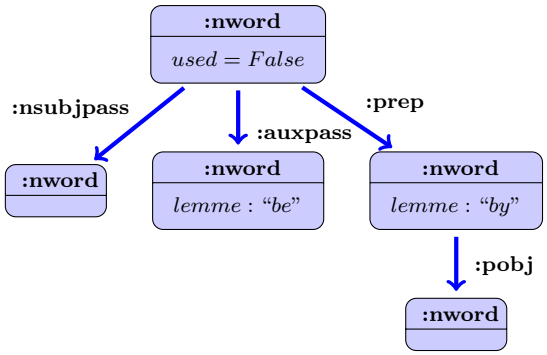
À partir des 3 familles de base, 37 nouvelles familles ont été crées. Pour cela nous avons du créer manuellement 111 règles pour les variations de base. Nous avons

obtenu 378 nouvelles règles générées automatiquement, et nous avons du créer manuellement 78 règles supplémentaires pour les variations qui ont été mal ou pas dérivées.

4.2.4 Création de la stratégie d'application des règles

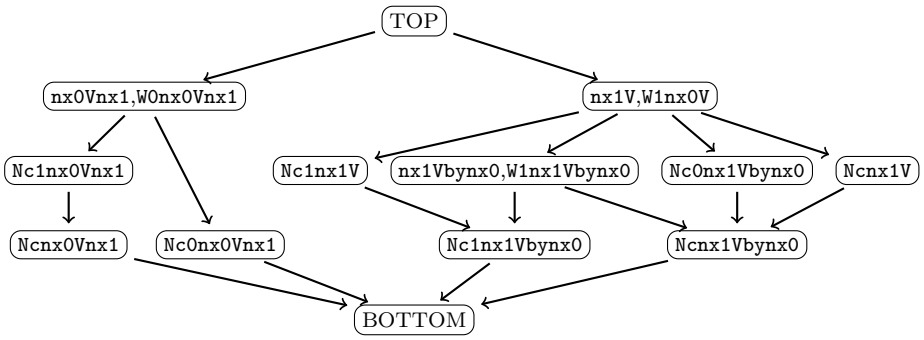
Une fois toutes ces règles créées, il est important d'établir un ordre d'application pour que les règles plus générales ne soient pas exécutées avant les règles les plus spécifiques. En effet, si nous appliquons la règle `nx1V` avant la règle `nx1Vbynx0` sur le graphe de la figure 2.6 (page 40) représentant la phrase « the city was destoyed by Caesar », la règle `nx1V` filtrera le verbe « destroyed », et l'étiquettera avec « city » comme $arg1$. La règle `nx1Vbynx0` ne pourra plus filtrer le verbe « destroyed » car il aura déjà été étiqueté et que nous ne permettons pas d'étiqueter un prédicat deux fois. L'argument « Caesar » qui aurait du être étiqueté comme $arg0$ de « destroyed » ne le sera donc pas, et l'étiquetage final sera incomplet.

Pour éviter cela nous ordonnons les règles automatiquement par ordre de spécificité afin d'obtenir un treillis sur les règles. Pour cela nous créons pour chaque variation le graphe correspondant au mieux au motif de filtrage associé. Le graphe correspondant à un motif de filtrage est le graphe contenant le moins d'information possible pour pouvoir être filtré par le motif de filtrage. Par exemple pour la variation `nx1Vnx0` on obtient le graphe :



Ensuite pour chaque variation nous testons si son motif de filtrage f filtre ou non les graphes g associés aux autres variations. Si f filtre g cela signifie que la variation associée à g est plus spécifique que celle associée à f. Par exemple, si on applique le motif de filtrage de `nx1V` au graphe de `nx1Vbynx0`, le filtrage fonctionne bien et nous déduisons donc que `nx1Vbynx0` est plus spécifique que `nx1V`. Par contre, le motif de filtrage de `nx0Vnx1` ne filtre pas le graphe de `nx1Vbynx0`, et l'inverse n'est pas vrai non plus, donc ces variations sont considérées comme indépendantes. Si deux règles sont mutuellement plus spécifique l'une que l'autre, cela signifie qu'elles sont équivalentes. Nous examinons alors les règles équivalentes afin de savoir si elles le sont bien ou si c'est que l'une des règles n'est pas assez spécifique ou générale. Si elles

sont bien équivalentes nous gardons une seule des règles, et sinon nous modifions les règles de façon à qu'elles aient les bonnes spécifications. Ci-dessous nous avons un sous-treillis du treillis des variations pour les verbes transitifs, les variations les plus générales étant en haut et les plus spécifiques en bas.



Ce treillis nous montre entre autre que les motifs de dépendances pour les variations affirmatives et interrogatives sont équivalentes [16] pour l'actif (`nx0Vnx1`, `W0nx0Vnx1`), le passif (`nx1Vbynx0`, `W0nx1Vbynx0`) et le passif court (`nx1V`, `W0nx1V`). Pour tous les groupes de règles équivalentes nous avons gardé uniquement une des règles afin d'avoir un système sans redondance. Le treillis montre aussi que les relatives sont des versions plus spécifiques de la variation qu'elles relativisent (e.g.,`Nc0nx0Vnx1` et `nx0Vnx1`). Nous aurions pu imaginer utiliser les règles générales dans les règles plus spécifiques pour factoriser le code, et avoir un système plus performant, mais le problème est qu'une règle considérée plus générale ne filtre pas forcément de la même façon qu'une règle spécifique. Par exemple, le motif de la variation `nx0Vnx1` est plus générale que la variation `Nc0nx0Vnx1`, mais le nœud connecté au verbe par la relation *nsubj* représente le premier argument du prédicat dans le motif de `nx0Vnx1`, et la relative dans le motif de `Nc0nx0Vnx1`. Les familles pour les verbes ergatifs et intransitifs sont équivalentes mais étiquettent le prédicat différemment. Nous avons décidé de garder uniquement la famille des verbes intransitif et nous allons voir plus tard comme nous gérons les variations d'étiquetage dues au lexique.

Une fois notre treillis obtenus nous créons une stratégie dans laquelle les règles les plus spécifiques sont appliquées en premier, et où chaque règle est appliquée autant de fois que possible avant de passer à la suivante. En effet, le système doit transformer toutes les occurrences des motifs plus spécifiques avant de passer aux motifs plus généraux. Pour le treillis ci-dessus nous obtenons la stratégie suivante [17] :

$$Ncnx0Vnx1* \mid Nc0nx0Vnx1* \mid Nc1nx1Vbynx0* \mid Ncnx1Vbynx0* \mid Nc1nx0Vnx1* \mid$$
$$Nc1nx1V* \mid nx1Vbynx0* \mid Nc0nx1Vbynx0* \mid Ncnx1V* \mid nx0Vnx1* \mid nx1V*$$

16. Ce qui est cohérents avec la description des dépendances faite dans [de Marneffe *et al.*, 2006]
17. Pour rappel, $r*$ signifie que l'on applique la règle r autant que possible, et $r_1|r_2$ signifie que l'on applique la règle r_1 puis la règle r_2

4.2.5 Augmentation du lexique à partir de PropBank

Le lexique associé à XTag étant plutôt limité et les réalisations sémantiques étant toujours les même sauf pour les ergatifs, nous avons décidé d'utiliser un autre lexique ayant plus d'entrées et de variations sémantiques. Notamment, nous souhaitons gérer des variations comme l'instrumentalisation du sujet (38), et avoir une annotation cohérente entre des verbes similaires (39).

(38) a. « [John]^{arg0} [broke]^{pred} [the window]^{arg1}. »

b. « [The hammer]^{arg2} [broke]^{pred} [the window]^{arg1}. »

(39) a. « [John]^{arg0} [killed]^{pred} [the president]^{arg1}. »

b. « [The president]^{arg1} [died]^{pred}. »

(40) « [He]^{arg0} [would]$^{argm-mod}$ [n't]$^{argm-neg}$ [accept]^{pred} [anything of value]^{arg1} from [those he was writing about]^{arg2}. »

Pour extraire ce lexique nous avons utilisé PropBank [Palmer *et al.*, 2005], une annotation sémantique de la section Wall Street Journal du Penn TreeBank2 [Marcus *et al.*, 1993]. Cette annotation contient des entrées qui pour une occurrence de verbe, donne le sens associé au verbe et ses arguments sémantiques comme le montrent les exemples ci-dessus. Pour chaque annotation nous récupérons la variation correspondante en transformant l'arbre syntaxique associé en graphe de dépendances puis en appliquant notre système de réécriture dessus. Nous définissons ainsi une association entre arguments syntaxiques et arguments sémantiques pour chaque couple prédicat-famille de verbe. Pour l'exemple (39b), Afazio nous dit que « died » est analysé comme verbe intransitif avec « The président » comme $arg0$ alors que PropBank le considère comme un $arg1$. Nous ajoutons donc a notre lexique l'entrée :

$$die, \texttt{nx0V}, \{arg0 \rightarrow arg1\}$$

qui dit que lorsque la famille `nx0V` filtre un graphe avec le prédicat *die*, elle doit considérer $arg0$ comme un $arg1$.

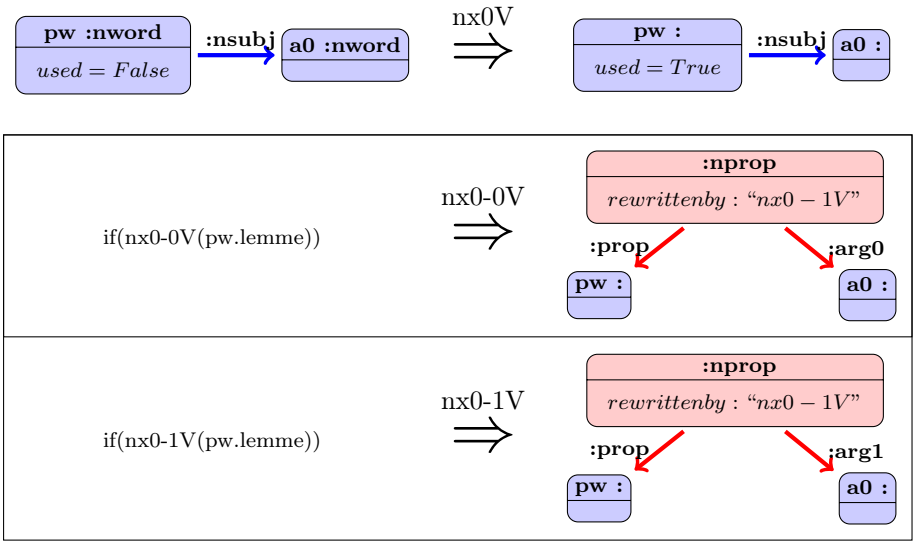
En réalisant cette extraction, il est apparu que certains prédicats avaient plusieurs distributions possibles pour la même famille de verbes. Une façon de résoudre ce problème, est de trouver un moyen de désambigüiser les distributions en ajoutant par exemple des types aux arguments sémantiques, mais cela ne désambigüiserait pas tous les cas et cela implique de connaitre les types sémantiques de chaque prédicat. Bien que cette solution soit la meilleure, nous avons décidé pour des contraintes de temps de prendre la distribution apparaissant le plus de fois dans PropBank. Nous avons aussi découvert qu'il nous manquait des familles pour pouvoir tout gérer. Nous avons donc créé de nouvelles familles permettant de gérer les verbes avec deux et trois compléments prépositionnels.

Une fois le lexique créé nous l'avons associé aux règles de manière à avoir un motif de transformation différent selon le prédicat filtré. Pour cela nous avons créé des fonctions prenant un lemme en argument et retournant vrai si le lemme appartient au lexique du couple distribution-famille souhaité. Par exemple, pour la famille `TnxOV`, nous avons entre autre les deux fonctions suivantes :

$$nx0\text{-}0V(lemme) : lemme \in \{run, eat, \dots\}$$

$$nx0\text{-}1V(lemme) : lemme \in \{die, sink, \dots\}$$

Nous réutilisons ensuite ces fonctions pour étiqueter différemment le graphe filtré selon le lexique. Pour la famille `TnxOV` avec les distributions ci-dessus, nous obtenons la règle suivante (les blocs représentent une alternative, voir page 36) :



4.2.6 Évaluation sur le corpus PropBank

Nous avons réalisé une évaluation de notre système d'étiquetage de rôles sémantiques sur le corpus de PropBank. Un argument est considéré comme bien reconnu s'il est lié au bon mot et si le rôle sémantique associé correspond à l'annotation de PropBank. La précision est la proportion d'arguments prédits par le système qui sont corrects. Le rappel est la proportion d'argument de PropBank qui sont prédits par le système. La F-mesure est la moyenne harmonique entre la précision et le rappel. Les résultats sont :

Arguments	**0**	**1**	**2**	**3**	**4**	**5**	**a**	**m**	**Total**
Rappel	68,4%	68,2%	62,4%	47,2%	57,6%	5,3%	0,0%	64,4%	66,1%
Précision	88,0%	80,2%	76,4%	83,1%	83,3%	50,0%	—	75,0%	80,6%
f-mesure	77,0%	73,7%	68,7%	60,2%	68,1%	9,5%	—	69,3%	72,6%

La précision (80%) est comparable aux résultats obtenus lors de la campagne du CoNLL 2005 [Carreras & Màrquez, 2005] (basé principalement sur PropBank) où les 8 meilleurs systèmes (sur un total de 20 sytèmes) ont des précisions allant de 76,55% à 82,28%. Le rappel est un peu plus bas (pour le CoNLL 2005 ils vont de 64,99% à 76,78%) pour principalement deux raisons : l'analyseur de Stanford ne donne pas de bonne analyse, ou la règle de réécriture nécessaire à l'étiquetage n'existe pas.

4.3 Normalisation des variations nominales

Pour détecter les implications textuelles il est nécessaire de normaliser les représentations le plus possible. Nous avons vu précédemment comment normaliser les variations verbales comme (41a) et (41b), mais un prédicat peut également être réalisé par un nom comme dans les phrases (41c) et (41d).

(41) a. « Rome was destroyed by the barbarians. »

b. « The barbarians have destroyed Rome. »

c. « Rome's destruction by the barbarians. »

d. « Barbarians destruction of Rome. »

Nous avons donc décidé de normaliser les variations nominales de la même manière que les variations verbales afin de pouvoir considérer toutes les phrases de (41) comme des paraphrases. Pour cela, nous nous sommes basé sur NomLexPlus [Meyers *et al.*, 2004] un dictionnaire des nominalisation de l'anglais qui en plus de donner les compléments possibles qu'un nom peut avoir lie ces compléments aux arguments du verbe correspondant. NomLexPlus est composé des 1025 entrées de NomLex [Macleod *et al.*, 1998b] développées manuellement auxquelles ont été rajouté semi-automatiquement plus de 7000 entrées à partir de diverses ressources dont NomLex et ComLex. Nous allons maintenant voir comment nous avons utilisé ces ressources pour créer automatiquement les règles de normalisation des variations nominales. La figure 4.1 montre comment les différentes ressources sont utilisées pour obtenir les règles de réécriture. Pour commencer nous avons extrait automatiquement le lexique de sous-catégorisation de NomLexPlus ① en dépliant ses entrées factorisées ③. Ensuite nous avons extrait les associations entre arguments syntaxiques et sémantiques ② présentes dans ComLex pour les intégrer dans le lexique obtenu à partir de NomLexPlus ④. Les résultats obtenus ont été filtrés afin d'obtenir des résultats linguistiquement corrects ⑤. Pour finir, nous avons établi une correspondance entre réalisation nominale et graphe de dépendances ⑥ permettant de dériver les règles de réécriture ⑦ à partir du lexique résultant.

4.3.1 Dépliage de NomLexPlus

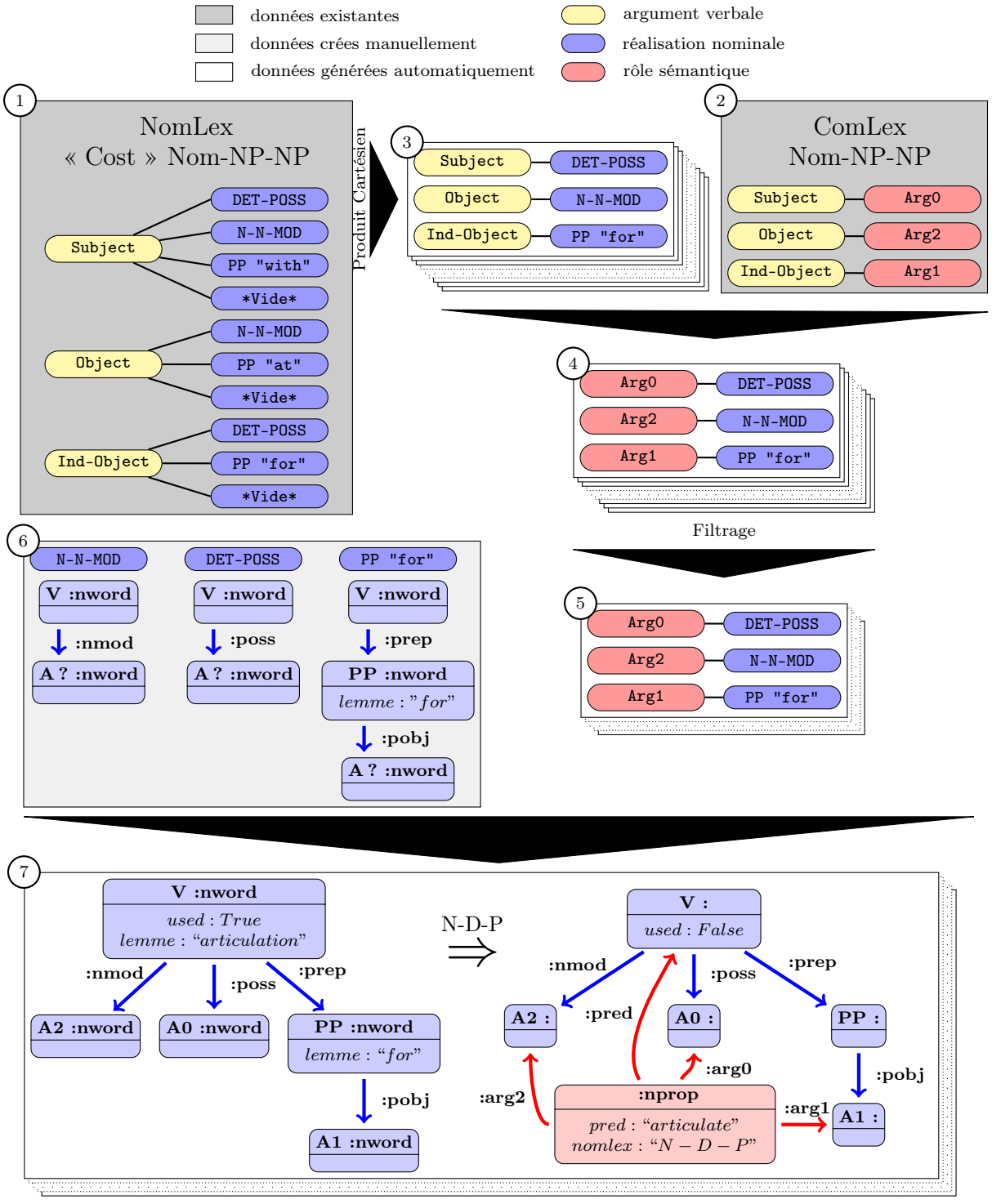


FIGURE 4.1 – Idée générale de la combinaison des ressources

```
(NOM :ORTH "articulation"
     :PLURAL *NONE*
     :VERB "articulate"
     :NOM-TYPE ((VERB-NOM))
     :VERB-SUBJ ((PP :PVAL ("by")))
     :VERB-SUBC ((NOM-NP :SUBJECT ((N-N-MOD)
                                   (DET-POSS)
                                   (PP :PVAL ("by")))
                         :OBJECT ((DET-POSS)
                                  (N-N-MOD)
                                  (PP :PVAL ("of")))
                         :REQUIRED ((OBJECT)))
                 (NOM-NP-PP :SUBJECT ((N-N-MOD)
                                      (DET-POSS)
                                      (PP :PVAL ("by")))
                            :OBJECT ((DET-POSS)
                                     (N-N-MOD)
                                     (PP :PVAL ("of")))
                            :PVAL ("with"))
                 (NOM-INTRANS :SUBJECT ((N-N-MOD)
                                        (DET-POSS)
                                        (PP :PVAL ("of" "by")))
                              :REQUIRED ((SUBJECT))))
```

FIGURE 4.2 – Entrée NomLexPlus pour « articulation »

Le plus gros défaut de NomLexPlus, est de ne pas être directement utilisable. En effet, cette ressource représente les informations sous un format factorisé, et le dépliage de la ressource nécessite la prise en compte d'un certains nombre de cas particuliers listés dans le manuel. Chaque entrée décrit le cadre de sous-catégorisation d'un prédicat nominal. La figure 4.2 représente l'entrée pour le nom « articulation ». Les éléments `ORTH`, `PLURAL`, `VERB`, `NOM-TYPE` et `VERB-SUBJ` représentent respectivement le lemme du nom, le lemme au pluriel, le type de nominalisation, et la liste des réalisations possibles où le sujet verbal peut se trouver. L'élément `VERB-SUBC` énumère les cadres de sous-catégorisation possibles pour l'entrée. Les noms des sous-catégorisations sont ceux de ComLex représentant les compléments verbaux ajoutés au sujet préfixés par `NOM-`. Par exemple, dans l'entrée « articulation », `NOM-NP` représente le complément verbal `NP` et correspond à la classe des verbes transitifs, `NOM-NP-PP` représente la classe des verbes ditransitifs, et `NOM-INTRANS` la classe de verbes intransitifs. À chaque sous-catégorisation, est associé un ensemble de réalisations nominales possibles pour chacun de ses arguments verbaux. Par exemple, pour le cadre `NOM-NP` de « articultation » l'argument verbal `SUBJECT` peut être réalisé comme argument du nom en tant que `N-N-MOD`, `DET-POSS` ou `PP :PVAL ("of")`. Les réalisations nominales possibles sont les pré-noms modifieurs du nom `N-N-MOD` (42a), les déterminants possessifs `DET-POSS` (42b), les groupes prépositionnels `PP :PVAL ("prep")` (42c), ainsi que quelques autres cas plus rares permettant de gérer les compléments obliques comme (42d). Le dépliage d'une entrée doit respecter un certain nombre de contraintes définies dans le manuel d'utilisation de NomLex [18] pour

18. `http://nlp.cs.nyu.edu/meyers/nombank/those-other-nombank-dictionaries.pdf`

éviter de surgénérer les cadres de sous-catégorisation nominaux.

(42) a. « The white house assessment »

b. « John's assessment of the situation »

c. « The assessment of the government »

d. « Their assesment of whether it was a good idea or not »

La première contrainte est la contrainte d'unicité qui contraint les arguments verbaux à n'être assignés qu'une seule fois, et les réalisations autres que `N-N-MOD` à n'apparaitre qu'une fois dans un cadre de sous-catégorisation nominal. Ainsi dans la phrase (43), le `DET-POSS` ne peut pas être associé à l'argument `SUBJECT` car `PP :PVAL ("by")` l'est forcément, il est donc associé à `OBJECT`. Pour les phrases (44a) et (44b) on a ainsi un argument différent associé à la réalisation `DET-POSS`. Pour l'unicité des réalisations, cela signifie que la phrase (45a) ne peut pas être associée à une des phrases (45b) ou (45c) mais devra être associée à (45d). La réalisation `N-N-MOD`, peut être associée à plusieurs arguments comme dans la phrase (46) où le premier `N-N-MOD` est associé à `SUBJECT` et le second à `OBJECT`. Il y a cependant des contraintes d'ordre que nous allons voir pour savoir à quel argument est associé chaque `N-N-MOD`.

(43) « An analysis articulation by Mr Mason »

(44) a. « Mary's description of Mother »

b. « Mother's description by Mary »

(45) a. « Mary's mother's description »

b. « Mary described Mother »

c. « Mary's mother described Mary »

d. « Mary's mother described »

(46) « The U.S. debt accumulation »

Les contraintes d'ordre permettent de savoir quel argument verbal associer à un modifieur pré-nominal. Si il y a plusieurs `N-N-MOD` et `DET-POSS`, ils doivent apparaitre dans l'ordre suivant :

$$\texttt{SUBJECT} > \texttt{IND-OBJ} > \texttt{OBJECT} > \texttt{OBLIQUE}$$

Les exemples (47) et (48) illustrent bien ces contraintes. La phrase (47a) ne peut pas être associée à la phrase (47b) car cela violerait la contrainte d'ordre $\texttt{OBJECT} > \texttt{SUBJECT}$. Par contre la phrase (48a) peut être associée à la phrase (48b) car elle respecte bien la contrainte d'ordre $\texttt{IND-OBJ} > \texttt{OBJECT}$.

(47) a. « "John Smith's school board appointment" »

b. « The school board appointed John Smith to some position »

(48) a. « The project's food allocation »

b. « Someone allocated food to the project »

Enfin les contraintes d'obligation permettent de forcer ou non des arguments verbaux à être réalisés. Par défaut, seuls les arguments `SUBJECT` et `OBJECT` sont optionnels. Ces contraintes peuvent être modifiées pour un cadre verbal grâce aux mots-clés `REQUIRED` et `OPTIONAL` qui permettent respectivement de forcer ou non un argument à être réalisé. De plus `OBJECT` est obligatoire pour un cadre `NOM-NP` si le cadre `NOM-INTRANS` appartient aussi à l'entrée. Il en est de même pour tous les paires de cadres `NOM-NP-*`, `NOM-*`. Il en est ainsi, car le cadre `NOM-*` permet de générer un sous-ensemble de `NOM-NP-*`, et lorsque l'on a les deux en même temps c'est que `SUBJECT` ne peut pas être réalisé de la même manière selon que `OBJECT` soit réalisé ou non. Ainsi, la phrase (49) ne peut pas être associée au cadre `NOM-NP` de l'entrée « articulation », mais doit l'être au cadre `NOM-INTRANS`.

(49) « articulation by John »

Pour déplier la ressource, nous commençons par faire le produit des ensembles de réalisations associés à chaque argument pour un cadre syntaxique. Pour représenter le fait qu'un argument peut être optionnel nous rajoutons avant de faire le produit cartésien une réalisation vide `*Vide*` à l'ensemble de réalisations correspondant. La partie de la contrainte d'unicité empêchant un argument verbal d'être réalisé plusieurs fois est respectée car pour chaque cadre nominal nous associons une seule réalisation à chaque argument verbal. Pour le cadre syntaxique `NOM-NP-PP` d'« articulation » nous obtenons donc la liste d'éléments de la table 4.1.

Nous appliquons ensuite un filtre qui garde uniquement les cadres nominaux satisfaisant la partie de la contrainte d'unicité qui empêche certaines réalisations syntaxiques d'apparaitre plusieurs fois dans un cadre nominal. Ce filtre supprime le cadre (5). Pour la contrainte d'ordre, on ajoute en indice de chaque réalisation pré-nominale l'ordre d'apparition souhaité. Nous obtenons ainsi pour l'entrée (2), l'association de $\texttt{N-N-MOD}_1$ à `SUBJECT` et de $\texttt{N-N-MOD}_2$ à `OBJECT`.

4.3.2 Dérivation de la correspondance syntaxe-sémantique

Nous avons vu comment obtenir les cadres nominaux qui associent un rôle verbal à chaque réalisation, mais nous souhaitons obtenir des rôles sémantiques pour avoir la même normalisation que pour les verbes. Pour cela nous utilisons les associations entre cadres verbaux et rôles sémantiques définis dans ComLex [Macleod *et al.*, 1998a] (voir figure 4.3). Le mot-clé `gs` donne les associations entre les arguments verbaux et les rôles sémantiques (les rôles sémantiques commençant à `1` nous les décrémentons tous de 1). La convention de dénomination n'étant pas exactement la même que pour NomLexPlus il est nécessaire de faire quelques transformations

id	SUBJECT	OBJECT	PVAL
1	N-N-MOD	DET-POSS	"with"
2	N-N-MOD	N-N-MOD	"with"
3	N-N-MOD	PP :PVAL ("of")	"with"
4	N-N-MOD	*VIDE*	"with"
5	DET-POSS	DET-POSS	"with"
6	DET-POSS	N-N-MOD	"with"
7	DET-POSS	PP :PVAL ("of")	"with"
8	DET-POSS	*VIDE*	"with"
9	PP :PVAL ("by")	DET-POSS	"with"
10	PP :PVAL ("by")	N-N-MOD	"with"
11	PP :PVAL ("by")	PP :PVAL ("of")	"with"
12	PP :PVAL ("by")	*VIDE*	"with"
13	*VIDE*	DET-POSS	"with"
14	*VIDE*	N-N-MOD	"with"
15	*VIDE*	PP :PVAL ("of")	"with"
16	*VIDE*	*VIDE*	"with"

TABLE 4.1 – Entrée NomLexPlus pour « articulation » dépliée.

(e.g., `obj2` est équivalent à `IND-OBJ`). Ainsi nous savons que pour `NOM-NP SUBJECT` et `OBJECT` font respectivement référence aux rôles sémantiques `arg0` et `arg1` ; et que pour `NOM-NP SUBJECT`, `OBJECT` et `IND-OBJ` font respectivement référence aux rôles sémantiques `arg0`, `arg2` et `arg1`. En appliquant ces informations sur les cadres nominaux obtenus précédemment, nous obtenons des cadres nominaux qui lient les réalisations nominales aux rôles sémantiques.

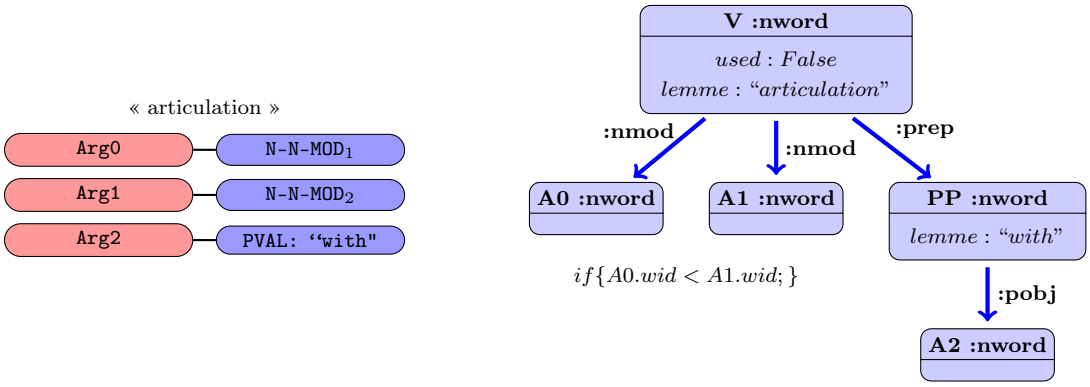
4.3.3 Spécification des règles de réécriture

Pour avoir la normalisation il nous faut associer des règles de réécriture à chaque cadre nominal. Pour cela, nous avons associé un motif de filtrage pour chaque type de réalisation comme nous le montre la boite ⑥ de la figure 4.1. Ensuite pour chaque cadre de sous-catégorisation que nous avons obtenu, nous prenons les motifs de graphe de dépendances associés à chacune des réalisations puis nous les fusionnons. Si le cadre contient plusieurs `N-N-MOD` et `DET-POSS`, nous rajoutons les contraintes sur les numéros d'apparition des mots pour avoir la bonne transformation. Enfin, nous rajoutons un nœud `:pred` relié au prédicat et à ses arguments.

Si nous prenons l'exemple du cadre nominal correspondant au cadre nominal (2) vue précédemment, puis que nous le combinons à ComLex, nous obtenons le cadre ci-dessous à gauche. En combinant ces réalisations nous obtenons le motif de filtrage ci-dessous à droite.

```
(vp-frame intrans :cs ()
                  :gs (:subject 1)
                  :ex "he went.")
(vp-frame np      :cs ((np 2))
                  :gs (:subject 1 :obj 2)
                  :ex "I bought the book.")
(vp-frame *np-np  :cs ((np 2) (np 3))
                  :gs (:subject 1 :obj 3 :obj2 2)
                  :ex "he gave his mother a big kiss.")
(vp-frame *np-pp  :cs ((np 2) (pp 3 :pval ("")))
                  :gs (:subject 1 :obj 2 :obj2 3)
                  :ex "she added the flowers to the bouquet.")
(vp-frame pp-pp   :cs ((pp 2 :pval ("")) (pp 3 :pval ("")))
                  :gs (:subject 1 :obj 2 :obj2 3)
                  :ex  they flew from London to Rome.")
```

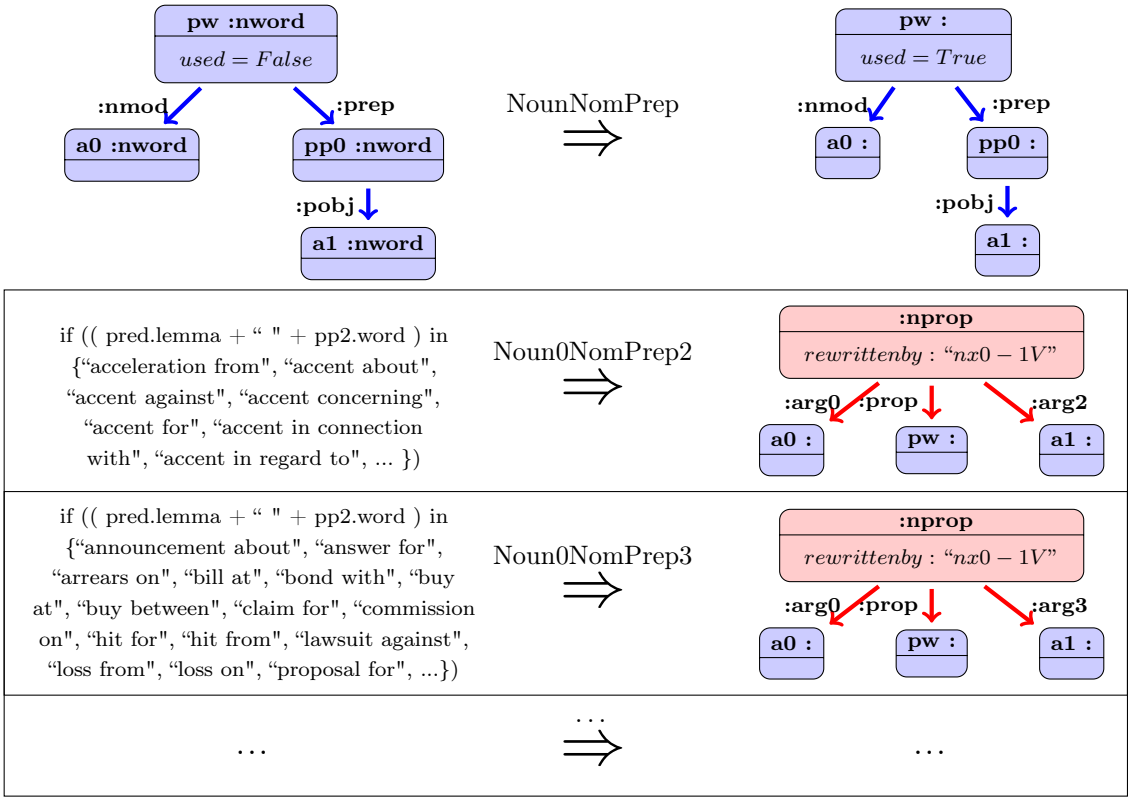
FIGURE 4.3 – Entrée ComLex définissant des cadres verbaux



Il suffit alors de rajouter le nœud `:pred` ainsi que les arcs permettant d'étiqueter les arguments du prédicat.

Nous avons obtenu au final 16 cadres de sous-catégorisation nominaux sur l'ensemble des entrées de NomLexPlus. Un même cadre peut apparaitre dans plusieurs nominalisations en ayant des associations différentes des rôles sémantiques. C'est pourquoi nous avons regroupé les règles nominales de la même façon que les règles verbales en ayant une règle pour chaque cadre avec des alternatives de transformations selon l'entrée lexicale qui a été filtrée. Nous avons donc au final 16 règles de filtrage permettant de récupérer un motif de graphe de dépendances et 423 réécritures possibles de ces motifs en fonction du lexique. Nous appliquons la même technique que pour la normalisation des verbes pour définir la stratégie.

On obtient les règles de réécriture en combinant nos trois ressources.



4.3.4 Évaluation sur le corpus CoNLL 2009

Nous avons évalué la combinaison de ce système de réécriture et de celui d'étiquetage sémantique pour les verbes sur le corpus de la campagne d'évaluation CoNLL 2009 [Hajič *et al.*, 2009].

	Précision	**Rappel**	**F-mesure**
Non-étiqueté	83,56%	65,36%	73,35
Étiqueté	72,66%	56,83%	63,78

Le score non-étiqueté donne la proportion de dépendances prédicats-arguments trouvées (pour les verbes et les noms), et le score étiqueté vérifie en plus que le type des dépendances est le bon. Le score global situe notre étiqueteur dans la moyenne des système du CoNLL 2009 (la F-mesure allant de 36,05 à 85,44) avec une bonne précision mais un rappel bas du en partie au fait que l'analyseur de Stanford ne donne pas de bonne analyse[19]. La F-mesure est inférieure à celle que le système avait sur PropBank car les structures syntaxiques associées aux noms sont plus ambigües et qu'il est donc plus difficile de bien les gérer.

19. Klein & Manning [2003] reportent une F-mesure de 86,3 sur la section 23 du Penn Treebank

4.4 Normalisation des variations sémantiques

Nous allons donc voir maintenant les différents traitements que nous appliquons à nos structures hybrides contenant des constituants, des dépendances et un étiquetage sémantique pour obtenir l'arbre syntaxique de la formule de logique du premier ordre associé. La représentation sémantique que nous utilisons est néo-Davidsonienne [Higginbotham, 2000] ce qui signifie qu'une variable existentielle est associée au prédicat, et que les arguments du prédicat sont définis avec des prédicats logiques additionnels permettant ainsi de gérer facilement les implications entre un prédicat réalisant deux de ses arguments où seulement un seul, comme dans l'exemple (50).

(50) **T :** « John sends a book to Mary »
SEM: $john(j) \wedge \exists B.(book(B) \wedge mary(m) \wedge \exists S.(send(S) \wedge arg0(S, j) \wedge arg1(S, B) \wedge arg2(S, m)))$
H : « a book is send »
SEM: $\exists B.(book(B) \wedge \exists S.(send(S) \wedge arg1(S, B))$
Implication : VRAI

Cette transformation d'une structure linguistique multi-niveaux vers une fomule de la logique du premier ordre est composée de 6 règles principales et de 3 sous-règles utilisées par les règles principales élaborées manuellement. L'idée générale de la construction sémantique est de créer des fragments d'arbres syntaxiques vides pour chaque nœud étiqueté par l'étiquetage sémantique, puis de les compléter et de les lier entre eux afin d'avoir un arbre syntaxique complet de formule de logique du premier ordre.

Les règles principales sont :

- **add_root**
  ajoute une racine syntaxique pour la formule.
- **add_head**
  ajoute un fragment d'arbre syntaxique composé uniquement d'une tête à chaque mot étiqueté par le module d'étiquetage sémantique.
- **add_structs**
  complète les fragments des arguments
- **link_structs**
  lie les fragments entre eux.
- **fill_structs**
  ajoute les prédicats logiques à la formule.
- **link_root**
  lie les prédicats des propositions au reste de la formule.

Les sous-règles utilisées sont :

- **head_scope** $(s : nprop) \rightarrow (c : semcon)$
  récupère la tête du fragment associé à une proposition et lie c à cette tête.
- **super_scope** $(mw : nword, s : nprop) \rightarrow (c : semcon)$
  récupère la tête du fragment devant être lié à la portée du fragment associé à mw par rapport à la proposition s, et lie c à cette tête.
- **exists_path** $(s : nword, d : nword) \rightarrow ()$
  vérifie si il existe un chemin de dépendances entre s et d.

La stratégie d'application de ces règles est :

```
add_root | add_head* | add_structs* | link_structs* | fill_structs* | link_root*
```

La terminaison du système est assurée par la conception des règles qui ne peuvent pas s'appliquer deux fois au même endroit car elle contiennent une négation de ce qu'elle ajoute dans le motif de filtrage.

4.4.1 Création et imbrication des fragments de formule

Lors de la création d'une formule logique, il est important de savoir comment s'imbriquent les différents fragments sémantiques, et plus précisément comment s'imbriquent les fragments sémantiques d'un prédicat et de ses arguments ainsi que les fragments sémantiques associés à plusieurs propositions connectées les unes aux autres (e.g, une subordonnée relative).

Nous imbriquons les fragments sémantiques d'une proposition dans leur ordre d'apparition dans le texte, mise à part le prédicat qui est toujours en dernière position. Cela permet ainsi d'avoir une version distributionnelle des événements qui est la plus commune (pour gérer les évènements de groupes il suffirait de faire remonter le quantificateur évènementiel au dessus des quantificateurs liés aux arguments). Par exemple dans la phrase (51a) nous avons une sémantique où il y a un évènement de « danser » pour chaque femme.

Un autre élément important, est de savoir pour les formules associées aux quantificateurs, ce qui va dans leurs restrictions et ce qui va dans leurs portées. De manière générale, on met les prédicats logiques associés à l'argument dans la restriction et ce qui est emboité dans la portée. Les propositions relativisées vont dans la restriction des arguments qui les relativisent. Les exemples (51) montrent les formules de logiques du premier ordre que nous souhaitons obtenir.

(51) a. « John dances with every woman »
$john(j) \wedge \forall w.(woman(w) \Rightarrow (\exists d.dance(d) \wedge arg0(d, j) \wedge arg1(d, w)))$

b. « Every man who dances with every woman, lives in Paris »
$\forall m.((man(m) \wedge \forall w.(woman(w) \Rightarrow \exists d.(dance(d) \wedge arg0(d, m) \wedge arg1(d, w)))) \Rightarrow$
$(\exists l.live(l) \wedge arg0(l, m) \wedge arg1(l, p)))$

c. « Every man dances with every woman who lives in Paris »
$\forall m.(man(m) \Rightarrow (\forall w.(woman(w) \wedge (\exists l.live(l) \wedge arg0(l,w) \wedge arg1(l,p))) \Rightarrow \exists d.(dance(d) \wedge arg0(d,m) \wedge arg1(d,w))))$

d. « Every man loves to dance with every woman »
$\forall m.(man(m) \Rightarrow (\exists d.(dance(d) \wedge arg1(f,d) \wedge arg0(d,m) \wedge \forall w.(woman(w) \Rightarrow arg1(d,w)))))$

Nous allons maintenant décrire étape par étape comment nous construisons les arbres syntaxiques de formules logiques, et nous utiliserons la phrase « Every man loves a woman » comme exemple.

La première étape consiste à ajouter une racine pour la formule logique ainsi que des têtes de fragments sémantique pour chaque mot étiqueté par le module d'étiquetage sémantique. Cette étape peux être vue comme une semaison de graines sémantiques donnant des bourgeons de sémantiques que l'on va ensuite faire fleurir. La figure 4.4 montre la structure syntaxique de la phrase « Every man loves a woman » sur laquelle on a fait bourgeonner l'arbre syntaxique de la formule sémantique grâce aux règles de réécriture **add_root** et **add_head**. Nous avons nommé certains nœuds de la figures 4.4 afin d'y faire référence dans les prochaines figures où nous allons successivement ajouter les structures syntaxiques aux têtes, les lier entre elles, puis ajouter les prédicats logiques.

La seconde étape consiste à ajouter les structures syntaxiques aux têtes de fragments de formules sémantiques que nous avons liées aux arguments et aux prédicats des propositions. Les structures syntaxiques que nous ajoutons sont toujours composées au minimum d'un quantificateur, d'une restriction et d'une portée qui sont accessibles à partir du nœud auquel est lié la tête afin de pouvoir y accéder dans les règles suivantes. Le quantificateur servira à connaitre quelles variables sont en arguments des différents prédicat logiques. La restriction et la portée permettront d'ajouter les prédicats logiques au bon endroit, et de lier les structures entre elles. Pour les arguments des propositions, la structure syntaxique que l'on ajoute à la tête dépend du quantificateur qui lui est associé. Si on a un quantificateur universel on associe une structure implicative à la tête, où l'a partie gauche de l'implication à le statut de restriction et la partie droit le statut de portée. Dans notre exemple, nous ajoutons à la tête **ha0** la structure de gauche de la figure 4.5. Pour les quantificateur existentiels, et les constantes nous ajoutons une conjonction à la tête qui combine le statut de restriction et de portée [20]. Nous ajoutons ainsi la structure de droite de la figure 4.5 à la tête **ha1**. Pour les têtes des prédicats nous ajoutons uniquement un quantificateur existentiel représentant la variable évènementielle puis une conjonction combinant le statut de restriction et de portée. Le fragment d'arbre du milieu

20. Il est uniquement important de distinguer la restriction et la portée quand le connecteur logique qui les lies n'est pas commutatif

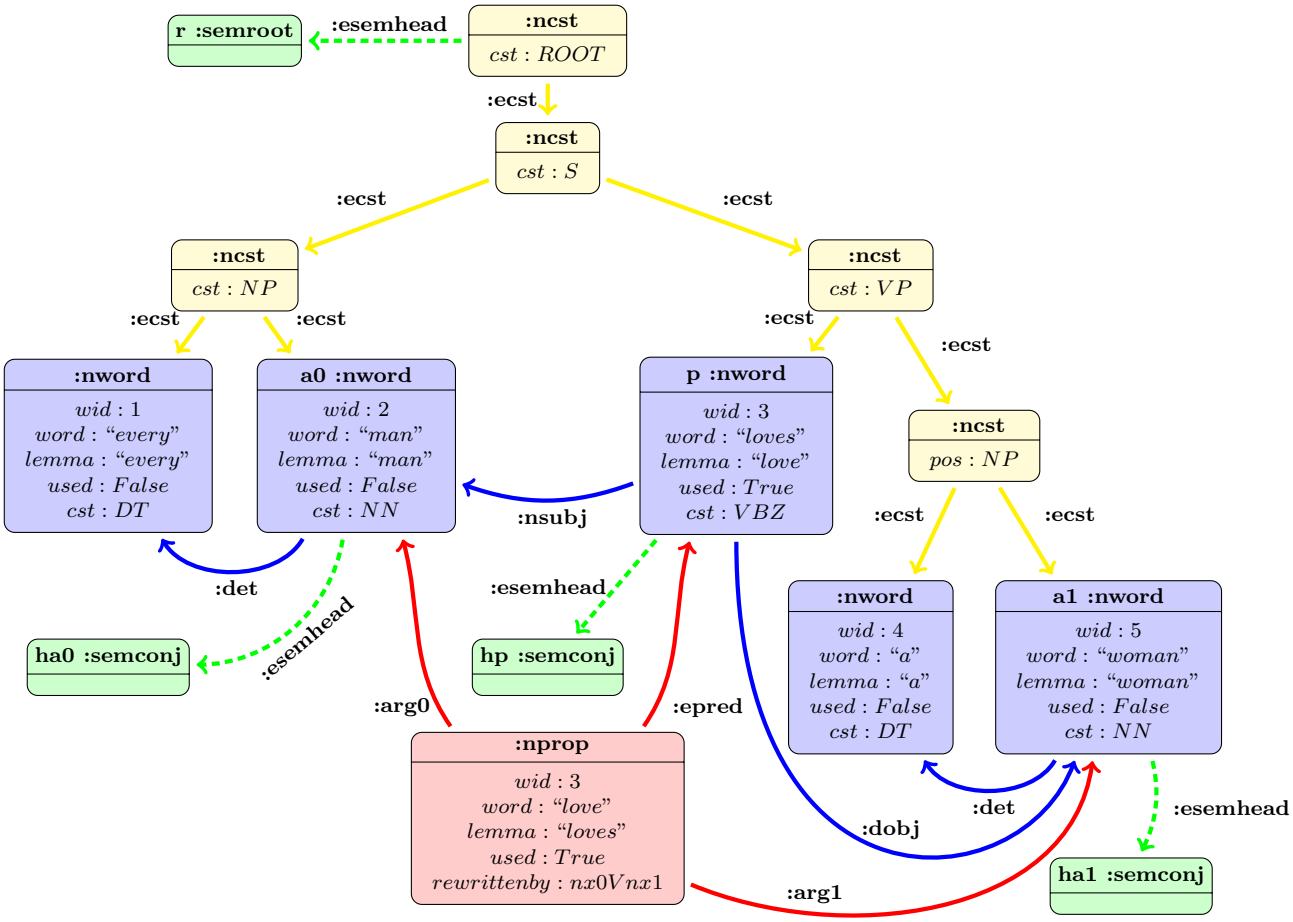


FIGURE 4.4 – Structure syntaxique de la phrase « Every man loves a woman » étiquetée sémantiquement et augmentée avec les bourgeons de sémantique.

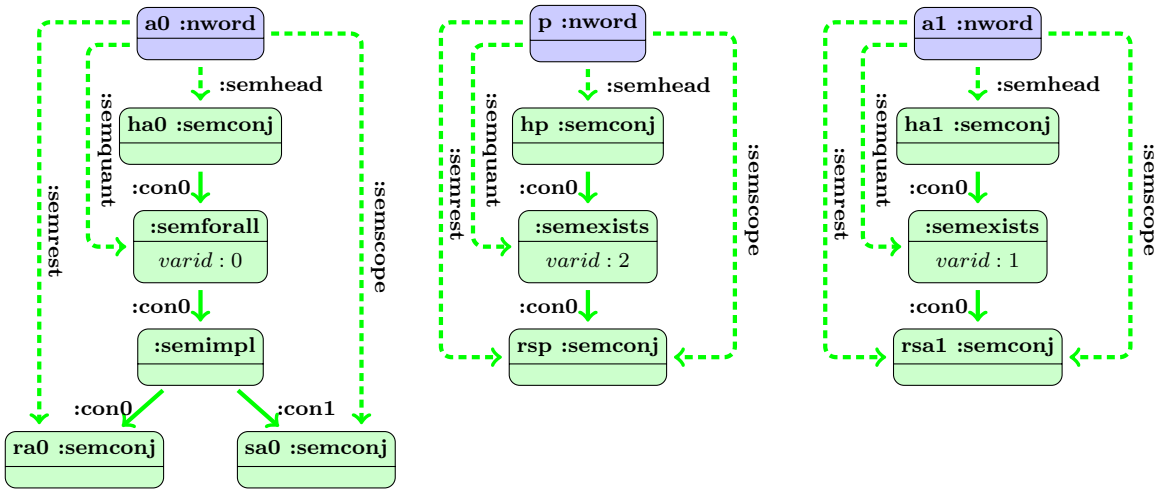


FIGURE 4.5 – Structures syntaxiques des fragments de formules sémantiques associées aux bourgeons de la figure 4.4.

de la figure 4.5 est ajouté au nœud **hp**. Nous associons un nouvel identifiant pour chaque quantificateur créé. Nous associons le lemme du mot associé pour représenter les constantes. Pour mieux gérer les constantes, nous souhaitons ajouter un système de dénomination de constantes permettant d'associer la même constante à « États-Unis » et « U.S.A. » afin de détecter que l'implication (52) est vraie. Cette étape est réalisée grâce à la règle **add_structs**.

(52) **T :** « John likes United-States »
lpo : $\exists v0.likes(v0) \wedge arg0(v0, john) \wedge arg0(v0, unitedstates)$
H : « John likes U.S.A. »
lpo : $\exists v0.likes(v0) \wedge arg0(v0, john) \wedge arg0(v0, usa)$
Implication : OUI

La prochaine étape sert à lier les différentes structures syntaxiques précédemment créées entre elles. La règle générale de liaison des fragments d'une proposition consiste à lier la portée d'un argument à la tête de l'argument apparaissant après elle dans la phrase, et la portée du dernier argument à la tête du prédicat. Il existe des cas particuliers de liaisons. Pour les subordonnées relatives on lie la restriction de l'argument relativisé à la tête de l'argument suivant ou du prédicat s'il n'y a qu'un seul argument. Les compléments phrastiques sont liés à la tête de leur premier argument et lient la restriction de leur prédicat à la tête du prédicat de la proposition englobante. Notre exemple nous montre uniquement un cas générale de liaison, où la portée **sa0** de l'argument **a0** est liée à la tête **ha1** de l'argument **a1**, où la portée **rsa1** de l'argument **a1** est liée à la tête **hp** de l'argument **a0** et où la portée

FIGURE 4.6 – Liaison des structures syntaxiques entre elles.

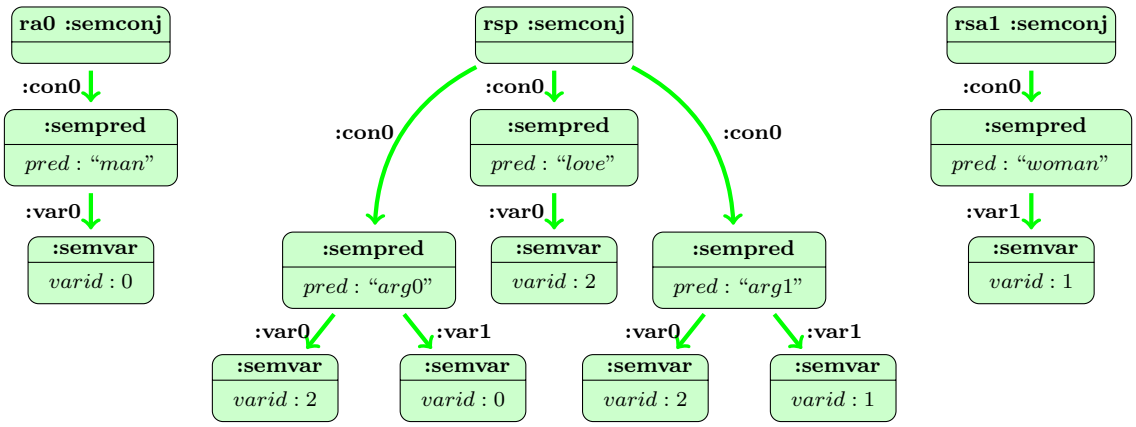


FIGURE 4.7 – Prédicats logiques ajoutés aux fragments d'arbre syntaxique de formules sémantiques de la figure 4.5

rsp du prédicat **p** n'est lié à aucune tête et représente une branche finale de la formule sémantique (c.f., figure 4.6). Ces liaisons ont été ajoutées à l'aide de la règle **link_structs**. L'annexe A contient des exemples de représentations obtenues pour les cas particuliers.

Nous allons maintenant voir l'étape qui sert à remplir les structures logiques que nous avons créé avec des prédicats logiques. Pour les arguments, nous ajoutons un prédicat logique correspondant au lemme du mot associé que nous appliquons à une variable ayant le même identifiant que le quantificateur que nous avons précédemment lié au nœud. Pour les prédicats, nous ajoutons un prédicat logique pour le représenter appliqué à une variable ayant le même identifiant que le quantificateur que nous lui avons précédemment lié, ainsi qu'un prédicat logique pour le lier à chacun de ses arguments qui est appliqué à des variables ayant les mêmes identifiants que ceux des quantificateurs du prédicat et de l'argument. Cette étape est réalisée à l'aide de la règle **fill_structs**.

Enfin la dernière étape lie la racine de la sémantique aux têtes qui n'ont pas été liées. Cette étape est réalisée à l'aide de la règle **link_root**. On obtient ainsi pour notre exemple la représentation finale de la figure 4.8.

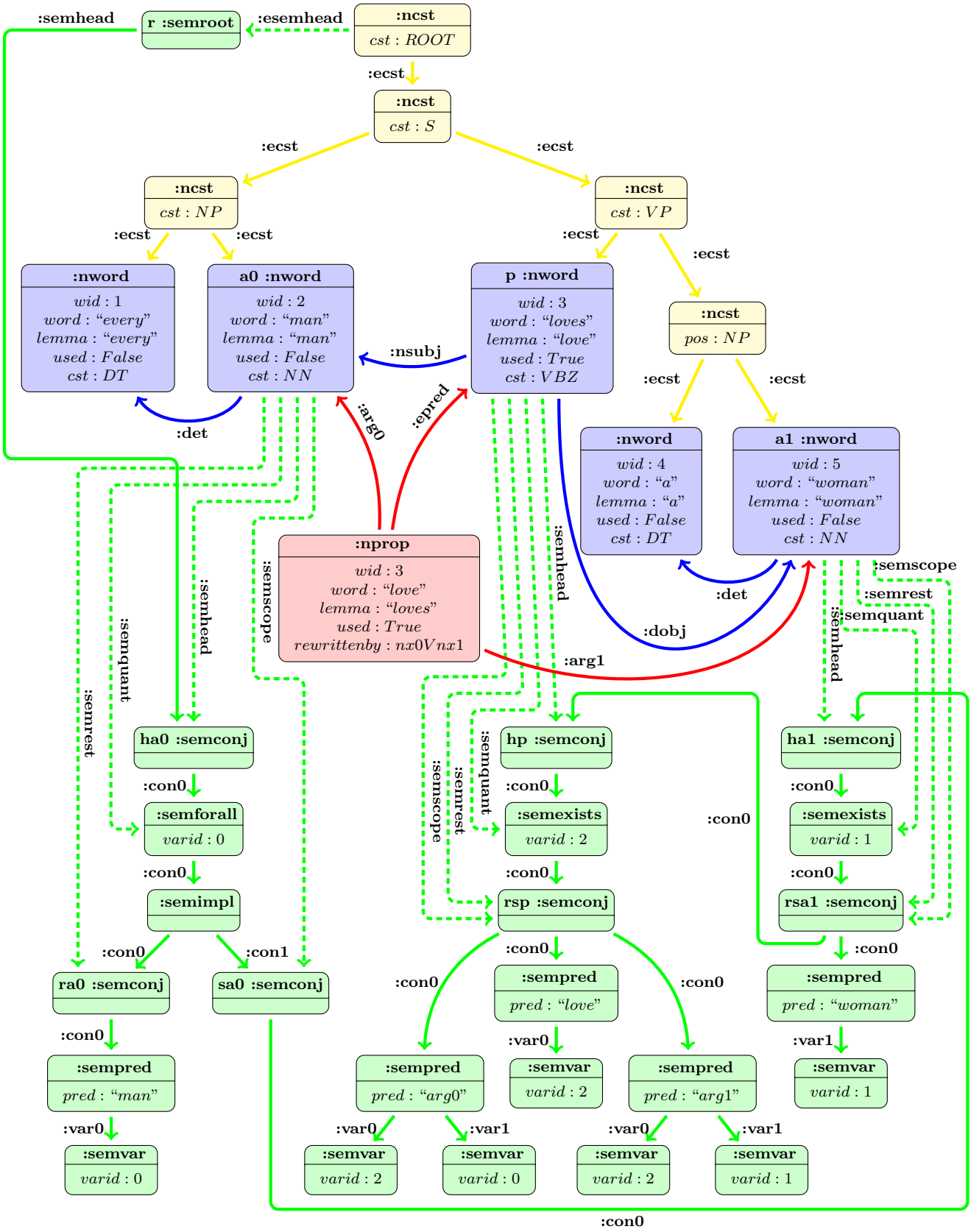


FIGURE 4.8 – Représentation complète de la phrase « Every man loves a woman »

À partir de cette représentation nous pouvons dériver, pour la phrase « Every man loves a woman », la formule logique :

$$\forall V0.(man(V0) \wedge \exists V1.(womam(V1) \wedge \exists V2.(love(V2) \wedge arg0(V2, V0) \wedge arg1(V2, V1))))$$

Nous générons pour le moment qu'une seule lecture pour chaque analyse, mais nous envisageons d'ajouter des règles de réécriture pour inverser les quantificateurs quand cela est possible afin d'obtenir toutes les lectures possibles. Il est important de prendre en compte toutes les lectures possibles, car il y a implication textuelle entre deux textes si une des lectures de l'un implique une des lectures de l'autre [21]. Mais le problème est que la complexité du calcul est nettement augmentée. Étant donné n le nombre d'analyses que l'on prend pour chaque phrase, on a en tout n^2 tests d'implications à faire pour savoir si une des analyses du premier texte implique une de celles du second. Étant donné q_1 et q_2 les nombres de quantificateurs des deux phrases, on obtient $n^2 * 2^{((q_1-1)+(q_2-1))}$ implications à tester dans le pire des cas, ce qui augmente largement le temps de calcul et les chances de tomber sur une formule indécidable.

4.4.2 Connaissances lexicales

Nous avons intégré à notre système la base de connaissances lexicales WordNet [Fellbaum, 1998], où chaque entrée représente un ensemble de synonymes. Un lemme polysémique sera présent dans un ensemble de synonymes par sens qui peut lui être associé. Ces ensembles de synonymes sont ensuite reliés via un grand nombre de relations lexicales binaires comme l'antonymie, l'hyponymie, ou encore la méronymie. Plutôt que d'intégrer toutes les connaissances pour chaque test d'implication, nous avons préféré les intégrer sur demande afin de simplifier au maximum les formules qui sont données aux outils de preuve. Étant donné une couple de textes T-H, nous extrayons toutes les connexions existantes entre un mot de T et un mot de H puis les transformons en axiomes.

Pour ce qui est des transformations des connaissances lexicales en axiomes de logique du premier ordre nous avons repris les transformations définies dans [Bos & Markert, 2005] :

relation lexicale	axiome logique
A est un synonyme de B	$\forall x.A(x) \Leftrightarrow B(x)$
A est un antonyme de B	$\forall x.A(x) \Leftrightarrow \neg B(x)$
A est un hyponyme de B	$\forall x.A(x) \Rightarrow \neg B(x)$
A et B partagent un hyperonyme	$\forall x.\neg(A(x) \wedge B(x))$

21. On pourrait aussi considérer qu'il y a implication textuelle si chaque lecture d'un des textes implique une des lectures de l'autre texte

Pour la sélection de connaissances lexicales, nous regardons pour chaque mot de T s'il existe une relation entre un des ensembles de synonymes qui lui sont associés et un ensembles de synonymes associés à H. Nous voulons cependant éviter de récupérer pour un même mot des relations liés à plusieurs de ses sens, car cela nous permettrait de dire que des implications comme (53) sont vraies. C'est pourquoi si il existe plusieurs relations associées à un mot de T ou de H, mais concernant différents ensembles de synonymes de ce mot (et donc un sens différent du mot), nous testons l'implication entre T et H successivement avec les relations associées à un des ensembles de synonymes puis à l'autre. Cela permet ainsi de reconnaitre l'implication (53) comme fausse et les implications (54) et (55) comme vraies.

(53) **T :** « Bill likes records »
H : « Bill likes music and performance »
Implication : NON

(54) **T :** « Bill likes records »
H : « Bill likes music »
Implication : OUI

(55) **T :** « Bill likes records »
H : « Bill likes performance »
Implication : OUI

(56) **T :** « A cat eats a mouse »
H : « An animal eats an animal »
Implication : OUI

Pour l'implication (56) nous récupérons donc les liens entre les mots de T (i.e., {« cat », « eat », « mouse »}) et les mots de H (i.e. : {« animal », « eat »}). Nous avons donc que « cat » et « mouse » sont des hyponymes de « animal ». Nous allons donc créer les deux axiomes suivants :

$$\forall X.cat(X) \Rightarrow animal(X)$$

$$\forall X.mouse(X) \Rightarrow animal(X)$$

La relation de partage d'hyperonyme entre « cat » et « mouse » pourrait créer l'axiome ci-dessous mais comme ces deux mots appartiennent à la même phrase on ne l'ajoute pas. Cette information est juste nécessaire pour vérifier que la phrase T est correcte

$$\forall X.\neg(cat(X) \wedge dog(x))$$

(57) **T :** « The dog is a cat »
H : « The dog is an animal »
Implication : OUI

Un autre avantage de la sélection de connaissances est que cela permet de tester des implications sur des phrases pouvant être incohérentes avec les connaissances qu'un être humain a du monde. Par exemple, la phrase T de (57) est incohérente avec le fait que « cat » et « dog » partagent un hyperonyme et qu'une entité ne peut pas être la fois un chat et un chien. Cependant, même si cette phrase est incohérente avec la connaissance du monde, un être humain peut quand même détecter l'implication (57) en permettant une même entité d'être à la fois un chat et chien et un utilisant le relation d'hyponymie entre « cat » et « animal ».

4.4.3 Réécriture, Lambda-calcul, Sémantique compositionnelle

Nous avons fait le choix de la réécriture comme modèle de calcul pour la construction sémantique pour avoir un système uniforme et voir si ce modèle de calcul apporte quelque chose par rapport au lambda-calcul.

Nous allons tout d'abord comparer d'un point de vue théorique les deux systèmes. L'un des grands avantages du lambda-calcul typé est que sa confluence et sa terminaison ont été démontrées (⁅Girard *et al.*, 1989⁆). Cela signifie qu'étant donné un lambda-terme, il existe une unique forme normale (un lambda-terme qui ne peut plus être réduit) que l'on peut obtenir à partir du lambda-terme via un nombre fini de réduction, peu importe l'ordre dans lequel on applique les réductions. On est ainsi sûr que le système terminera toujours, et donnera toujours la même réponse. Pour ce qui est de la réécriture, la confluence et la terminaison ne peuvent pas être prouvées car il existe des systèmes non confluents et qui ne terminent pas. Cependant, la confluence peut facilement être imposée grâce à des stratégies de réécriture, et il est facile de prouver la terminaison d'un système de réécriture donné, en regardant les ressources consommées et créées. Dans notre cas, les règles sont faites de façon à ne pas filtrer les motifs contenant un des éléments qu'elles ajoutent les empêchant ainsi d'être appliquées plusieurs fois au même endroit. De plus, chaque système de réécriture filtre les données d'un niveau pour ajouter celles du niveau suivant. Comme il n'y a pas d'ajout de données au niveau filtré et que les règles ne peuvent pas s'appliquer deux fois au même endroit, nos systèmes de réécriture terminent forcément. Théoriquement le lambda-calcul typé est donc supérieur à la réécriture, car il conflue et termine toujours. Cependant, la non confluence de la réécriture peut être vue comme un plus, car l'utilisation des stratégies permet d'obtenir des résultats différents (pouvant représenter les différentes lectures d'un texte) à partir d'un même ensemble de règles.

La théorie n'est pas l'unique élément important pour avoir un système élégant. Il est aussi nécessaire que le modèle de calcul permette de définir un système de façon concise et facilement compréhensible. Il n'est pas possible de comparer la lisibilité du lambda-calcul à celle de la réécriture, mais nous pouvons comparer la taille et

la complexité de systèmes équivalents dans ces deux formalismes. Le système de réécriture que nous avons développé permet de gérer la création de la sémantique à partir de structures syntaxiques étiquetées. Grâce à cela nous pouvons gérer un grand nombre de variations syntaxiques avec uniquement 10 règles, alors que la première version de Nutcracker [Bos *et al.*, 2004] était composée de 245 lambda-termes. Notons que plus on a de lambda-termes ou de règles de réécriture, plus le nombre d'interactions possible augmente rapidement. Il devient donc bien plus difficile de vérifier que tout fonctionne comme on le souhaite avec 245 lambda-termes qu'avec 10 règles de réécriture. Enfin, la réécriture permet de gérer facilement des phénomènes non locaux, ainsi que de modifier des structures précédemment créées. Il est ainsi possible d'intégrer facilement une négation au sein de la formule, de faire remonter un quantificateur d'un niveau, ou de relier une anaphore à son antécédent.

4.5 Conclusion

Nous avons donc vu dans ce chapitre la description de notre système de détection d'implications textuelles basé sur la réécriture. Nous avons montré que la réécriture pouvait être utilisée à la fois pour récupérer des motifs de structure prédicative afin de les étiqueter, mais qu'elle pouvait aussi être utilisée pour obtenir la sémantique associée à un texte. Nous avons aussi montré que l'utilisation d'un algorithme de fusion, permet d'augmenter facilement la couverture d'une ressource. Pour ce qui est de l'ajout des règles de réécriture pour la normalisation, nous nous somme aperçu qu'un des problèmes de conversion de ressources qui pouvait exister est d'avoir des ressources existant uniquement sous forme factorisée et nécessitant une lecture approfondie du rapport technique afin de pouvoir la développer. Les ressources existant dans une forme factorisée devrait donc toujours exister aussi dans une forme développée afin de pouvoir être utilisées sans devoir comprendre les mécanismes de factorisation. Pour ce qui est de la génération de formules, nous avons proposé une première approche de création de formule logique qui est facilement extensible par l'ajout de règles de réécriture. Enfin nous avons montré comment nous intégrons des ressources lexicales à notre système de manière à ne pas surcharger les outils de preuves et de permettre la détection d'implication sur des textes incohérent avec la base de connaissances.

Chapitre 5

Évaluation et analyse d'erreurs

Dans ce chapitre nous commençons par un bref rappel des évaluations du RTE au cours des différentes campagnes. Ensuite, nous décrivons dans un premier temps la technique de fouille d'erreurs que nous souhaitons adapter à la DIT, et expliquons en quoi elle permet d'obtenir de meilleurs résultats que les calculs d'exactitude (section 5.1). Nous appliquons cette technique sur le corpus de la seconde campagne du RTE annotée avec ARTE, sur laquelle des évaluations ont précédemment été réalisées par Garoufi [2007] (section 5). Nous avons comparé nos résultats à ceux existant puis avons discuté des différences. Enfin, nous avons évalué notre système ainsi qu'un des systèmes ayant participé au challenge RTE sur les suites de tests que nous avons généré (section 5.3).

L'évaluation de systèmes sur une suite de tests n'est pas une tâche facile, car il est nécessaire de trouver une bonne métrique. Dans le cas du RTE, il est par exemple possible de calculer l'exactitude sur l'ensemble du corpus ou séparément sur les corpus d'implications et de non-implications. Le second choix permet tout d'abord de voir si les systèmes sont meilleurs pour détecter l'implication ou la non-implication, mais aussi de ne pas avoir de résultats biaisés si le corpus n'est pas équilibré. Un système évalué sur le RTE (qui a une distribution 50-50 pour les Oui-Non), ayant une exactitude de 70% sur le corpus complet peut avoir différentes répartitions sur les corpus Oui/Non. Quel est le meilleur type de système : celui qui a un couple d'exactitudes Oui/Non de 90%-50%, de 70%-70%, ou de 50%-90% ? Cela dépend surement du type de système dans lequel nous voulons intégrer le système de détection d'implications textuelles ainsi que l'échelle à laquelle il va être intégré. Il serait intéressant d'intégrer les systèmes de RTE dans des systèmes d'extraction d'information et de voir quel type de systèmes donne de meilleurs résultats. Nous pouvons nous dire que dans les cas réels il y aura beaucoup plus de non-implications que d'implications, et que les systèmes ayant une exactitude trop basse pour les non-implications risquent d'avoir un grand bruit sur ces résultats.

La proposition du RTE6 que nous avons vu dans la sous-section 1.1.1 est une

reformulation du problème de détection d'implications textuelles permettant une évaluation plus réelle des systèmes. Pour rappel, elle consiste à utiliser des couples avec plusieurs textes associés à une hypothèse et de demander au système d'identifier quels textes impliquent l'hypothèse. Si nous prenons une hypothèse associée à un ensemble de 10 textes et que seulement 2 textes impliquent l'hypothèse nous aurons alors que :

- un système répondant toujours Oui aura :
  précision : 20%, rappel : 100%, f-score : 33%
- un système répondant toujours Non aura :
  précision : 0%, rappel : 0%, f-score : 0%
- un système répondant aléatoirement Oui et Non (équiprobablement) aura :
  précision : 17%, rappel : 50%, f-score : 25%
- un système donnant une réponse bonne et une mauvaise aura :
  précision : 50%, rappel : 50%, f-score : 50%

Nous pouvons voir qu'une telle formulation du problème de reconnaissance d'implications textuelles pousse les systèmes à être précis sur les bonnes réponses tout en évitant qu'ils répondent faux à tout.

De plus, comme l'explique [Bayer *et al.*, 2005], si un système annote bien trois couples de plus qu'un autre, cela ne signifie pas forcément que ce système est meilleur que l'autre car cela dépend de la difficulté des couples bien annotés. [Bayer *et al.*, 2005] propose d'évaluer les systèmes au moyen de l'analyse de Rash [Bond & Fox, 2001] qui crée des interconnexions entre les performances des systèmes sur les couples et la difficulté de ceux-ci. Les résultats globaux des systèmes peuvent ainsi être plus facilement évalués, mais il n'est toujours pas possible de dire si un système est meilleur qu'un autre pour un type de phénomène. Garoufi [2007] a tenté de faire une analyse du RTE2 à partir de son annotation ARTE, mais nous allons voir dans la sous-section que son analyse est erronée à cause de la répartition des phénomènes et combinaisons de phénomènes dans ce corpus. Nous allons donc commencer par présenter la méthode d'évaluation des systèmes que nous utilisons sur un corpus annoté avec des informations sur les phénomènes mise en œuvre dans chaque couple comme notre suite de tests, ou celle du RTE2 augmentée avec ARTE. Nous testerons ensuite cette méthode sur RTE2+ARTE, et nous comparerons nos résultats à ceux de Garoufi [2007].

5.1 Fouille d'erreurs

Pour analyser les résultats des systèmes, nous avons choisi d'utiliser l'approche décrite dans [Sagot & Éric de La Clergerie, 2008], qui permet de faire de la fouille d'erreurs sur les résultats d'un analyseur syntaxique. Plus précisément l'approche

propose d'identifier les types de phénomènes syntaxiques que le système n'analyse pas bien. Pour cela, elle émet l'hypothèse qu'un seul phénomène est la cause d'un échec d'une analyse, et cherche ensuite quel phénomène à le plus de chance d'être le coupable de l'échec de cette analyse. Cette hypothèse, qui n'est pas nécessairement vérifiée, simplifie le modèle et donne des résultats pertinents. Nous allons voir l'algorithme de base dans un premier temps, puis nous décrirons l'extension de [de Kok *et al.*, 2009] permettant de regrouper des phénomènes s'ils ont plutôt tendance à provoquer l'échec quand ils sont ensemble que séparément.

5.1.1 Algorithme de base [22]

Pour appliquer cet algorithme, il faut que le corpus soit découpé en problèmes, contenant chacun un certains nombre de phénomènes. Nous notons p_i le i-ième problème, et $o_{i,j}$ $(1 \leq j \leq |p_i|)$ les occurrences des phénomènes apparaissant dans p_i. La fonction $F(o_{i,j})$ prend une occurrence de phénomène et retourne le phénomène correspondant, et la fonction *erreur* associe à un problème p_i la valeur 1 si le système donne la mauvaise réponse, et 0 sinon.

Soit $\mathcal{O}$ l'ensemble de toutes les occurrences du corpus, et $\mathcal{O}_f$ l'ensemble des occurrences d'un phénomène f dans le corpus :

$$\mathcal{O}_f = \{o_{i,j} | F(o_{i,j}) = f\}$$

Le nombre d'occurrences de f dans le corpus est noté $|\mathcal{O}_f|$, et le nombre d'occurrences de f dans des problèmes avec erreur est noté $|\mathcal{O}_f^{\text{err}}|$.

Le *taux de suspicion moyen global* $\overline{S}$ est la probabilité moyenne qu'une occurrence donnée d'un phénomène soit la cause d'un échec.

$$\overline{S} = \frac{\sum_i \text{erreur}(p_i)}{|\mathcal{O}|}$$

Soit un phénomène f qui est le j-ième phénomène du problème p_i, c'est-à-dire que $F(o_{i,j}) = f$. Supposons que l'analyse de p_i a échoué : $\text{erreur}(p_i) = 1$. Nous appelons taux de suspicion du j-ième phénomène $o_{i,j}$ du problème p_i la probabilité, notée $S_{i,j}$, que l'occurrence $o_{i,j}$ du phénomène f soit responsable de l'échec de p_i. Si au contraire le problème p_i a réussi, ses occurrences ont un taux de suspicion nul.

Nous définissons le taux de suspicion moyen S_f du phénomène f comme étant la moyenne des taux de suspicion de ses occurrences (tous problèmes confondus) :

$$S_f = \frac{1}{|\mathcal{O}_f|} \cdot \sum_{o_{i,j} \in \mathcal{O}_f} S_{i,j}$$

22. Description reprise en partie de l'article[Sagot & Éric de La Clergerie, 2008]

L'algorithme utilisé est un algorithme de recherche par point fixe, qui itère un certain nombre de fois les calculs suivants. Supposons que nous venons de terminer la n-ième itération : nous connaissons pour chaque problème p_i et pour chaque occurrence $o_{i,j}$ de ce problème l'estimation de son taux de suspicion $S_{i,j}$ par la n-ième itération, estimation notée $S_{i,j}^{(n)}$. Nous pouvons en déduire l'estimation de rang $n+1$ pour le taux de suspicion moyen de chaque phénomène f, noté $S_f^{(n+1)}$:

$$S_f^{(n+1)} = \frac{1}{|\mathcal{O}_f|} \cdot \sum_{o_{i,j} \in \mathcal{O}_f} S_{i,j}^{(n)}$$

Ce taux permet de calculer une nouvelle estimation des taux de suspicion des occurrences, en attribuant à chaque occurrence d'un problème p_i un taux de suspicion $S_{i,j}^{(n+1)}$ égale à l'estimation du taux de suspicion moyen $S_f^{(n+1)}$ de la forme correspondante, puis en normalisant à l'échelle du problème. Ainsi :

$$S_{i,j}^{(n+1)} = \text{erreur}(p_i) \cdot \frac{S_{F(o_{i,j})}^{(n+1)}}{\sum_{1 \leq j \leq |p_i|} S_{F(o_{i,j})}^{(n+1)}}$$

À ce stade, la $(n+1)$-ième itération est terminée, et nous pouvons recommencer à nouveau ces calculs, jusqu'à convergence sur un point fixe[23]. L'amorçage de cet algorithme se fait en posant, pour une occurrence $o_{i,j}$ dans le problème p_i, $S_{i,j}^{(0)} = \text{erreur}(p_i)/|p_i|$. Autrement dit, si l'analyse de p_i a échoué, nous partons d'une estimation où toutes ses occurrences ont une probabilité égale d'être la cause de l'échec.

Après un certain nombre d'itérations nous obtenons donc des estimations des taux de suspicion moyens de chaque forme, ce qui permet de repérer les phénomènes les plus vraisemblablement responsable d'erreurs et pour chaque phénomène f, identifier les problèmes p_i où une de ses occurrences $o_{i,j} \in \mathcal{O}_f$ est un des principaux suspects et où $o_{i,j}$ a un taux de suspicion parmi les plus élevés parmi les occurrences de f.

Il est possible d'utiliser un estimateur afin d'obtenir le taux de suspicion moyen lissé, noté $\widetilde{S}_f^{(n)}$, qui prend en compte le nombre d'occurrences de f. En effet, la confiance que l'on peut accorder à l'estimation $S_f^{(n)}$ est d'autant plus faible que le nombre d'occurrences $|\mathcal{O}_f|$ est faible. D'où l'idée de lisser $S_f^{(n)}$ en le remplaçant par un barycentre $\widetilde{S}_f^{(n)}$ de $S_f^{(n)}$ et de $\overline{S}$ dont le coefficient de pondération λ dépend de

23. Nous pouvons par exemple dire que le point fixe est atteint quand la moyenne des différences entre les taux de suspicion de deux itérations successives ne dépasse pas un certain seuil :

$$\left(\frac{1}{|\mathcal{O}|} \cdot \sum_{o_{i,j} \in \mathcal{O}} |S_{i,j}^{n+1} - S_{i,j}^{n}| \right) \leq \text{seuil}$$

.

$|\mathcal{O}_f|$: si $|\mathcal{O}_f|$ est grand, $\widetilde{S}_f^{(n)}$ sera proche de $S_f^{(n)}$; s'il est petit, il sera plus proche de $\overline{S}$:

$$\widetilde{S}_f^{(n)} = \lambda(|\mathcal{O}_f|) \cdot S_f^{(n)} + (1 - \lambda(|\mathcal{O}_f|)) \cdot \overline{S}$$

Les taux calculés sont des probabilités. Ainsi, $S_{i,j}^{(n)}$ peut s'interpréter comme la probabilité que l'échec de l'analyse de la phrase p_i soit causé par l'occurrence $o_{i,j}$. Pour ordonner les résultats nous avons différentes fonctions de mesure M_f :

- $M_f^b = S_f^n$ (mesure brute de l'erreur)
- $M_f^c = S_f^n * |\mathcal{O}_f|$ (mesure pondérée par le nombre d'occurrences de f dans le corpus)
- $M_f^{b,c} = S_f^n * \log(|\mathcal{O}_f|)$ (balance entre M_f^b et M_f^c)
- $M_f^{err} = S_f^n * \log(|\mathcal{O}_f^{\text{err}}|)$ (mesure pondérée par le nombre d'occurrences de f dans les couples causant problème permettant d'avoir l'espérance de gain si nous corrigeons le problème)
- $M_f^{b,err} = S_f^n * \log(|\mathcal{O}_f^{\text{err}}|)$ (balance entre M_f^b et M_f^{err})

5.1.2 Extension aux combinaisons de phénomènes

Le seul problème de cette approche est qu'il n'est pas possible de savoir si l'erreur est due à une combinaison de problèmes où à un seul. Heureusement de Kok *et al.* [2009] proposent une extension permettant de résoudre ce problème. Elle est proposée sous la forme d'un pré-traitement à réaliser sur les phénomènes, et permet de trouver si c'est une combinaison de phénomènes plutôt qu'un phénomène unique qui pose problème. L'idée de base est qu'étant donné deux phénomènes f_1 et f_2, soit l'un de ces phénomènes, soit la conjonction de ces phénomènes est la cause de l'échec. Dans le second cas, la conjonction $f_1 \wedge f_2$ aura alors un taux de suspicion supérieur à ceux de f_1 et f_2, et nous grouperons les formes f_1 et f_2 pour avoir $f_1 \wedge f_2$. La même méthodologie est appliquée pour avoir des conjonctions de plus de deux phénomènes.

De manière plus générale un groupements de phénomènes f_i et f_j est permis si $S_{(f_i \wedge f_j)} > S_{(f_i)}$ et $S_{(f_i \wedge f_j)} > S_{(f_j)}$. Comme la fréquence d'apparition d'une conjonction de phénomènes est inversement proportionnelle au nombre de phénomènes pris en compte, les grandes conjonctions sont souvent les plus suspectes. Pour pallier ce problème un facteur de groupement est pris en compte. Ce facteur dépend de la fréquence de la conjonction dans les problèmes qui provoquent une erreur. Nous obtenons donc les formules, avec $\alpha = 1.0$:

$$S_{(f_i \wedge f_j)} > S_{(f_i)} \cdot extFactor(f_i \wedge f_j)$$

$$S_{(f_i \wedge f_j)} > S_{(f_j)} \cdot extFactor(f_i \wedge f_j)$$

$$extFactor(f) = 1 + \exp^{-\alpha|\mathcal{O}_f^{\text{err}}|}$$

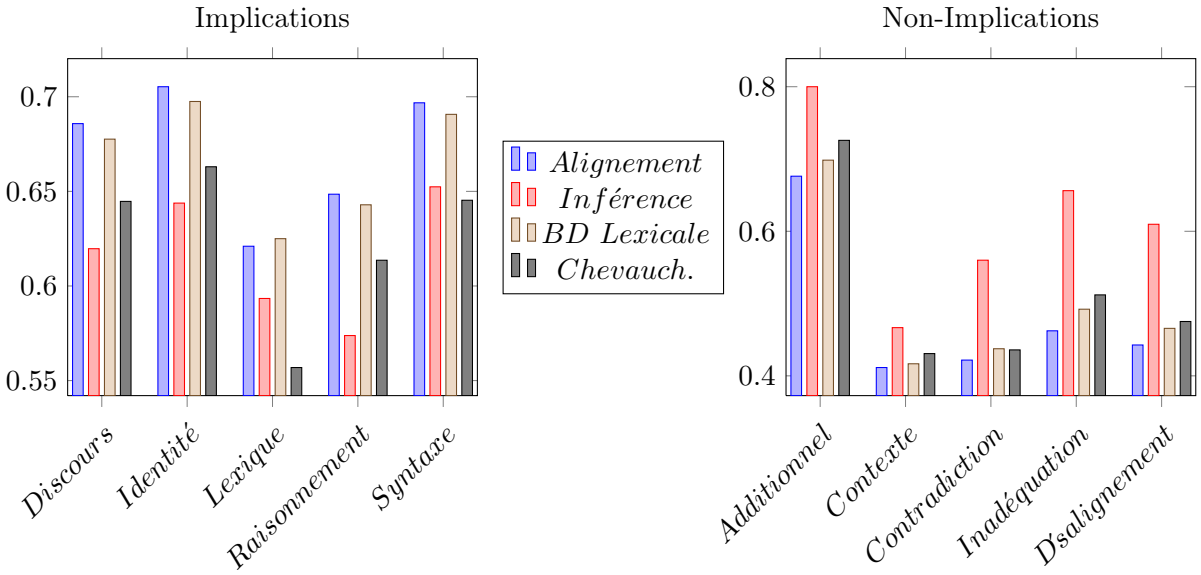


FIGURE 5.1 – Résultats de Garoufi : Graphiques représentants les moyennes d'exactitudes (en ordonnée) de chaque type de système (colonnes) pour chaque phénomène (en abscisse).

5.2 Fouille d'erreurs et comparaison des systèmes soumis à la campagne RTE2

Pour tester cet algorithme sur la tâche de reconnaissance d'implication textuelle, nous avons utilisé la suite de tests du RTE2 [Bar-Haim *et al.*, 2006] avec l'annotation ARTE [Garoufi, 2007]. En plus de son annotation Garoufi a annoté les systèmes avec des types représentant les caractéristiques qu'ils mettent en œuvre. Ensuite Garoufi a évalué les types de systèmes par rapports à leurs capacités à gérer les différents phénomènes. Il y a en tout 41 systèmes évalués avec 32 systèmes utilisant des *BD lexicales*, 13 systèmes utilisant un calcul de *chevauchement de mots*, 29 système utilisant des techniques d'*alignement* et enfin 5 systèmes utilisant des techniques d'*inférence*.

Garoufi obtient ses résultats en calculant pour chaque couple *système/phénomène* l'exactitude du système sur l'ensemble des couples étant annotés avec ce phénomène (les couples peuvent aussi être annotés avec d'autres phénomènes). Garoufi calcule ensuite pour chaque couple *type de système/phénomène* la moyenne des exactitudes des systèmes de ce type pour ce phénomène. Les résultats obtenus sont visibles sur l'histogramme FIG. 5.1. Pour le corpus d'implications, Garoufi conclut que les systèmes ayant un *moteur d'inférence logique* sont en général plus mauvais que les autres, et cela particulièrement pour les implications nécessitant du *raisonnement* ce qui est contraire à ce que l'on pourrait penser. À l'inverse, sur le corpus de non-

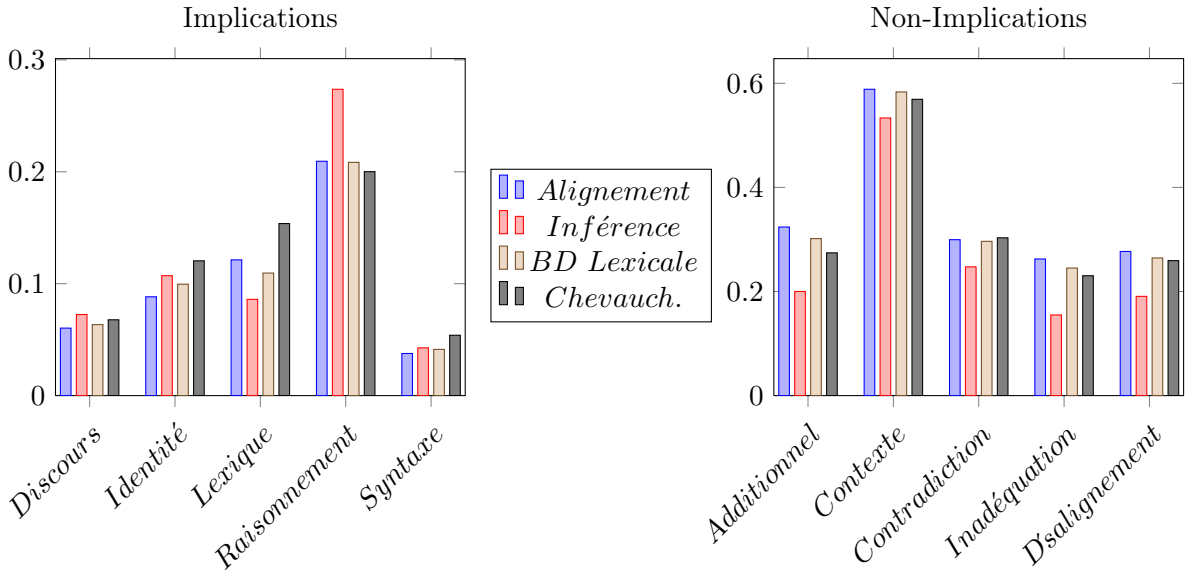


FIGURE 5.2 – Résultats de la fouille d'erreurs appliquée séparément sur chacun des systèmes. L'ordonnée représente la moyenne des suspicions pour chaque type de système.

implications Garoufi montre que ces systèmes sont meilleurs que les autres. Notons de plus que Garoufi infère aussi que les systèmes à base de *chevauchement de mots* ont plus de problèmes sur le *lexique* que sur le *raisonnement*.

Maintenant regardons ce que donne l'algorithme de fouille d'erreurs sur ces données. Pour chaque système nous calculons les taux de suspicion. Ensuite pour chaque couple *type de système/phénomène*, nous calculons la moyenne des taux de suspicion de ce phénomène sur chaque système de ce type. Le résultat obtenu est l'histogramme FIG. 5.2, où les barres les plus hautes sont celles qui posent le plus de problèmes, contrairement à l'exactitude utilisée par Garoufi. Nous pouvons voir que la fouille d'erreurs donne des informations contradictoires à celles données par les exactitudes. Par exemple, les systèmes à base de *chevauchement de mots* ont plus de problèmes à cause du *raisonnement* que du *lexique*.

Il est normal de se demander si la fouille d'erreurs donne vraiment de meilleures informations ou seulement des informations différentes. Les exactitudes et le nombre de couples pour chaque combinaison de phénomènes (c.f. 1.13) nous montrent pourquoi l'utilisation des exactitudes donne de mauvais résultats. Le problème vient du fait que la distribution des phénomènes et des combinaisons de phénomènes n'est pas du tout équilibrée. Pour mieux comprendre en quoi cela fait que les calculs d'exactitudes donnent de mauvaises informations, nous allons nous focaliser sur les exactitudes des couples annotés avec *lexique* et *raisonnement* pour les applications à base de *chevauchement de mots*. Le tableau ci-dessous donne les cardinalités et

les exactitudes pour différents ensembles de couples possibles avec les annotations *lexique* et *raisonnement*.

Phénomènes	**# Couples**	**Exactitude**
+ Lexique - Raisonnement	29	68,97%
+ Raisonnement - Lexique	213	65,62%
+ Lexique + Raisonnement	92	51,51%
+ Lexique	121	55,69%
+ Raisonnement	305	61,36%

Nous pouvons tout d'abord noter que les systèmes à base de *chevauchement de mots* ont une meilleure exactitude sur les couples avec *lexique* et sans *raisonnement* que sur ceux avec *raisonnement* et sans *lexique*. En utilisant uniquement ces données, nous pourrions conclure comme la fouille d'erreurs que les systèmes à base de *chevauchement de mots* sont meilleurs pour les couples avec *lexiques* que pour ceux avec *raisonnement*. Nous pouvons aussi observer que les couples avec *lexique* et sans *raisonnement* sont nettement moins présents que ceux avec du *raisonnement* et sans *lexique*. Quand nous les mélangeons aux autres couples avec du *lexique* et du *raisonnement* qui contient trois fois plus d'éléments, l'exactitude va fortement descendre. Alors que pour les couples avec *raisonnement* et sans *lexique* qui contient deux fois plus d'éléments que les couples avec du *lexique* et du *raisonnement* l'intégration diminuera l'exactitude moins fortement. Nous nous retrouvons donc à dire que les systèmes à base de *chevauchement de mots* sont plus performants sur le *raisonnement* que sur le *lexique* alors que c'est l'inverse. Il serait possible de calculer les exactitudes pour chaque ensemble de couples étant annoté avec une des combinaisons de phénomènes possibles, mais il serait difficile de conclure quelque chose à partir de ces résultats.

Le problème des étiquettes multiples qui existe sur un couple existe aussi sur les systèmes. En effet, comment savoir si le problème sur un couple vient plutôt d'une caractéristique du système plutôt qu'une autre. Pour résoudre ce problème nous proposons de passer tous les systèmes en une seule fois à la fouille d'erreurs en mettant comme étiquette pour chaque couple analysé par un système le produit cartésien des étiquettes de ARTE et des caractéristiques. Par exemple un système ayant comme caractéristiques *BD lexicale* et *chevauchement de mots* et un couple ayant comme phénomènes *lexique* et *raisonnement*, nous aurons l'ensemble de couples suivant : *(BD lexicale, lexique)*, *(BD lexicale, raisonnement)*, *(chevauchement de mots, lexique)* et *(chevauchement de mots, raisonnement)*. En appliquant tout cela nous obtenons les taux de suspicion de l'histogramme Fig. 5.3. Cet histogramme modifie grandement les conclusions faites précédemment. En effet, les systèmes à base d'*inférence* sont en fait les meilleurs partout (cela est surement dû au fait qu'il y a peu de systèmes

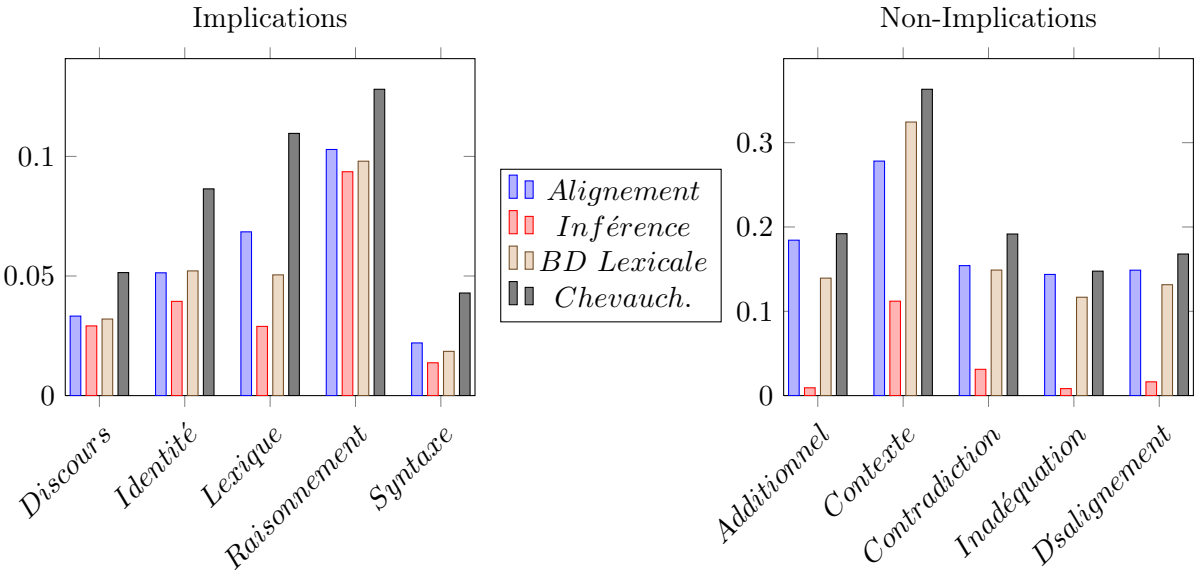


FIGURE 5.3 – Résultats de la fouille d'erreurs appliqué à l'ensemble des couples (type de système, phénomène). L'ordonnée représente la suspicions pour chaque couple (type de système, phénomène).

à base d'*inférence* et que le meilleur système fait partie de ceux-ci), et qu'ils ne sont du coup plus les pires en *raisonnement* (ce qui parait logique). Le *chevauchement de mots* quant à lui est devenu le pire partout. Dans les non-implications, il y a surtout une augmentation des différences.

5.3 Application à Afazio et Nutcracker sur les données construites

Dans la suite de tests que nous avons développé (c.f., chapitre 3), nous avons associé aux problèmes des annotations sur les variations syntaxiques, les quantificateurs, les implicatifs, et les relations lexicales. Nous souhaitons utiliser la méthode de fouille d'erreurs sur cette suite de tests annotée afin de connaitre les limites de notre système. Le problème est que la méthode de fouille d'erreurs utilise des annotations associées aux problèmes de détection d'implications textuelles et que les annotations produites par le réalisateur de surface utilisé sont associées aux phrases et non aux problèmes. Nous allons donc voir dans un premier temps comment adapter les annotations obtenues précédemment à la fouille d'erreurs. Ensuite, nous appliquerons la fouille d'erreurs sur les résultats produits par Afazio et Nutcracker [24] sur la suite de tests ré-annotée.

24. Nutcracker est le système de détection d'implications textuelles qui ressemble le plus à notre système et que nous avons présenté dans la section 2.5

5.3.1 Adaptation des annotations

Pour que la fouille d'erreurs fonctionne, il est nécessaire que les annotations modélisent précisément les phénomènes mis en œuvre dans les problèmes. Pour rappel, un problème est annoté de la manière suivante :

(58) **T :** « A man did not refused to dance with every woman »
ANNOTATIONS SYNTAXIQUES :
send :{(`n0Vn2n1`, `active`, `CanSubject`),
(`n0Vn2n1`, `active`, `CanObject`)}
ANNOTATIONS QUANTIFICATEURS :
{`a:scope:every`}
ANNOTATIONS IMPLICATIFS :
{`top:both-inv`, `both-inv:pos-inv`, `pos-inv:bottom`}
H : « A man danced with every lady »
ANNOTATIONS SYNTAXIQUES :
send :{(`n0Vn2n1`, `active`, `CanSubject`),
(`n0Vn2n1`, `active`, `CanObject`)}
ANNOTATIONS QUANTIFICATEURS :
{`a:scope:every`}
ANNOTATIONS IMPLICATIFS :
{`top::bottom`}
Implication : OUI
ANNOTATIONS LEXICALES : {`hyperonyme`}

Les annotations faites sur les relations lexicales mettent bien en avant un phénomène existant entre le texte et l'hypothèse du problème et pouvant être la cause de la mauvaise reconnaissance d'implications textuelles. Cette annotation peut donc être gardée telle quelle.

Les autres annotations donnent uniquement des informations sur les phénomènes apparaissant dans chacune des phrases. Pour ce qui est des variations syntaxiques, ce type d'annotations permet uniquement de trouver les variations syntaxiques pour lesquelles l'analyseur syntaxique donne de mauvaises analyses provoquant ainsi une reconnaissance incorrecte de l'implication.

Cependant, nous souhaitons aussi pouvoir détecter des dysfonctionnements liés à un changement de variation syntaxique, comme le fait d'avoir un argument d'un prédicat réalisé en tant qu'objet d'un verbe transitif dans le texte, et réalisé en tant que sujet d'un verbe ergatif dans l'hypothèse. Il est important de noter que le sens du changement est important, car le système peut bien fonctionner dans un sens mais pas dans l'autre. Par exemple, si la sémantique que l'on associe à une variation est plus générale que ce qu'elle devrait être, elle n'impliquera pas les variations équivalentes, mais les variations équivalentes l'impliqueront. Afin de gérer ce type de phénomènes,

nous réalisons le produit cartésien des annotations syntaxiques associées à chacune des phrases et obtenons ainsi la liste des changements qui auraient pu affecter le jugement du système. Pour cela nous prenons tous les prédicats apparaissant dans l'une des phrases et réalisons le produit cartésien des annotations associées à ce prédicat dans chacune des phrases. Si un prédicat n'est pas présent dans une des phrases, on lui associe l'ensemble d'annotations {`*Vide*`}. Nous obtenons ainsi pour l'exemple (58) l'ensemble d'annotations suivant :

ANNOTATIONS SYNTAXIQUES :

{`n0Vn2n1:active:canSubj ▷ n0Vn1Pn2:shortPassive:canSubj`,
`n0Vn2n1:active:canSubj ▷ n0Vn1Pn2:shortPassive:canSubj`,
`n0Vn2n1:active:canSubj ▷ n0Vn1Pn2:shortPassive:canSubj`,
`n0Vn2n1:active:canSubj ▷ n0Vn1Pn2:shortPassive:canSubj`}

Nous appliquons la même méthode de propagation des annotations au niveau de la phrase pour les quantificateurs et les implicatifs. Nous obtenons ainsi pour l'exemple (58) les ensembles d'annotations suivants :

ANNOTATIONS QUANTIFICATEURS :

{`a:scope:every ▷ top:scope:every`}

ANNOTATIONS IMPLICATIFS :

{`top:both-inv ▷ top:bottom`,
{`both-inv:pos-inv ▷ top:bottom`,
{`pos-inv:bottom ▷ top:bottom`}

Pour la fouille d'erreurs, nous utilisons à la fois les annotations associées aux phrases et aux problèmes, afin de pouvoir identifier à la fois les phénomènes et les variations de phénomènes qui trompent le système.

5.3.2 Evaluation

Outre le choix des annotations utilisées pour la fouille d'erreurs, se pose la question de savoir sur quelle partie du corpus appliquer l'algorithme. Nous pourrions prendre le corpus entier, mais il parait logique de penser que les causes d'erreurs d'une implication détectée comme une non-implication sont différentes des causes d'erreurs d'une non-implication détectée comme une implication. Dans le premier cas c'est souvent que le système est trop restrictif et dans le second qu'il est trop permissif. Le seconde difficulté est de savoir avec quel ensemble de problèmes bien reconnus nous devons comparer ces erreurs. Est-il plus juste de comparer les implications détectées comme des non-implications aux implications bien détectées ou aux non-implications bien détectées. Nous avons pris le premier choix car quand une implication est détectée comme non-implication cela est dû à un certain phénomène mal géré par le système, et que les non-implications bien détectées contenant ce

phénomène peuvent l'être à cause de la mauvaise gestion de celui-ci, alors que les implications bien détectée le seront car la gestion de ce phénomène a bien fonctionné. Nous analysons donc séparément le corpus d'implications et de non-implications.

Pour l'évaluation d'Afazio et de Nutcracker, nous avons défini deux suites de tests différentes pour évaluer séparément la capacité du système à gérer les variations syntaxiques et les quantificateurs.

Évaluation de la gestion des variations syntaxiques

Pour la première suite de tests nous avons créé manuellement 81 formules sémantiques pouvant être réalisés par 7 familles de verbes `n0V` (intransitif), `n0Vn1` (transitif), `n0Vn2n1` (ditransitif), `n0Va1` (prédicat adjectival), `n0Vs1` intransitif à complément phrastique, `betaVv` (auxiliaires) et `n0Vn1Pn2` (ditransitif avec preposition). À partir de ces formules nous avons dérivé automatiquement 226 phrases permettant de générer 6386 problèmes avec un ratio de 42,6% d'implications et de 57,4% de non-implications. 44464 problèmes ont été filtrés car représentant des non-implications triviales. Nous avons ensuite appliqué notre algorithme de sélection de suite de tests afin d'obtenir un échantillon de 1000 problèmes. La première contrainte de distribution impose d'avoir autant d'implications que de non-implications. Les autres contraintes imposent certaines familles de verbes à avoir un tiers de problèmes avec un verbe de la famille et de deux tiers de problèmes sans. Nous avons contraint les familles `n0V`, `n0Vn1`, `n0Vn2n1`, `n0Va1` car ce sont des familles représentant des variations syntaxiques principales. La distribution finale des problèmes pour satisfaire nos contraintes est visible table 5.1. Les cases cochées permettent de savoir si l'implication a lieu ou si un verbe de la famille est présent. Les contraintes globales sont bien respectées mais les contraintes locales ne le sont que partiellement – on voit en particulier, qu'il n'y a pas de problème impliquant plus de deux des familles principales. Les problèmes manquant à certaines combinaisons de contraintes ont donc été répartis dans les autres combinaisons afin de respecter les contraintes globales.

Nous avons testé Afazio et Nutcracker sur cette suite de tests. Nutcracker utilise un module permettant de donner une réponse basée sur le chevauchement de mots si les outils de preuve mettent trop de temps. Comme notre suite de tests est composée de problèmes avec un chevauchement de mots élevé, ce module tend à prédire une implication textuelle presque tout le temps. Nous avons donc décidé de considérer les résultats de Nutcracker tels quels mais aussi la version Nutcracker* où nous considérons comme faux les cas où Nutcracker dit que l'implication est vraie à cause du module de chevauchement de mots. Il arrive parfois que l'analyseur syntaxique utilisé par Nutcracker n'arrive pas à trouver d'analyse pour une phrase (e.g., « the woman gives the flower to the man that sings »), ce qui provoque des erreurs lors du test d'implication.

Implication	`n0V`	`n0Vn1`	`n0Vn2n1`	`n0Va1`	TOTAL	BEST	SELECT
					18	100	18
				✓	54	49	44
			✓		102	49	101
			✓	✓	60	25	60
		✓			616	49	616
		✓		✓	1830	25	43
		✓	✓		51	25	31
		✓	✓	✓	0	12	0
	✓				257	49	107
	✓			✓	584	25	40
	✓		✓		64	25	2
	✓		✓	✓	0	12	0
	✓	✓			0	25	0
	✓	✓		✓	0	12	0
	✓	✓	✓		0	12	0
	✓	✓	✓	✓	0	6	0
✓					8	100	8
✓				✓	64	49	21
✓			✓		117	49	59
✓			✓	✓	88	25	47
✓		✓			361	49	90
✓		✓		✓	960	25	53
✓		✓	✓		134	25	39
✓		✓	✓	✓	0	12	0
✓	✓				204	49	126
✓	✓			✓	496	25	43
✓	✓		✓		272	25	12
✓	✓		✓	✓	0	12	0
✓	✓	✓			46	25	2
✓	✓	✓		✓	0	12	0
✓	✓	✓	✓		0	12	0
✓	✓	✓	✓	✓	0	6	0

TABLE 5.1 – Distribution de la suite de tests syntaxiques (TOTAL représente le nombre total de problèmes dans l'ensemble de base, BEST est le nombre théorique de problèmes qu'il faudrait sélectionner et SELECT est le nombre de problèmes sélectionnés)

Système	Erreur	VN	FN	VP	FP	VN/N	VP/P	Exactitude
Afazio	0	322	160	340	178	64,4%	68,0%	66,2%
Nutcracker	27	9	31	446	464	1,8%	89,2%	46,8%
Nutcracker*	27	454	338	139	19	90,8%	27,8%	60,9%

Table 5.2 – Résultats des systèmes sur la suite de tests syntaxiques (VN = vrais négatifs, FN = faux négatifs, VP = vrais positifs, FP, faux positifs, N = VN + FP, P = VP + FN)

La table 5.2 donne les résultats des trois systèmes considérés. Nous pouvons voir qu'Afazio se trompe a peu près autant sur les implications que sur les non-implications. La version de Nutcracker avec le module de chevauchement de mots détecte l'implication comme vraie dans plus de 90% des cas et obtient donc un bonne exactitude pour les implications, et une mauvaise exactitude pour les non-implications. La seconde version de Nutcracker détecte l'implication comme fausse dans un peu moins de 80% des cas et obtient donc des exactitudes inverses à celles de la première version. Au final, notre système obtient une meilleure exactitude que les deux versions de Nutcracker d'un peu plus de 5 points.

Les tables 5.3 et 5.4 montrent respectivement la fouille d'erreurs faite sur les corpus d'implications et de non-implications pour Afazio. Nous avons utilisé la fouille d'erreurs en utilisant la fonction de score $M_f^{b,err}$ et l'extension de regroupement des phénomènes. Le premier tableau montre les phénomènes qui empêchent Afazio de bien détecter les implications. Le phénomène qui pose le plus de problèmes avec un fort taux de suspicion, est le passage d'une nominalisation à une forme verbale du prédicat (1, 2 et 3). Le phénomène inverse (passage de la forme verbale à la nominalisation) pose aussi problème (4). Un autre phénomène qui pose problème est le passage d'une forme relative à une forme active pour les verbes bitransitifs avec préposition (5). Le second tableau montre que ce qui pousse Afazio à détecter une non-implication comme implication sont principalement des phénomènes apparaissant dans la partie gauche de l'implication. Une analyse de surface de ces résultats ne nous a pas permis de comprendre pourquoi ces phénomènes posaient problème. Après une étude plus approfondie des résultats, nous nous sommes finalement rendus compte que ces phénomènes apparaissaient la plupart du temps dans des problèmes où l'hypothèse était très courte (moins de quatre mots). Le problème des phrases très courtes est que sur les 9 analyses que nous récupérons, il y a des analyses complètement fausses qui donnent des sémantiques partielles pouvant être impliquées beaucoup plus facilement. Une solution pour résoudre ce problème serait de choisir le nombre d'analyses en fonction de la taille du texte, ou de filtrer les analyses.

Pour Nutcracker, nous avons réalisé la fouille d'erreurs, uniquement sur la version sans le module de chevauchement de mots car elle donne de meilleurs résultats. La

#	Phénomène			S_f^n	$M_f^{b,err}$
1	n0V :betaVn :	▷	n0V :active :RelativeCovertSubject	0,89	1,97
2	n0Vn1 :active :NPGerundSubjectCan	▷	n0Vn1 :active :CanSubject	0,55	1,26
3	n0V :betaVn :	▷	n0V :active :CanSubject	0,29	0,83
4	n0Vn1 :active :CanSubject	▷	n0Vn1 :active :NPGerundSubjectCan	0,32	0,74
5	n0Vn1Pn2 :passive :RelativeOvertSubject	▷	n0Vn1Pn2 :active :CanObject	0,35	0,73

TABLE 5.3 – Fouille d'erreurs sur les implications de la suite de tests syntaxiques pour Afazio

#	Phénomène			S_f^n	$M_f^{b,err}$
1	n0Vn1 :active :NPGerundSubjectPro	▷	: :	0,48	0,67
2	n0V :active :RelativeOvertSubject	▷	: :	0,43	0,48
3	n0Vn1 :passive :CanByAgent	▷	n0Vn1 :passive :CanSubject	0,26	0,47
4	n0V :betaVn :	▷	: :	0,63	0,44
5	n0Vs1 :active :CanSubject	▷	: :	0,14	0,38

TABLE 5.4 – Fouille d'erreurs sur les non-implications de la suite de tests syntaxiques pour Afazio

table 5.5 montre que les phénomènes qui empêchent Nutcracker* de détecter des implications sont les mêmes que pour Afazio. Nous pouvons cependant remarquer que les suspicions non rééquilibrées (S_f^n) sont inférieures à celles d'Afazio, alors que Nutcracker* reconnaît moins bien les implications qu'Afazio (exactitude de 27,8% pour Nutcracker* contre 68,0% pour Afazio). Cela est dû au fait que les suspicions sont mieux réparties sur les différents phénomènes pour Nutcracker*. Donc, même si les phénomènes qui posent le plus de problèmes sont les mêmes, ils se détachent plus du reste des phénomènes pour Afazio que pour Nutcracker*. Pour ce qui est des phénomènes qui poussent Nutcracker* à croire qu'une non-implication est vraie, la table 5.6 montre que l'utilisation de subordonnées relatives est la source principale d'erreurs.

#	Phénomène			S_f^n	$M_f^{b,err}$
1	n0V :betaVn :	▷	n0V :active :RelativeOvertSubject	0,45	0,73
2	n0V :active :CanSubject n0V :betaVn :			0,16	0,64
3	n0V :active :RelativeOvertSubject	▷	n0V :betaVn :	0,31	0,60
4	n0V :active :RelativeOvertSubject	▷	n0V :active :CanSubject	0,18	0,54
5	n0V :active :CanSubject	▷	: :	0,24	0,50

TABLE 5.5 – Fouille d'erreurs sur les implications de la suite de tests syntaxiques pour Nutcracker*

#	Phénomène	S_f^n	$M_f^{b,err}$
1	n0Vn1Pn2 :passive :CanByAgent ▷ n0Vn1Pn2 :active :RelativeOvertObject	0,18	0,37
2	n0V :active :RelativeCovertSubject ▷ n0V :active :RelativeCovertSubject	0,27	0,37
3	n0Vn1Pn2 :passive :RelativeOvertSubject ▷ n0Vn1Pn2 :active :RelativeOvertObject	0,32	0,36
4	n0Vn1Pn2 :passive :CanByAgent ▷ n0Vn1Pn2 :active :RelativeCovertObject	0,12	0.26
5	n0V :active :RelativeCovertSubject	0,05	0,22

TABLE 5.6 – Fouille d'erreurs sur les non-implications de la suite de tests syntaxiques pour Nutcracker*

Évaluation de la gestion des quantificateurs

La seconde suite de tests a pour but d'évaluer les systèmes sur leur gestion des quantificateurs. Pour cela nous avons créé manuellement 35 formules sémantiques avec le plus de cas différents possibles. À partir de ces formules nous avons dérivé automatiquement 35 phrases permettant de générer 220 problèmes avec un ratio de 25% d'implications et de 75% de non-implications. 970 problèmes ont été filtrés car représentant des non-implications triviales.

Nous avons appliqué l'algorithme de sélection afin d'obtenir un échantillon de 60 problèmes. La première contrainte est la même que celle utilisée précédemment sur le ratio d'implications et de non-implications (50-50). Les autres contraintes forcent l'échantillon à avoir, pour chaque quantificateur, autant de problèmes avec et sans. La distribution finale que nous avons obtenue est visible table 5.7. Elle respecte globalement les contraintes posés.

Nous avons testé les deux systèmes sur cette nouvelle suite de tests et obtenu les résultats donnés dans la table 5.8. Afazio et Nutcracker* obtiennent les même exactitudes globales, mais ils ne font pas les même types d'erreurs. Afazio étiquette des non-implications comme étant des implications et Nutcracker* fait l'inverse. Comme Afazio répond parfaitement au corpus d'implications il ne sert à rien de faire la fouille d'erreurs sur ce corpus pour Afazio. Il en est de même pour Nutcracker* sur le corpus de non-implications.

Pour Afazio, la fouille d'erreurs sur le corpus d'implications donne la table 5.9. Ce tableau montre que les quantificateurs qui posent problème sont uniquement `no` et `notevery`. En regardant, les analyses et les représentations sémantiques des problèmes mal reconnus, nous nous sommes rendus compte que dans les 9 analyses associées aux phrases contenant un de ces quantificateurs, il y avait souvent une des analyses qui était fausse et qui générait une sémantique partielle de la phrase. Nous retombons donc sur le même problème que pour la syntaxe, et il est donc nécessaire de filtrer les analyses.

La table 5.10 montre les résultats de la fouille d'erreurs pour Nutcracker* sur le

Implication	a	every	not every	no	TOTAL	BEST	SELECT
					0	3	0
				✓	2	3	0
			✓		2	3	2
			✓	✓	8	3	8
		✓			2	3	2
		✓		✓	4	3	4
		✓	✓		10	3	10
		✓	✓	✓	0	3	0
	✓				2	3	0
	✓			✓	29	3	0
	✓		✓		5	3	0
	✓		✓	✓	10	3	2
	✓	✓			18	3	0
	✓	✓		✓	64	3	1
	✓	✓	✓		10	3	1
	✓	✓	✓	✓	0	3	0
✓					0	3	0
✓				✓	0	3	0
✓			✓		0	3	0
✓			✓	✓	0	3	0
✓		✓			0	3	0
✓		✓		✓	4	3	4
✓		✓	✓		0	3	0
✓		✓	✓	✓	0	3	0
✓	✓				0	3	0
✓	✓			✓	21	3	11
✓	✓		✓		7	3	7
✓	✓		✓	✓	0	3	0
✓	✓	✓			22	3	8
✓	✓	✓		✓	0	3	0
✓	✓	✓	✓		0	3	0
✓	✓	✓	✓	✓	0	3	0

TABLE 5.7 – Distribution de la suite de tests sur les quantificateurs (TOTAL représente le nombre total de problèmes dans l'ensemble de base, BEST est le nombre théorique de problèmes qu'il faudrait sélectionner et SELECT est le nombre de problèmes sélectionnés)

Système	Erreur	VN	FN	VP	FP	VN/N	VP/P	Exactitude
Afazio	0	22	0	30	8	73,3%	100,0%	86,7%
Nutcracker	0	7	0	30	23	23,3%	100,0%	61,7%
Nutcracker*	0	30	8	22	0	100,0%	73,3%	86,7%

Table 5.8 – Résultats des systèmes sur la suite de tests sur les quantificateurs (VN = vrais négatifs, FN = faux négatifs, VP = vrais positifs, FP, faux positifs, N = VN + FP, P = VP + FN)

#	Phénomène	S_f^n	$M_f^{b,err}$
1	top :scope :notevery ▷ top :scope :no	0,49	0,79
2	top :scope :notevery ▷ top :scope :notevery	1,00	0,69
3	top :scope :no ▷ top :scope :notevery	0,5	0,69
4	top :scope :no ▷ top :scope :notevery no	0,17	0,18
5	notevery	0,09	0,10

Table 5.9 – Fouille d'erreurs sur les non-implications de la suite de tests sur la quantification pour Afazio

corpus d'implications. Seuls les quatre premiers résultats ont été listés car les autres ont tous un taux de suspicion de 0,00. La table montre 4 phénomènes avec un score identique et impliquant tous les quantificateurs. Cela signifie que l'erreur n'est pas due à un quantificateur en particulier mais plutôt à un autre phénomène.

5.4 Conclusion

Dans ce chapitre nous avons proposé un nouveau moyen d'évaluer les systèmes de reconnaissance d'implications textuelles permettant de comparer leurs performances sur les différents phénomènes nécessaires à la détection d'implications textuelles. Nous avons tout d'abord décrit la méthode de fouille d'erreurs de Sagot & Éric de La Clergerie [2008] servant à l'origine à trouver pour un analyseur syntaxique les

#	Phénomène	S_f^n	$M_f^{b,err}$
1	top :scope :no ▷ top :scope :every	1,00	0,69
2	top :scope :a ▷ top :scope :notevery	1,00	0,69
3	top :scope :notevery ▷ top :scope :a	1,00	0,69
4	top :scope :every ▷ top :scope :no	1,00	0,69

Table 5.10 – Fouille d'erreurs sur les implications de la suite de tests sur la quantification pour Nutcracker*

groupes de mots qu'un analyseur n'arrive pas à analyser. Nous avons ensuite utilisé cette méthode pour refaire l'analyse que Garoufi [2007] avait fait sur sa version annotée du RTE2. Nous avons ensuite démontré que la fouille d'erreurs donne des résultats plus cohérents et plus utilisables que les calculs d'exactitude fait sur les sous-corpus correspondant à chaque phénomène. Nous avons ainsi contredit certaines des conclusions de l'analyse de Garoufi [2007] qui étaient fausses. Enfin, nous avons appliqué cette méthode aux résultats produits par Afazio et Nutcracker sur une suite de tests syntaxiques et sur une suite de tests sur la quantification. De cette façon, nous avons pu identifier les limites des deux systèmes et proposer des pistes d'amélioration.

Conclusion

Si le thème directeur de cette thèse est la reconnaissance d'implications textuelles, les contributions apportées vont de la génération automatique de suites de tests, au calcul sémantique en passant par l'étiquetage en rôles sémantiques et une approche critique de la tâche de détection d'implications textuelles. Pour chacun de ces points, nous proposons une nouvelle approche. Pour l'étiquetage en rôles sémantiques, nous présentons une approche symbolique (alors que les systèmes utilisent normalement des méthodes d'apprentissage automatique). Pour le calcul sémantique, nous proposons une approche basée sur la réécriture (alors que le lambda-calcul est normalement utilisé). Et pour l'évaluation de la tâche de reconnaissance d'implications textuelles, nous développons une approche basée sur la fouille d'erreurs plutôt que sur le calcul d'exactitude. Dans ce qui suit, nous passons brièvement en revue chacune de ces contributions puis nous évoquons les pistes de recherche naturellement évoquées par le travail réalisé.

Contributions

Critique du RTE Dans un premier temps, nous avons réalisé un examen critique des suites de tests existantes. Nous avons montré en particulier (Chapitres d'introduction et 5), que la campagne RTE manque d'annotations et ne propose pas suffisamment d'exemples impliquant des phénomènes logiques. Plus généralement, une critique qui peut être formulée est la suivante. Le but de la campagne RTE est d'évaluer les moteurs d'inférence des systèmes d'extraction d'information, de recherche d'information, de question-réponse et les synthétiseurs de documents, afin de pouvoir améliorer ces systèmes. Or, cette campagne utilise ces mêmes systèmes et les campagnes d'évaluation de ces systèmes pour la création de sa suite de tests. Ainsi, les différents problèmes de la suite de tests n'apportent pas de nouveau challenge par rapport aux évaluations existantes pour chacun de ces systèmes. Cette campagne permet donc uniquement d'améliorer les systèmes sur ce qu'ils savent faire et non sur de nouveaux phénomènes.

Les autres suites de tests (i.e. : FraCaS et AQUAINT) sont plus intéressantes pour le traitement automatique des langues car elles proposent soit des problèmes

mettant en œuvre des phénomènes précis et complexes comme les quantificateurs, la monotonicité et les comparatifs (voir la section 1.2.1 sur FraCaS), soit des problèmes pour lesquels il existe plusieurs réponses selon le contexte (voir la section 1.2.2 sur AQUAINT). La première suite de tests permet ainsi de se poser des questions fondamentales sur le fonctionnement du système de déduction humain, alors que la seconde propose des exemples mettant en évidence les différences entre l'implication logique et l'implication textuelle.

Système de SRL symbolique Notre première contribution est la proposition d'une approche symbolique d'étiquetage de rôles sémantiques. Cette contribution propose un ensemble de règles de réécriture permettant de filtrer les différentes variations syntaxiques en dépendances, puis d'étiqueter les structures filtrées à l'aide d'un ensemble de lexiques. L'avantage d'une telle méthode par rapport à une approche basée sur l'apprentissage automatique, est qu'elle permet d'identifier et de corriger la cause d'un mauvais étiquetage. Elle est aussi facilement adaptable à un nouveau domaine au sens où il suffit d'adapter le lexique au domaine considéré – les variations syntaxiques seront ensuite automatiquement gérées par les règles de réécriture. Les résultats obtenus, montrent que l'étiquetage obtenu est dans la moyenne des systèmes évalués lors du CoNLL 2009 et que la gestion des verbes est plus fiable que celle des noms. Cela est dû au fait que les variations syntaxiques nominales de l'anglais sont très permissives et qu'il y a beaucoup d'ambigüités dont la résolution requiert des connaissances sur le type sémantique des arguments.

Système de calcul sémantique Pour le calcul sémantique, nous avons proposé un système basé sur la réécriture associant une règle de réécriture par phénomène. La théorie obtenue est moins contrainte que la sémantique compositionnelle en lambda-calcul typé normalement adoptée pour le calcul sémantique, mais elle permet d'avoir un système plus concis qui est ainsi plus facile à comprendre, corriger et améliorer (cf. Chapitre 4). De plus, la réécriture permet de gérer des phénomènes non locaux sans avoir à propager artificiellement les informations requises pour le traitement de ces phénomènes. Nous avons défini un système basique permettant de gérer les quantificateurs, les propositions simples relatives et les compléments phrastiques. Comme l'ont montré les évaluations de notre système (Chapitre 5), s'ils restent limités en termes de couverture sémantique, les résultats obtenus sont satisfaisants.

Génération de suites de tests équilibrées Les travaux sur la génération de suites de tests équilibrées recouvrent deux contributions.

Une première contribution consiste à utiliser des outils de génération de textes pour construire une suite de tests pour les systèmes de détection d'implications textuelles. L'approche proposée se démarque des approches existantes de deux façons.

Premièrement, l'utilisation de générateurs de phrases permet la création de suites de tests conséquentes et annotées très finement (il serait très long pour un être humain de faire de telles annotations). Deuxièmement, l'approche permet de choisir le type d'implications que l'on souhaite inclure dans la suite de tests. Ainsi, il devient possible d'évaluer les systèmes sur leurs capacités à gérer les différents phénomènes et combinaisons de phénomènes important pour l'implication textuelle.

Une seconde contribution concerne le développement d'un algorithme de sélection qui permet de sélectionner un sous-ensemble d'une suite de tests afin de satisfaire au mieux un certain nombre de contraintes. De cette façon, il est possible d'éviter la sur- ou la sous-représentation d'un phénomène ou d'une combinaison de phénomènes dans les suites de tests créées par rapport aux autres permettant ainsi une évaluation moins biaisée des systèmes.

Nouvelle évaluation La dernière contribution est une nouvelle proposition d'évaluation des systèmes de détection d'implications textuelles, avec une méthode de fouille d'erreurs provenant du domaine de l'analyse syntaxique. Cette méthode permet d'identifier, étant donné une suite de tests annotée, les phénomènes faisant qu'un système se trompe. Dans un premier temps nous avons utilisé cette méthode sur la suite de tests du RTE2 annotée avec ARTE. Nous avons obtenu des résultats contradictoires et avons démontré que les résultats de Garoufi [2007] impliquaient de mauvaises conclusions dues au fait que la suite de tests RTE n'est pas bien équilibrée. Nous avons ensuite proposé d'utiliser la fouille d'erreurs sur tous les systèmes à la fois afin de pouvoir identifier précisément quels types d'approches permettent le mieux de résoudre quels types de phénomènes. Une telle analyse permet ainsi de mettre en place un système combinant ces approches pour obtenir le meilleur système possible. Pour finir, nous avons réalisé une évaluation d'Afazio et de Nutcracker sur deux suites que nous avons générées. Cela nous a permis de montrer qu'Afazio gère mieux les variations syntaxiques que Nutcracker, et que les deux systèmes gérent aussi bien la quantification. Cela nous a aussi permis de nous rendre compte que le module de normalisation nominale d'Afazio fonctionnait moins bien que le module de normalisation verbale, et qu'il fallait trouver un moyen de filtrer les mauvaises analyses.

Perspectives

Nous avons vu dans le chapitre 1 que l'implication textuelle est différente de l'implication logique au sens où elle considère comme vraie, une implication fortement probable. La suite de tests AQUAINT a mis en avant des exemples où l'implication est vraie ou fausse selon les connaissances dont on dispose. Afin de pouvoir traiter de

cette différence et à terme, étendre Afazio au traitement des implications textuelles non logiques, nous projetons de faire une étude plus poussée des cas pour lesquels l'implication textuelle et l'implication logique diffèrent et en particulier, d'analyser les problèmes de la campagne RTE qui n'ont pas été sélectionnés car les annotateurs étaient en désaccord.

L'algorithme permettant de sélectionner un sous-ensemble d'une suite de tests satisfaisant un certain nombre de contraintes pourrait être amélioré de façon à pouvoir trouver un meilleur compromis entre la taille du sous-ensemble sélectionné et le respect des contraintes auxquelles est soumis ce sous-ensemble. Pour ce faire, une piste possible serait d'intégrer un paramètre supplémentaire permettant d'équilibrer le rapport taille/respect des contraintes pour avoir soit une suite de tests de petite taille respectant bien les contraintes, soit une suite de tests de grande taille respectant moyennement les contraintes.

Le module de calcul sémantique a pour l'instant une couverture limitée (phrases simples avec ou sans argument phrastique, verbes de contrôle, subordonnées relatives). Nous projetons d'étendre la couverture de notre approche et en particulier, d'explorer l'apport du calcul sémantique basé sur la réécriture pour le traitement de phénomènes non locaux tels les implicatifs et les anaphores. Nous projetons également d'étendre notre approche afin de pouvoir mieux rendre compte des différentes lectures possibles des cas complexes impliquant différentes combinaisons de quantificateurs.

Enfin, un problème crucial reste la sélection de la bonne analyse syntaxique parmi les analyses proposées par l'analyseur de Stanford. Nous projetons d'étudier dans quelle mesure l'ordonnancement statistique intégré par cet analyseur peut être combiné avec des méthodes symboliques permettant d'améliorer la détection de la bonne analyse.

Glossaire

Afazio : notre système de détection d'implications textuelles.

AQUAINT : suite de tests composée de problèmes de DIT mettant en avant l'ambigüité de la tâche par rapport aux connaissances prises en compte.

ARTE : Annotating RTE.

CoNLL : Conference on Computational Natural Language Learning qui propose une campagne d'évaluation sur l'étiquetage de rôles sémantiques.

DIT : Détection d'Implications Textuelles.

exactitude : pourcentage de problèmes de DIT bien reconnus.

f-score : $2 \cdot \frac{\text{précision} \cdot \text{rappel}}{\text{précision} + \text{rappel}}$

FraCaS : suite de tests composée de 346 problèmes de DIT impliquant une sémantique profonde.

GenI : réalisateur de surface basé sur les grammaires d'arbres adjoints.

GrGen : outil de réécriture de graphes.

LPO : Logique du Premier Ordre.

NomBank : corpus annoté sur les relations prédicat-argument pour les propositions nominales.

NomLexPlus : lexique de nominalisations pour l'anglais.

Nutcraker : le système de détection d'implications textuelles de Johan Bos.

précision : $\frac{\text{nombre d'implications bien détectées}}{\text{nombre d'implications détectées}}$

PropBank : corpus annoté sur les relations prédicat-argument pour les propositions verbales.

rappel : $\frac{\text{nombre d'implications bien détectées}}{\text{nombre d'implications existantes}}$

RTE : Recognizing Textual Entailment.

SP : Sémantique Plate.

TAL : Traitement Automatique des Langues.

WordNet : base de données lexicales pour la langue anglaise.

XTag : grammaire d'arbres adjoints lexicalisée de l'anglais.

Annexe A

Normalisation

A.1 Hiérarchie complète des classes d'arcs et de nœuds

- Classes de nœuds :
  - **gnode** : classe de nœuds génériques
    - *gid* : identifiant du graphe auquel le nœud appartient
    - *used* : booléen permettant de savoir si le nœud a été utilisé
    - **ncst** : un constituant
      - cst : type de constituant (e.g., NNP, VP, ...)
      - **nword** : un mot
        - wid : identifiant du mot représentant sa position dans la phrase
        - word : le mot tel qu'il apparait dans le texte
        - lemma : le lemme associé au mot
        - verb : le verbe pouvant être associé au mot en cas de nominalisation d'un verbe
        - **nprop** : proposition sémantique pour l'étiquetage
          - rewrittenby : pour savoir quelle règle a créée l'annotation
    - **nsem** : classe générique pour les nœuds sémantiques
      - **semroot** : racine de la formule sémantique
      - **sempred** : prédicat logique
        - name : nom du prédicat logique
      - **semvar** : variable associée à un prédicat logique
        - varid : identifiant de la variable
      - **semcon** : classe générique pour les connecteurs logiques
        - **semconj** : conjonction
        - **semdisj** : disjonction
        - **semneg** : négation
        - **semimpl** : implication
      - **semquant** : classe générique pour les quantificateurs
        - varid : identifiant de la variable associée au quantificateur

- **semexists** : quantificateur existentiel
- **semforall** : quantificateur universel
- **semconst** : constante

– Classes d'arcs :

- **gedge**
  - *gid* : identifiant du graphe auquel l'arc appartient
- **ecst** : relie les constituants et les mots entre eux
- **edep** : classe générique pour les dépendances
  - **dep** : connexion générique entre deux mots
  - **nsubj** : connexion entre un verbe et son sujet
  - **dobj** : connexion entre un verbe et son objet
  - ...
- **eprop** : classe générique pour les arcs d'étiquetage
  - **epred** : liaison de la proposition au mot représentant le prédicat
  - **earg** : classe générique pour les arguments
    - **arg0** : liaison de la proposition au mot représentant le premier argument
    - **arg1** : liaison de la proposition au mot représentant le second argument
    - ...
- **esem** : classe générique pour les arcs sémantiques
  - **con** : classe générique pour lier les connecteurs logiques à leurs arguments
    - **con0** : lie un connecteur logique à son premier argument ou à tous ces arguments s'il est commutatifs
    - **con1** : lie une implication à sa partie droite
  - **var** : classe générique pour lier les prédicats logiques à leurs arguments
    - **var0** : lie le prédicat à son premier argument
    - **var1** : lie le prédicat à son second argument
  - **semhead** : lie un mot à la tête sémantique qui lui est associé
  - **semrest** : lie un mot à la restriction sémantique qui lui est associé
  - **semscope** : lie un mot à la portée sémantique qui lui est associé
  - **semquant** : lie un mot au quantificateur qui lui est associé

A.2 Exemples de normalisations

Le premier exemple, figure A.1, représente la normalisation de la phrase « Every man who sings dances » qui contient une relative. La sémantique associée est :

$$\forall V0, ((man(X) \wedge \exists V1.(sing(V1) \wedge arg0(V1, V0))) \Rightarrow (\exists V2.(dance(V2) \wedge arg0(V2, V0))))$$

Le second exemple, figure A.2, représente la normalisation de la phrase « John wants that Mary leaves » qui contient un verbe à contrôle. La sémantique associée est :

$$john(C0) \wedge mary(C2) \wedge \exists V3.(leave(V3) \wedge arg0(V3, C2) \exists V1.(want(V1) \wedge arg0(V1, C0) \wedge arg1(V1, V3)))$$

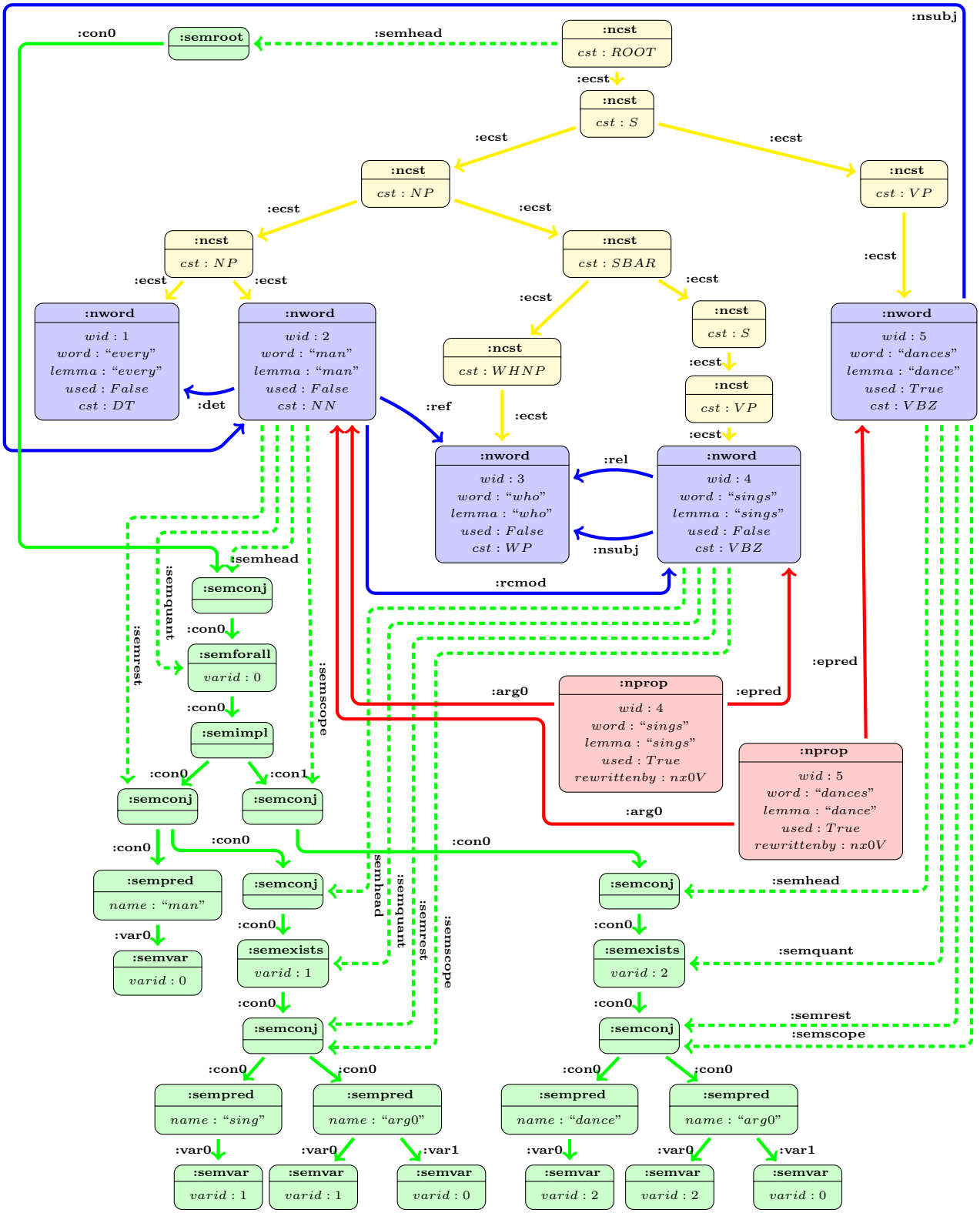


FIGURE A.1 – Représentation complète de la phrase « Every man who sings dances »

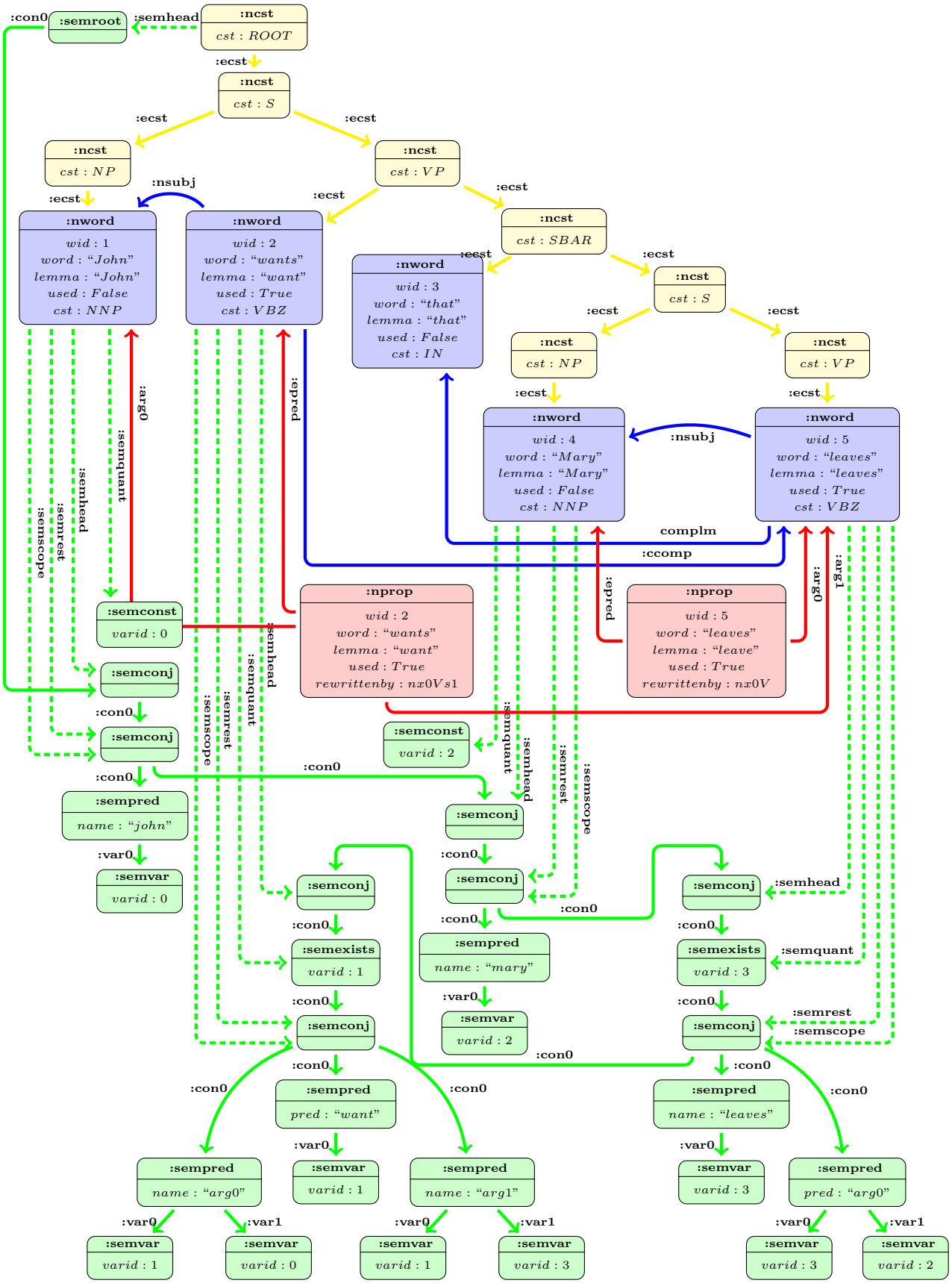


FIGURE A.2 – Représentation complète de la phrase « John wants that Mary leaves »

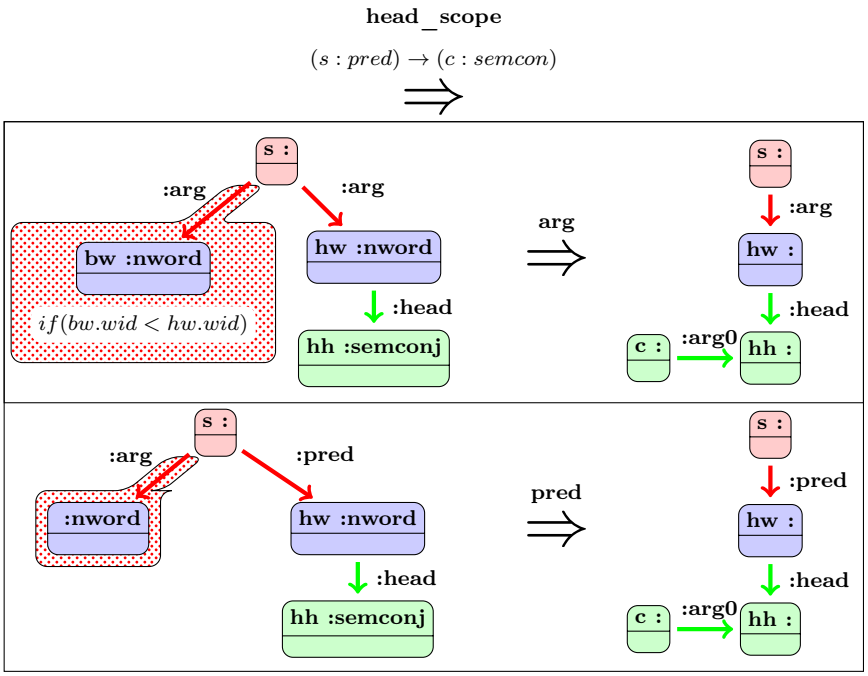


FIGURE A.3 – sous-règle de réécriture **head_scope**

A.3 Règles de réécriture de la dérivation sémantique

Cette section de l'annexe donne le comportement exact des règles de réécritures utilisées pour dériver la sémantique. Nous décrivons dans un premier temps les trois sous-règles utilisées par les règles principales, puis nous décrivons les règles principales.

A.3.1 Les sous-règles

head_scope (Figure A.3) Cette sous règle permet de récupérer la tête d'une proposition afin de la rattacher à un connecteur donné en paramètre de transformation. Elle cherche l'argument apparaissant le premier dans la phrase, et s'il n'y a pas d'argument, prend le prédicat. Cela permet ainsi de rattacher facilement une formule à la tête d'une proposition pour les compléments phrastiques par exemple.

super_scope (Figure A.4) Cette sous-règle permet de récupérer la tête qui succède à un fragment **mw** associé à une proposition **s** et d'y rattacher un connecteur logique **c**. Cette règle est composée de trois alternatives. La première alternative **arg_arg** gère le cas où **mw** est un argument de **s** et où la tête qui le succède appartient à l'argument qui apparait juste après lui dans le texte et qui n'est pas un prédicat puis relie **c** à la tête de ce successeur. L'alternative **arg_argpred** gère le cas où **mw** est un argument de **s** et où la tête qui le succède appartient à l'argument

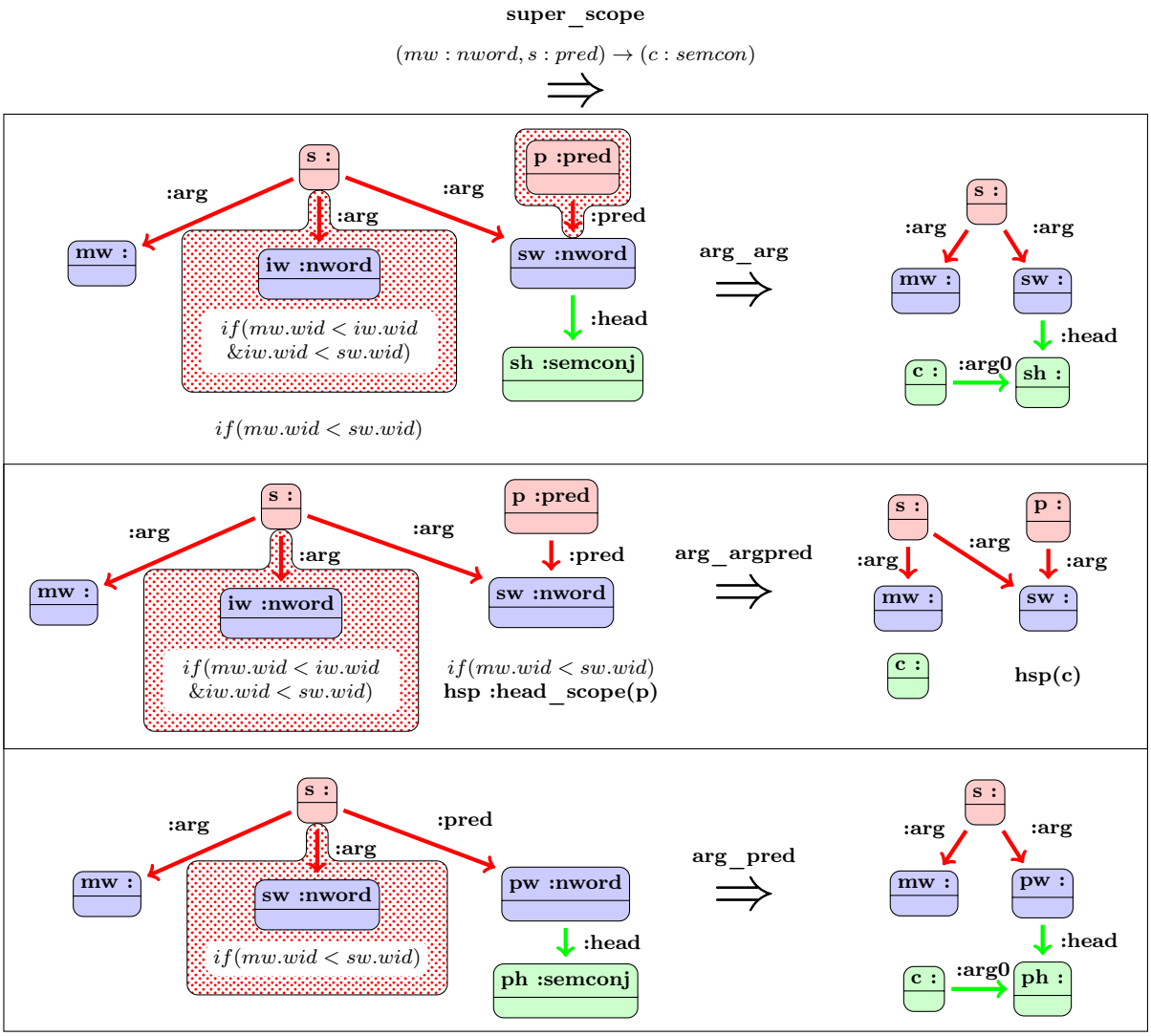


FIGURE A.4 – sous-règle de réécriture **super_scope**

qui apparait juste après lui dans le texte et qui est un prédicat puis relie **c** à la tête du syntagme prédicatif associé grâce à la sous-règle **head_scope**. L'alternative **arg_pred** gère le cas où **mw** est un argument de **s** et où il n'y a plus d'argument se situant après dans le texte. Elle rattache alors **c** au prédicat de la proposition **s**. Cette sous-règle sera utilisée dans la règle **link_structs** qui sert à lier les fragments de sémantique entre eux.

exists_path (Figure A.5) Lorsqu'un fragment est l'argument de deux propositions, il prend la proposition satellite (e.g., une relative) comme restriction et la principale comme portée. La proposition principale se différencie de la proposition secondaire car il est possible d'atteindre à partir de son prédicat et via les arcs de

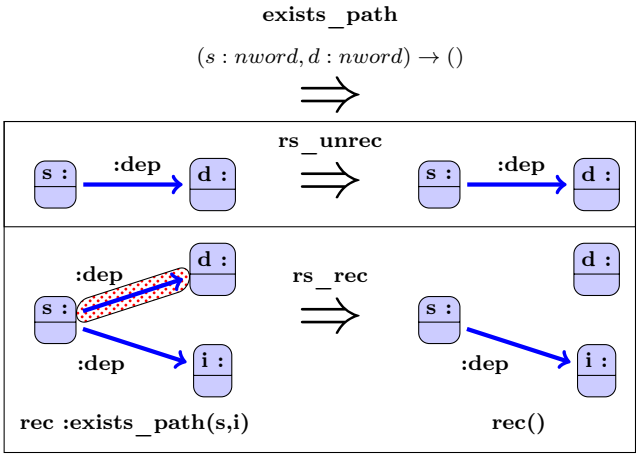


FIGURE A.5 – sous-règle de réécriture **exists_path**

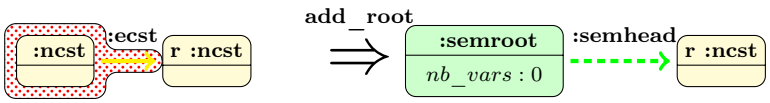


FIGURE A.6 – règle de réécriture **add_root**

dépendances, le fragment de l'argument partagé. Par exemple, si on a une proposition relative, l'argument partagé pourra être atteint par le prédicat de la proposition principale mais pas par celui de la proposition relative. C'est pourquoi nous avons défini la sous-règle récursive **exits_path** qui vérifie s'il existe un chemin entre deux nœuds **s :nword** et **d :nword**. Cette règle peut fonctionner car les graphes dépendances sont acycliques, sinon il aurait été possible de boucler à l'infini. La règle est composée de deux alternatives, soit il existe un chemin direct entre **s** et **d**, soit on prend un nœud atteignable par **s** et on regarde s'il existe un chemin entre ce nœud et **d**.

A.3.2 Les règles principales

Nous allons maintenant voir les règles de réécriture permettant de générer un arbre syntaxique de formule logique à partir de nos représentations hybrides, dans leur ordre d'application.

add_root (Figure A.6) Pour commencer, nous avons une première règle qui consiste uniquement à associer une racine sémantique à la racine syntaxique. Pour trouver la racine syntaxique nous cherchons un nœud **:ncst** sans parent. Pour générer la formule sémantique finale il faudra partir de cette racine et parcourir la structure sémantique.

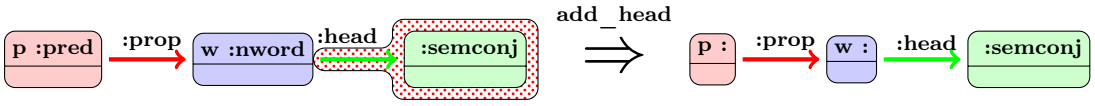


FIGURE A.7 – règle de réécriture **add_head**

add_head (Figure A.7) Cette règle permet de rajouter des têtes sémantiques à chaque mot étiqueté comme prédicat ou argument. Cela permet ainsi d'avoir un point d'attache pour les règles suivantes qui consistent à lier les fragments sémantiques entre eux. Nous avons une conjonction comme tête car ce connecteur logique a une monotonie descendante qui est celle nécessaire pour l'implication ($a \wedge b \Rightarrow a$, $a \wedge b \Rightarrow b$).

add_structs (Figure A.8) La règle que nous allons maintenant décrire est l'une des plus complexes du système. Elle permet de gérer l'ajout d'une restriction, d'une portée et d'un quantificateur aux mots ayant une tête sémantique. Elle relie ensuite ces différents éléments entre eux à l'aide de connecteurs logiques. Cette règle est composée d'un règle principale avec une alternative pour chaque type de quantificateur et une autre pour les prédicats. Nous n'avons pas mis les alternatives pour les quantificateurs « no » et « not every » car la règle ne rentrait plus dans une page et que ces alternatives ressemblent beaucoup à celles de « a » et « every ». La règle principale filtre les graphes contenant un étiquetage prédicatif **s**, un de ses arguments ou son prédicat **mw** ainsi que la tête **mh** de ce dernier. La règle filtre aussi la racine sémantique afin de pouvoir incrémenter le compte des variables quantifiées. Pour finir, elle vérifie que **mw** n'a pas déjà un quantificateur associé. La transformation principale incrémente uniquement la racine sémantique. La première alternative **forall** filtre si il y a un déterminant quantifiant universellement **mw** et que **mw** n'est pas le prédicat de **s**. Elle connecte alors la formule $\forall X.$**restriction** $\Rightarrow$ **portée** à la tête de **mw** puis relie **mw** au quantificateur, à la restriction et à la portée. Au même titre que pour la tête, les connecteurs logiques associés à la restriction et à la portée sont des conjonctions. L'identifiant du quantificateur correspond au nombre de quantificateurs déjà instanciés. L'alternative **exists** fait la même chose que l'alternative précédente, mis à part qu'il filtre un déterminant quantifiant existentiellement **mw** et que la formule liée à la tête de **mw** est $\exists X.$**restriction-portée**. Dans cette règle nous avons un seul connecteur logique pour la restriction et la portée car une conjonction est commutative et associative contrairement à l'implication et qu'il n'est donc pas nécessaire de les séparer. La troisième alternative **constant** vérifie que **mw** n'a pas de déterminant et qu'il n'est pas le prédicat de **s**. Ensuite elle crée un quantificateur représentant la constante qui est indépendant de la formule et connecte la tête de **mw** directement au connecteur logique associé à la restriction et à la portée de **mw**.

Enfin la dernière règle **predicate** vérifie que **mw** est le prédicat de **s** et réalise la même transformation que la règle **exists** où le quantificateur représente la variable événementielle. Cette règle permet donc de quantifier les arguments et le prédicat, puis de leur associer un fragment de sémantique.

link_structs (Figure A.9) Cette règle a pour but de lier les fragments les uns aux autres. La règle principale sert uniquement à filtrer un prédicat et un argument. La première alternative **general** sert à gérer le cas où **mw** est l'argument principal d'une proposition. Elle regarde si le mot **mw** est l'argument d'une proposition, et qu'il existe un chemin de dépendances entre le prédicat de la proposition et lui. Elle vérifie ensuite que **mw** n'est pas un prédicat puis utilise la sous-règle **super_scope** pour retrouver la tête du fragment suivant dans la proposition **s**, puis le connecte à la portée de **mw**. La seconde alternative **relative** est similaire à la première sauf que cette fois, il ne faut pas que **mw** soit atteignable depuis de prédicat de **s** et que la sous-règle **super_score** connecte la tête du fragment dans la proposition **s**, à la restriction de **mw**. Enfin, la dernière alternative **predicate** vérifie que le mot **mw** est le prédicat de **s** et qu'il est en même temps l'argument d'une autre proposition **p**. Dans ce cas il utilise la sous-règle **super_scope** pour relier la portée de **mw** à la tête du fragment suivant dans la proposition **p**. Cette règle permet ainsi de connecter les fragments entre eux afin d'avoir une structure complète de formule.

fill_structs (Figure A.10) Cette règle permet de rajouter les prédicats logiques correspondant aux arguments, et aux prédicats, sur la structure arborescente que l'on a créée. La règle principale filtre un mot avec sa restriction et son quantificateur et ne change rien lors de la transformation. L'alternative **sem_arg** permet de rajouter les prédicats logiques associés aux mots qui ne sont pas des noyaux prédicatifs. Elle rajoute à la restriction du mot un prédicat correspondant au lemme du mot avec comme argument son quantificateur. L'alternative **sem_pred** permet de rajouter à la restriction des prédicats, un prédicat logique correspondant au verbe associé à l'étiquetage lors de la phase de normalisation verbale/nominale, avec comme argument le quantificateur associé au noyau. Enfin l'alternative **sem_predarg** permet de faire les lien entre le prédicat et ses arguments. Pour cela, il prend chaque couple de prédicat-argument et crée un prédicat logique correspondant à la liaison qui existe entre eux (e.g., **arg0**, **arg1**, ...), avec comme premier argument le quantificateur du prédicat et comme second argument le quantificateur de l'argument. Cette règle à donc permis de décorer notre structure syntaxique de formule logique avec des prédicats logiques.

link_root (Figure A.11) Pour finir, cette règle prend tous les connecteurs logiques sans connecteur logique entrant et les relie à la racine sémantique.

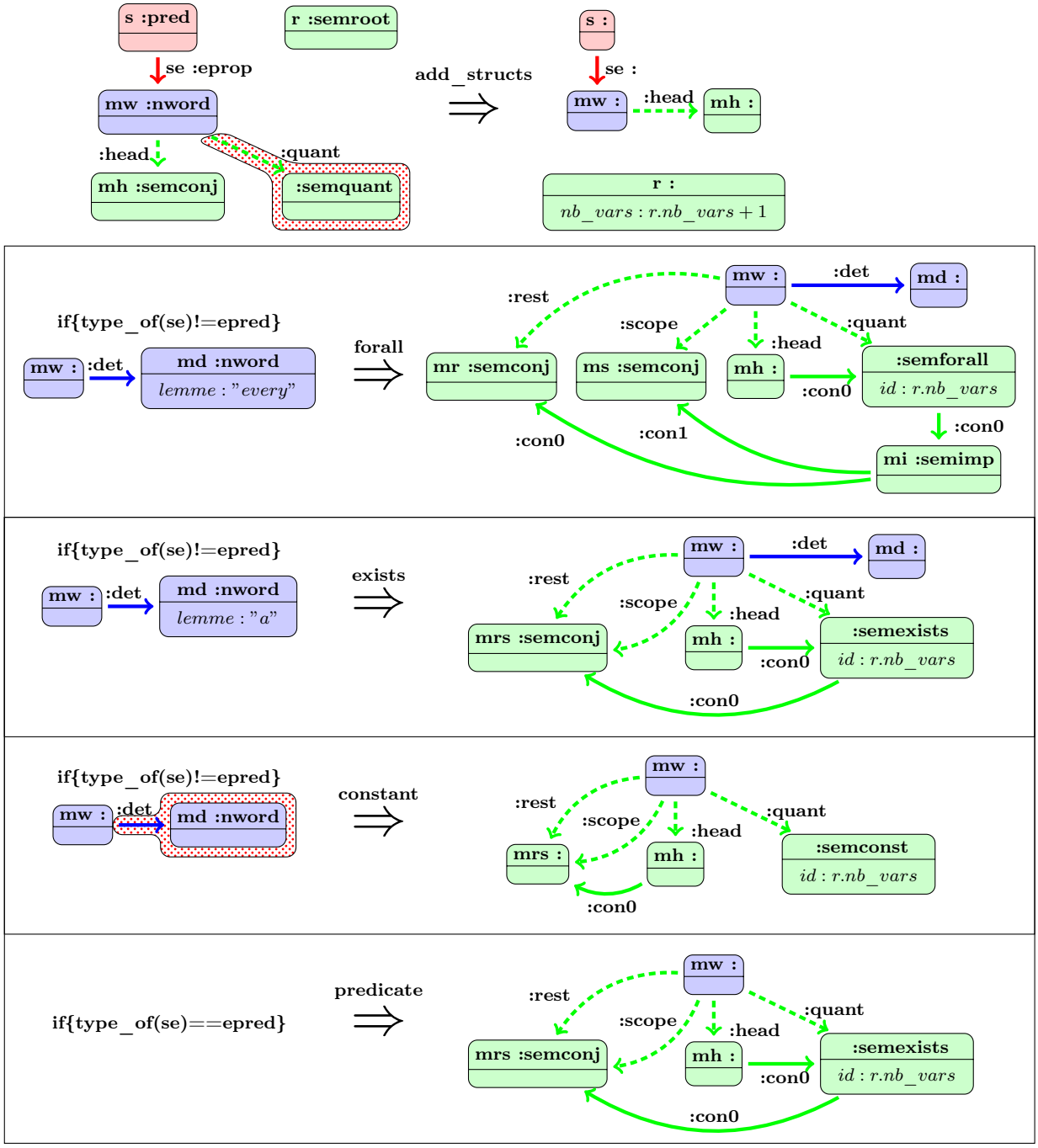


FIGURE A.8 – règle de réécriture **add_structs**

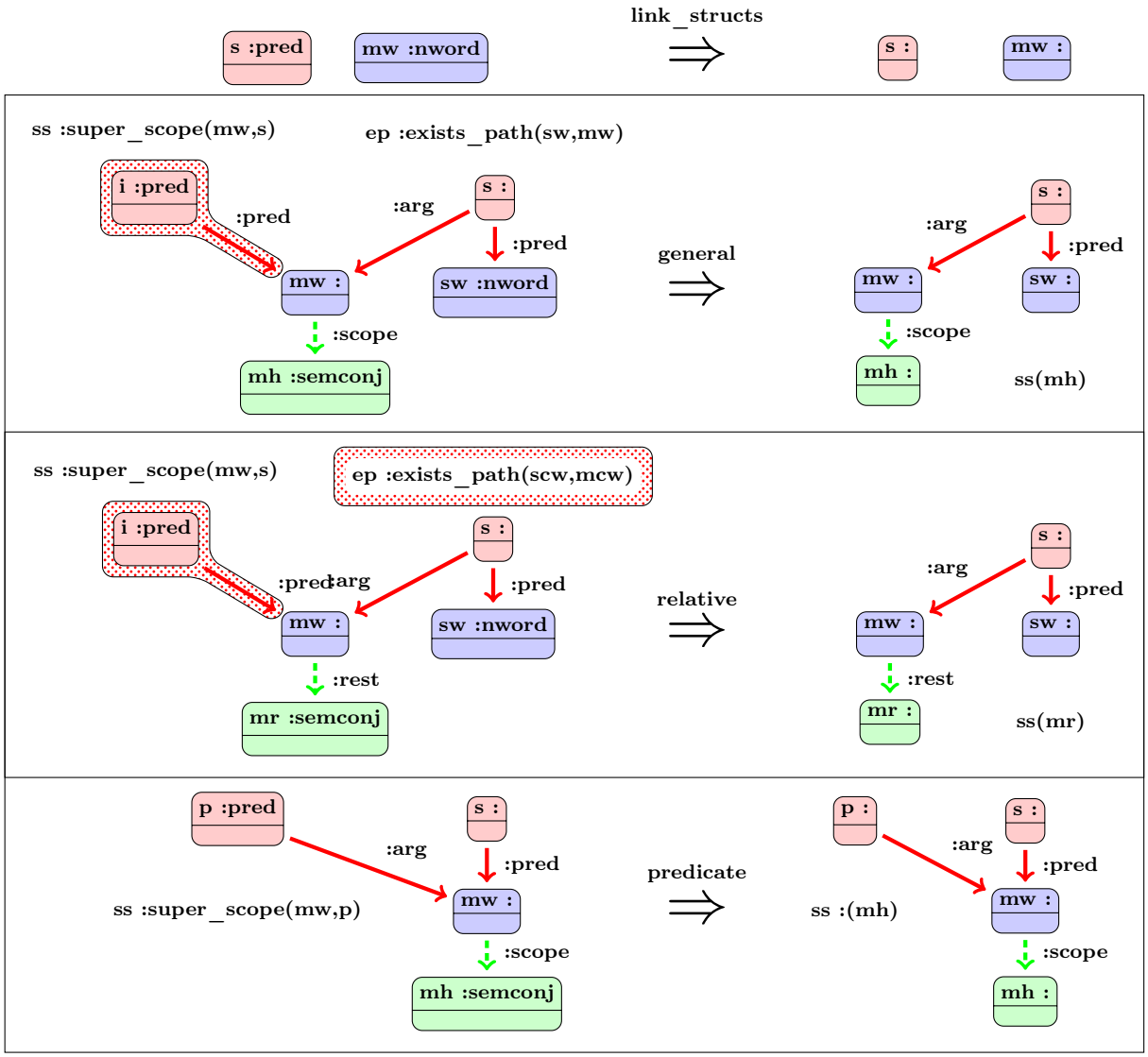


FIGURE A.9 – règle de réécriture **link_structs**

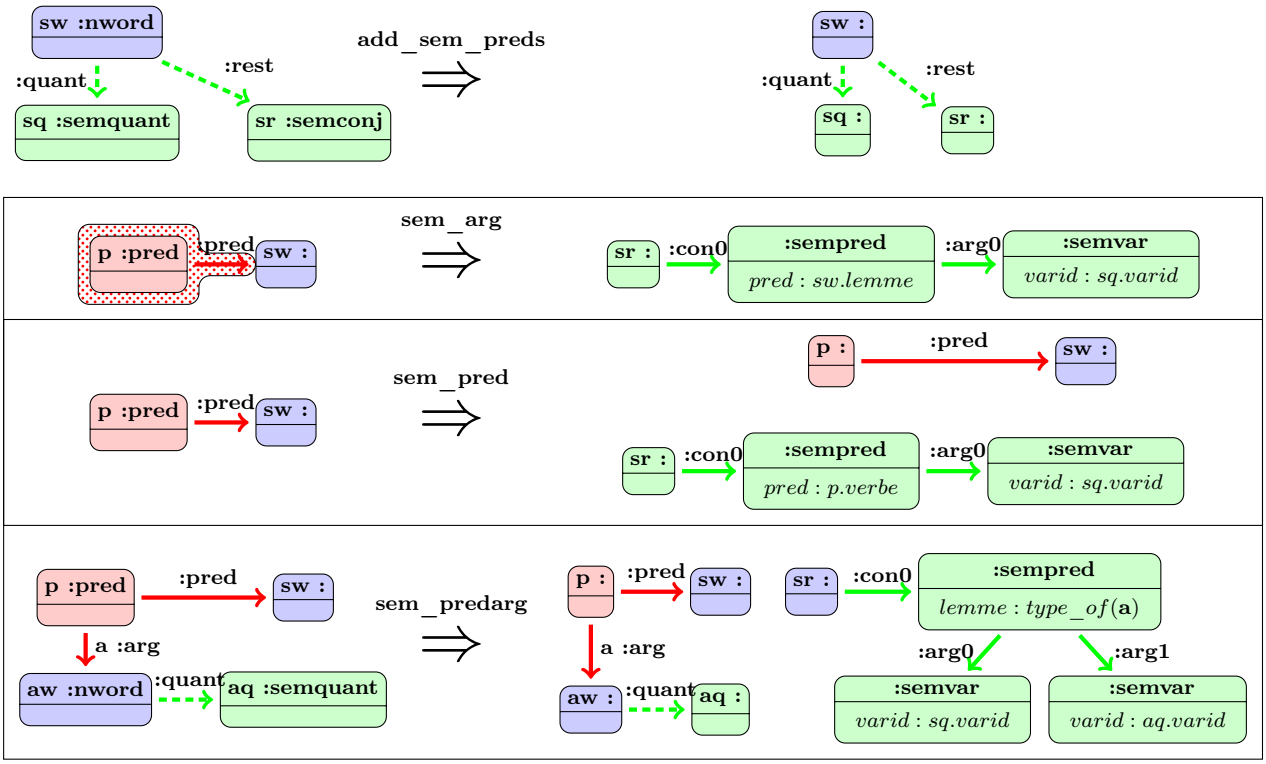


FIGURE A.10 – règle de réécriture **add_sem_pred**

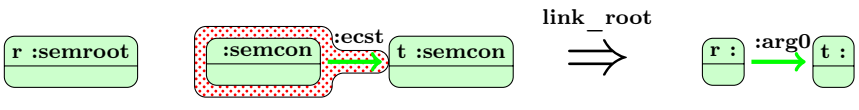
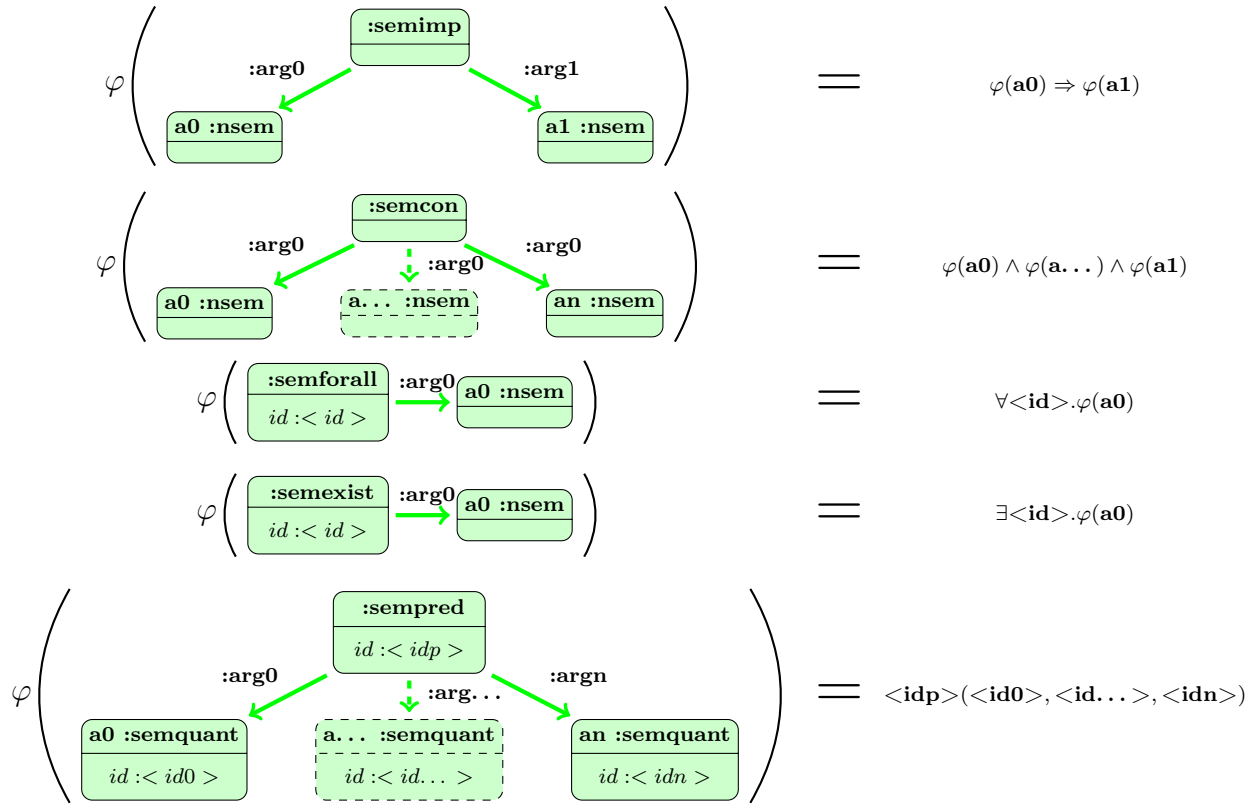


FIGURE A.11 – règle de réécriture **add_root**

Création de la formule

Une fois l'arbre syntaxique de la formule obtenue, il faut le transformer en texte pour pouvoir tester l'implication entre des formules via des outils de preuves. Pour obtenir la formule on part de la racine puis on crée la formule on parcourant récursivement l'arbre à l'aide de la fonction φ. Ci-dessous on peut voir les transformations que l'on a pour chaque type de nœud par rapport à ses fils. La racine est considérée comme une conjonction. Une fois les formules générées nous pouvons utiliser des outils de preuve pour vérifier si une phrase en implique une autre comme nous l'avons vu dans la section 2.4.



Bibliographie

[Alahverdzhieva, 2008] Katja Alahverdzhieva. Reimplementing xtag using xmg. Master's thesis, Master "Language and Communication Technology", Saarland University/Nancy University, September 2008.

[Bar-Haim *et al.*, 2006] Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, & Idan Szpektor. The second pascal recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, Venice, Italy, 2006.

[Bar-Haim *et al.*, 2007] Roy Bar-Haim, Ido Dagan, Iddo Greental, Idan Szpektor, & Moshe Friedman. Semantic inference at the lexical-syntactic level for textual entailment recognition. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 131–136, Prague, June 2007. Association for Computational Linguistics.

[Bayer *et al.*, 2005] Samuel Bayer, John D. Burger, Lisa Ferro, John C. Henderson, Lynette Hirschman, & Alexander S. Yeh. Evaluating semantic evaluations : How rte measures up. In *MLCW*, pages 309–331, 2005.

[Bentivogli *et al.*, 2009] Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, & Bernardo Magnini. The fifth pascal recognizing textual entailment challenge. In *TAC 2009 Workshop*. no publisher, 2009.

[Blackburn & Bos, 2005] Patrick Blackburn & Johan Bos. *Representation and Inference for Natural Language. A First Course in Computational Semantics*. CSLI, 2005.

[Bond & Fox, 2001] Trevor G. Bond & Christine M. Fox. *Applying the Rasch Model : Fundamental Measurement in the Human Sciences.* University of Toledo Press, 2001.

[Bos & Markert, 2005] Johan Bos & Katja Markert. Recognising textual entailment with logical inference. In *HLT '05 : Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 628–635, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

[Bos *et al.*, 2004] Johan Bos, Stephen Clark, Mark Steedman, James R. Curran, & Julia Hockenmaier. Wide-coverage semantic representations from a ccg parser.

In *Proceedings of the 20th International Conference on Computational Linguistics (COLING '04)*, pages 1240–1246, Geneva, Switzerland, 2004.

[Carreras & Màrquez, 2005] Xavier Carreras & Lluís Màrquez. Introduction to the CoNLL-2005 shared task : Semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 152–164, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[Clark & Curran, 2007] Stephen Clark & James R. Curran. Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistics*, 33(4) :493–552, 2007.

[Cohen, 1960] Jacob. Cohen. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1) :37, 1960.

[Copestake & Flickinger, 2000] Ann Copestake & Dan Flickinger. An open source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation*, Athens, Greece, 2000.

[Copestake *et al.*, 2005] Ann Copestake, Dan Flickinger, Carl Pollard, & Ivan A. Sag. Minimal recursion semantics : an introduction. *Research on Language and Computation*, 3.4 :281–332, 2005.

[Crouch *et al.*, 2005] Dick Crouch, Saurí Roser, & Fowler Abraham. Aquaint pilot knowledge-based evaluation : Annotation guidelines, May 2005.

[de Kok *et al.*, 2009] Daniel de Kok, Jianqiang Ma, & Gertjan van Noord. A generalized method for iterative error mining in parsing results. In *Proceedings of the 2009 Workshop on Grammar Engineering Across Frameworks (GEAF 2009)*, pages 71–79, Suntec, Singapore, August 2009. Association for Computational Linguistics.

[de Marneffe *et al.*, 2006] Marie-Catherine de Marneffe, Bill MacCartney, & Christopher D. Manning. Generating typed dependency parses from phrase structure parses. In *Language Resources and Evaluation Conference (LREC)*, page (electronic medium), Gènes, 2006. European Language Resources Association (ELRA).

[Fellbaum, 1998] Christiane Fellbaum. *WordNet An Electronic Lexical Database*. The MIT Press, Cambridge, MA ; London, May 1998.

[Fowler *et al.*, 2005] Abraham Fowler, Bob Hauser, Daniel Hodges, Ian Niles, Adrian Novischi, & Jens Stephan. Applying cogex to recognize textual entailment. In *In Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*, pages 69–72, 2005.

[Gardent & Kallmeyer, 2003] Claire Gardent & Laura Kallmeyer. Semantic construction in ftag. In *Proceedings of the 10th meeting of the European Chapter of the Association for Computational Linguistics*, Budapest, Hungary, 2003.

[Gardent & Kow, 2007a] Claire Gardent & Eric Kow. GenI, un réalisateur basé sur une grammaire réversible. In *14e conférence pour le Traitement Automatique des Langues Naturelles - TALN 2007*, Toulouse/France, 2007.

[Gardent & Kow, 2007b] Claire Gardent & Eric Kow. A symbolic approach to near-deterministic surface realisation using tree adjoining grammar. In *ACL07*, 2007.

[Garoufi, 2007] Konstantina Garoufi. Towards a better understanding of applied textual entailment : Annotation and evaluation of the rte-2 dataset. Master's thesis, Saarland University, September 2007.

[Giampiccolo *et al.*, 2008] Danilo Giampiccolo, Hao Trang Dang, , Bernardo Magnini, Ido Dagan, Elena Cabrio, & Bill Dolan. The fourth pascal recognizing textual entailment challenge. In *Proceedings of the Fourth Workshop on Textual Entailment and Paraphrasing*, Gaithersburg, Maryland, November 2008.

[Girard *et al.*, 1989] Jean-Yves Girard, Paul Taylor, & Yves Lafont. *Proofs and types*. Cambridge University Press, New York, NY, USA, 1989.

[Group, 2001] XTAG Research Group. A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania, 2001.

[Hajič *et al.*, 2009] Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, & Yi Zhang. The CoNLL-2009 shared task : Syntactic and semantic dependencies in multiple languages. In *Proceedings of the 13th Conference on Computational Natural Language Learning (CoNLL-2009), June 4-5*, Boulder, Colorado, USA, 2009.

[Harris, 1954] Zellig Harris. Distributional structure. *Word*, 10 :146–162, 1954.

[Hickl *et al.*, 2006] Andrew Hickl, Jeremy Bensley, John Williams, Kirk Roberts, Bryan Rink, & Ying Shi. Recognizing textual entailment with lcc's groundhog system. In *Proceedings of the Second PASCAL Recognizing Textual Entailment Challenge*, April 2006.

[Higginbotham, 2000] James Higginbotham. On events in linguistic semantics. In James Higginbotham, Fabio Pianesi, & Achille Varzi, editors, *Speaking of Events*. Oxford University Press, Oxford, 2000.

[Horridge *et al.*, 2008] Matthew Horridge, Bijan Parsia, & Ulrike Sattler. Laconic and precise justifications in owl. In *ISWC '08 : Proceedings of the 7th International Conference on The Semantic Web*, pages 323–338, Berlin, Heidelberg, 2008. Springer-Verlag.

[Jijkoun & de Rijke, 2007] Valentin Jijkoun & Maarten de Rijke. Learning to transform linguistic graphs. In *Proceedings of the Second Workshop on TextGraphs : Graph-Based Algorithms for Natural Language Processing*, pages 53–60, Rochester, NY, USA, 2007. Association for Computational Linguistics.

[Kay, 1996] Martin Kay. Chart generation. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 200–204, Morristown, NJ, USA, 1996. Association for Computational Linguistics.

[Klein & Manning, 2003] Dan Klein & Christopher D. Manning. Accurate unlexicalized parsing. In *ACL '03 : Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 423–430, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[Kroll & Geiß, 2007] Moritz Kroll & Rubino Geiß. Developing graph transformations with grgen.net. Technical report, October 2007. preliminary version, submitted to AGTIVE 2007.

[Levy & Andrew, 2006] Roger Levy & Galen Andrew. Tregex and tsurgeon : tools for querying and manipulating tree data structures. In *In LREC 2006*, 2006.

[Lin & Pantel, 2001] Dekang Lin & Patrick Pantel. Discovery of inference rules for question answering. *Natural Language Engineering*, 7 :343–360, 2001.

[MacCartney & Manning, 2009] Bill MacCartney & Christopher D. Manning. An extended model of natural logic. In *Proceedings of the Eighth International Conference on Computational Semantics (IWCS-8)*, 2009.

[Macleod *et al.*, 1998a] Catherine Macleod, Ralph Grishman, & Adam Meyers. *COMLEX Syntax Reference Manual*, 1998.

[Macleod *et al.*, 1998b] Catherine Macleod, Ralph Grishman, Adam Meyers, Leslie Barrett, & Ruth Reeves. Nomlex : A lexicon of nominalizations. In *In Proceedings of Euralex98*, pages 187–193, 1998.

[Marcus *et al.*, 1993] Mitchell P. Marcus, Mary Ann Marcinkiewicz, & Beatrice Santorini. Building a large annotated corpus of english : the penn treebank. *Comput. Linguist.*, 19(2) :313–330, 1993.

[Maxwell, 1996] R. M. Maxwell, J. T. ; Kaplan. Unification-based parsers that automatically take advantage of context freeness. In *The First LFG Conference*, Grenoble, France, 1996.

[Meyers *et al.*, 2004] Adam Meyers, Ruth Reeves, Catherine Macleod, Rachel Szekeley, Veronkia Zielinska, & Brian Young. The cross-breeding of dictionaries. In *Proceedings of LREC-2004*, Lisbon, Portugal, 2004.

[Nairn *et al.*, 2006] Rowan Nairn, Cleo Condoravdi, & Lauri Karttunen. Computing relative polarity for textual inference. In *In Proceedings of ICoS-5 (Inference in Computational Semantics)*, 2006.

[Nivre *et al.*, 2005] Joakim Nivre, Johan Hall, Sandra Kübler, & Erwin Marsi. Maltparser : A language-independent system for data-driven dependency parsing. In *In Proc. of the Fourth Workshop on Treebanks and Linguistic Theories*, pages 13–95, 2005.

[Palmer *et al.*, 2005] Martha Palmer, Daniel Gildea, & Paul Kingsbury. The proposition bank : An annotated corpus of semantic roles. *Comput. Linguist.*, 31(1) :71–106, 2005.

[Pradhan *et al.*, 2005] Sameer Pradhan, Wayne Ward, Kadri Hacioglu, James H. Martin, & Daniel Jurafsky. Semantic role labeling using different syntactic views. In *ACL '05 : Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 581–588, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

[Sagot & Éric de La Clergerie, 2008] Benoît Sagot & Éric de La Clergerie. *Fouille d'erreurs sur des sorties d'analyseurs syntaxiques*, pages 41–60. 2008.

[Shanker & Joshi, 1988] Vijay K. Shanker & Aravind K. Joshi. Feature structures based tree adjoining grammars. In *Proceedings of the 12th conference on Computational linguistics*, pages 714–719, Morristown, NJ, USA, 1988. Association for Computational Linguistics.

[Sleator & Temperley, 1993] Daniel D. K. Sleator & Davy Temperley. Parsing English with a link grammar. In *Third International Workshop on Parsing Technologies*, 1993.

[The Fracas Consortium *et al.*, 1996] The Fracas Consortium, Robin Cooper, Dick Crouch, Jan Van Eijck, Chris Fox, Josef Van Genabith, Jan Jaspars, Hans Kamp, David Milward, Manfred Pinkal, Massimo Poesio, Steve Pulman, Ted Briscoe, Holger Maier, & Karsten Konrad. Using the framework, 1996.

[Zaenen *et al.*, 2005] Annie Zaenen, Lauri Karttunen, & Richard Crouch. Local textual inference : Can it be defined or circumscribed ? In *Proceedings of the ACL Workshop on Empirical Modeling of Semantic Equivalence and Entailment*, pages 31–36, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[Zanzotto *et al.*, 2006] Fabio Massimo Zanzotto, Alessandro Moschitti, Marco Pennacchiotti, & Maria Teresa Pazienza. Learning textual entailment from examples. In *Proceedings of the Second PASCAL Recognizing Textual Entailment Challenge*, April 2006.

Résumé

Cette thèse propose plusieurs contributions sur le thème de la détection d'implications textuelles (DIT). La DIT est la capacité humaine, étant donné deux textes, à pouvoir dire si le sens du second texte peut être déduit à partir de celui du premier. Une des contributions apportée au domaine est un système de DIT hybride prenant les analyses d'un analyseur syntaxique stochastique existant afin de les étiqueter avec des rôles sémantiques, puis transformant les structures obtenues en formules logiques grâce à des règles de réécriture pour tester finalement l'implication à l'aide d'outils de preuve. L'autre contribution de cette thèse est la génération de suites de tests finement annotés avec une distribution uniforme des phénomènes couplée avec une nouvelle méthode d'évaluation des systèmes utilisant les techniques de fouille d'erreurs développées par la communauté de l'analyse syntaxique permettant une meilleure identification des limites des systèmes. Pour cela nous créons un ensemble de formules sémantiques puis nous générons les réalisations syntaxiques annotées correspondantes à l'aide d'un système de génération existant. Nous testons ensuite s'il y a implication ou non entre chaque couple de réalisations syntaxiques possible. Enfin nous sélectionnons un sous-ensemble de cet ensemble de problèmes d'une taille donnée et satisfaisant un certain nombre de contraintes à l'aide d'un algorithme que nous avons développé.

Mots-clés: chat, chien, puces.

Abstract

This thesis presents several contributions on the theme of recognising textual entailment (RTE). The RTE is the human capacity, given two texts, to determine whether the meaning of the second text could be deduced from the meaning of the first or not. One of the contributions made to the field is a hybrid system of RTE taking analysis of an existing stochastic parser to label them with semantics roles, then turning obtained structures in logical formulas using rewrite rules to finally test the entailment using proof tools. Another contribution of this thesis is the generation of finely annotated tests suites with a uniform distribution of phenomena coupled with a new methodology of systems evaluation using error minning techniques developed by the community of parsing allowing better identification of systems limitations. For this, we create a set of formulas, then we generate annotated syntactics realisations corresponding by using an existing generation system. Then, we test whether or not there is an entailment between each pair of possible syntactics realisations. Finally, we select a subset of this set of problems of a given size and a satisfactory a certain number of constraints using an algorithm that we developed.

Keywords: chat, chien, puces.