



HAL
open science

Grammaires locales étendues : principes, mise en œuvre et applications pour l'extraction de l'information

Cristian Martinez

► To cite this version:

Cristian Martinez. Grammaires locales étendues : principes, mise en œuvre et applications pour l'extraction de l'information. Traitement du texte et du document. Université Paris-Est, 2017. Français. NNT : 2017PESC1075 . tel-01799012

HAL Id: tel-01799012

<https://theses.hal.science/tel-01799012v1>

Submitted on 24 May 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Grammaires locales étendues

Principes, mise en œuvre et applications pour
l'extraction de l'information

UNIVERSITÉ —
— PARIS-EST

Cristian Martinez
Laboratoire d'Informatique Gaspard-Monge
Ecole doctorale MSTIC
Université Paris-Est

Thèse pour obtenir le grade de
Docteur de l'Université Paris-Est
Spécialité : Informatique

Décembre 2017

UNIVERSITÉ —
— PARIS-EST

École doctorale MSTIC
Université Paris-Est

Thèse pour obtenir le grade de
Docteur de l'Université Paris-Est
Spécialité : Informatique

Cristian Martinez

Directrice de thèse : Tita Kyriacopoulou
Co-Directeur de thèse : Antoine Schoen

Grammaires locales étendues
Principes, mise en œuvre et applications pour
l'extraction de l'information

Thèse soutenue le 19 décembre 2017

Jury :

Cédric Fairon	Prof. Université catholique de Louvain	(Rapporteur)
Tita Kyriacopoulou	Prof. Université Paris-Est Marne-la-Vallée	(Directrice de thèse)
Eric Laporte	Prof. Université Paris-Est Marne-la-Vallée	(Examineur)
Denis Maurel	Prof. Université François-Rabelais	(Rapporteur et Président du jury)
Antoine Schoen	Prof. ESIEE Paris	(Co-Directeur de thèse)
Duško Vitas	Prof. Université de Belgrade	(Examineur)

*Ha perdido sus manos el reloj
y el hombre marca el tiempo con las suyas,
siempre girando sobre su propio eje,
ruidoso viajero del espacio,
ese vasto silencio
que no rompen ni su voz ni los gritos,
o su neurótico paso por la tierra,
su ingratitud de hijo pródigo
que jamás retorna,
hasta que suene la hora de su muerte,
a la gran Madre Tierra que le dio la vida.
Ha perdido sus manos el reloj
y el hombre marca el tiempo con las suyas.*

Nicolás Suescún (1937–2017)

Remerciements

La réalisation de ce travail de thèse n'aurait pas été possible sans la disponibilité, la collaboration et les échanges de qualité avec de nombreuses personnes.

J'adresse de chaleureux remerciements à ma directrice de thèse, Tita Kyriacopoulou, pour m'avoir donné l'opportunité de réaliser ce travail, pour son encadrement, ses encouragements et son soutien constant pendant toutes ces années.

Je voudrais remercier sincèrement mon co-directeur, Antoine Schoen, pour m'avoir accordé sa confiance dès le début, pour avoir suivi le déroulement du travail, pour ces suggestions et conseils.

Mes remerciements vont ensuite à tous les membres du Laboratoire d'Informatique Gaspard-Monge (LIGM) et de la Plateforme digitale CORTEXT du Laboratoire Interdisciplinaire Sciences Innovations Sociétés (LISIS) de l'Université Paris-Est qui ont participé de près ou de loin à cette thèse.

Je ne remercierai jamais assez Claude Martineau, qui m'a soutenu dans les moments clefs et qui m'a offert son amitié et sa précieuse orientation. Je n'oublie pas les échanges constructifs avec Matthieu Constant, Dominique Perrin, Dominique Revuz, Philippe Gambette et Alexis Neme.

Je tiens à remercier aussi la disponibilité des membres des équipes administratives du LIGM, du personnel de l'école doctorale MSTIC. Je pense tout particulièrement à Patrice Hérault, Séverine Giboz, Corinne Palescandolo et Sylvie Cach.

Je présente mes vifs remerciements à la communauté des utilisateurs et développeurs d'Unitex sans lesquels ce travail n'aurait pas été ce qu'il est.

Je remercie aussi tous les amis qui m'ont toujours encouragé à continuer malgré les hauts et les bas : Amina, Aline, Marina, Anna, Viviana, Seïf, Michel, Lionel, Gabriel et Hossein. J'ai une pensée spéciale pour mes amies d'enfance : Karol et Jonathan merci infiniment pour être présents malgré le temps et la distance.

Mes plus profonds remerciements vont à mes chers parents Lola et Jesús Martínez, sans qui rien n'aurait été possible, ainsi qu'à mes frères Freddy, Fidel, Leonny et Winston.

Je dédie cette thèse à ma femme Eleni. Je ne saurai jamais assez te remercier pour ton soutien sans faille, ta patiente, ton aide, et tes mots doux qui continuent à me donner du courage.

Résumé

Les grammaires locales constituent un formalisme de description de constructions linguistiques et sont communément représentées sous la forme de graphes orientés. Utilisées pour la recherche et l'extraction de motifs dans un texte, elles trouvent leurs limites dans le traitement de variations non décrites ou fautives ainsi que dans la capacité à accéder à des connaissances exogènes, c'est-à-dire des informations à extraire, au cours de l'analyse, de ressources externes à la grammaire et qui peuvent s'avérer utiles pour normaliser, enrichir, valider ou mettre en relation les motifs reconnus.

Dans cette thèse nous introduisons la notion de grammaire locale étendue. Il s'agit d'un formalisme capable d'étendre le modèle classique des grammaires locales. Premièrement, en ajoutant des fonctions arbitraires à satisfaire, appelées *fonctions étendues*, qui ne sont pas prédéfinies à l'avance et qui sont évaluées en dehors de la grammaire. De surcroît, ce formalisme fournit à l'analyseur syntaxique la possibilité de déclencher des événements qui peuvent également être traités sous la forme de fonctions étendues.

Le travail présenté se divise en trois parties : dans un premier temps, nous étudions les principes concernant la construction des grammaires locales étendues. Nous présentons ensuite la mise en œuvre d'un moteur d'analyse textuelle implémentant le formalisme proposé. Enfin, nous étudions quelques applications pour l'extraction de l'information dans des textes bien formés et des textes bruités. Nous nous focalisons sur le couplage des ressources externes et des méthodes non-symboliques dans la construction de nos grammaires en montrant la pertinence de cette approche pour dépasser les limites des grammaires locales classiques.

Mots-clés : grammaire locale, grammaire locale étendue, apprentissage automatique, extraction de l'information.

Abstract

Local grammars constitute a descriptive formalism of linguistic phenomena and are commonly represented using directed graphs. Local grammars are used to recognize and extract patterns in a text, but they had some inherent limits in dealing with unexpected variations as well as in their capacity to access exogenous knowledge, in other words information to extract, during the analysis, from external resources and which may be useful to normalize, enhance validate or link the recognized patterns.

In this thesis, we introduce the notion of extended local grammar, a formalism capable to extend the classic model of local grammars. The means are twofold: on the one hand, it is achieved by adding arbitrary conditional-functions, called extended functions, which are not predefined in advance and are evaluated from outside of the grammar. On the other hand, it is achieved by allowing the parsing engine to trigger events that can also be processed as extended functions.

The work presented herewith is divided into three parts: In the first part, we study the principles regarding the construction of the extended local grammars. Then, we present a proof-of-concept of a corpus-processing tool which implements the proposed formalism. Finally, we study some techniques to extract information from both well-formed and noisy texts. We focus on the coupling of external resources and non-symbolic methods in the construction of our grammars and we highlight the suitability of this approach in order to overcome the inherent limitations of classical local grammars.

Keywords: local grammar, extended local grammar, machine learning, information extraction.

Table des matières

Notations et symboles	xxvii
I. Notations générales	xxvii
II. Représentation des grammaires locales/étendues	xxix
III. Masques lexicaux utilisées dans les grammaires locales/étendues	xxxii
1 Introduction	1
1.1 Motivation	3
1.2 Problématique de la thèse	6
1.2.1 Grammaire locales : définition utilisée	6
1.2.2 Grammaires locales : leurs limites	7
1.2.3 Grammaire locales : au-delà des limites	9
1.3 Contributions	10
1.4 Description des chapitres	12
1.5 Contenu des annexes	13
2 Grammaires, automates et langages formels	15
2.1 Grammaires formelles	15
2.2 Automates finis	21
2.2.1 Automates finis déterministes	22
2.2.2 Automates finis non-déterministes	25
2.3 Langages réguliers	27
2.3.1 Grammaires régulières	28
2.3.2 Expressions régulières	30

2.4	Langages non contextuels	31
2.4.1	Grammaires non contextuelles	31
2.4.2	Automates à pile	34
2.5	Langages contextuels	36
2.5.1	Grammaires contextuelles	36
2.6	Langages récursivement énumérables	37
2.6.1	Grammaires contextuelles avec effacement	37
2.6.2	Machines de Turing	38
3	Grammaires locales	41
3.1	Aperçu général	41
3.2	Définition	43
3.3	Modélisation	45
3.4	Outils existants	47
3.4.1	INTEX	47
3.4.2	UNITEX	48
3.4.3	NOOJ	48
3.4.4	OUTILEX	48
3.4.5	SXPIPE	48
3.4.6	OPENFST	48
3.5	Caractéristiques principales	49
3.5.1	Faire appel à des sous-graphes	49
3.5.2	Faire référence à des masques lexicaux	49
3.5.3	Faire référence à des méta-symboles	50
3.5.4	Gérer des sorties	50
3.5.5	Stocker des variables locales	50
3.5.6	Utiliser des filtres morphologiques	51
3.5.7	Définir l'unité minimale d'analyse	51
3.5.8	Capacité à reconnaître des langages contextuels	53
3.6	Grammaires locales et extraction de l'information	55

3.7	Chaîne de traitement pour l'extraction de l'information à l'aide des grammaires locales	57
3.8	Autres travaux autour des grammaires locales	60
3.9	Défis liés à la gestion et construction des grammaires locales	62
3.9.1	Gestion des grammaires	62
3.9.2	Génération automatique de grammaires locales	63
3.9.3	Limitations	64
4	Grammaires locales étendues : principes	67
4.1	Aperçu général	69
4.2	Principes	72
4.3	Préliminaires	73
4.4	Modélisation	75
4.5	Aplatissement	80
4.6	Caractéristiques principales	81
4.7	Évaluation des fonctions étendues	81
4.8	Machines abstraites et évaluation des fonction étendues	88
5	Grammaires locales étendues : mise en œuvre	91
5.1	Anatomie de l'appel à une fonction étendue	92
5.1.1	Nuls et booléens	93
5.1.2	Chaînes de caractères littérales	94
5.1.3	Variables d'entrée et de sortie	94
5.1.4	Chaînes de caractères non-littérales	95
5.1.5	Références aux variables d'entrée et de sortie	95
5.1.6	Constantes numériques	97
5.2	Un langage interprété comme infrastructure des fonctions étendues	98
5.2.1	Procédure d'exécution d'une fonction étendue	99
5.3	Opérations et événements	99
5.4	Chaîne de traitement	99
5.5	Application d'une grammaire locale étendue	99
5.5.1	Découpage en <i>tokens</i>	99

5.5.2	Optimisation de la grammaire	101
5.5.3	Construction des sorties	102
6	Extraction d'information : la reconnaissance d'entités nommées dans des textes formels et bruités	105
6.1	Les entités nommées	105
6.1.1	Aperçu général	106
6.1.2	Typologie des entités nommées	107
6.1.3	Difficultés autour de la catégorisation d'entités nommées	108
6.1.4	Conférences liées à la reconnaissance d'entités nommées	111
6.1.5	Métriques d'évaluation	115
6.1.6	Applications	117
6.2	Reconnaissance des entités nommées dans des textes formels et bruités .	120
6.2.1	Textes formels	120
6.2.2	Textes bruités	121
6.2.3	Quelles sont les difficultés autour de la reconnaissance d'entités nommées ?	123
6.3	Au-delà de l'identification des entités nommées	126
6.3.1	Normalisation	126
6.3.2	Validation	128
6.3.3	Enrichissement	128
6.3.4	Résolution	129
6.4	Conclusion	129
7	Grammaires locales étendues : approches pour l'extraction de l'information	131
7.1	Désambiguïsation des catégories grammaticales	131
7.1.1	Construction d'un automate du texte	132
7.2	Analyse sémantique prédicat–argument	134
7.3	Recherche adaptative de motifs dans un dictionnaire électronique	135
7.3.1	Approche naïve	136
7.3.2	Motivation et objectifs	137

7.3.3	Distance d'édition	137
7.3.4	Recherche approximative dans un dictionnaire classique	138
7.3.5	Dictionnaires électroniques du LADL	138
7.3.6	k-suppressions	139
7.4	Tolérance au bruit	143
7.4.1	Grammaires locales et tolérance au bruit	144
7.4.2	Reconnaissance tolérante au bruit à l'aide des grammaires étendues	145
7.4.3	Reconnaissance de dates et tolérance au bruit	147
7.4.4	Reconnaissance d'anthroponymes et classification du genre des prénoms	152
7.4.5	Reconnaissance de toponymes et géolocalisation des adresses d'organisations	155
8	Conclusions	159
A	Notions préliminaires	161
A.1	Ensembles et suites	161
A.2	Fonctions et relations	165
A.3	Graphes et arbres	168
A.4	Alphabets et mots	170
A.5	Langages	172
	Bibliographie	175
	Glossaire	191
	Abréviations	193
	Index	195

Liste des figures

FIGURE 1.1	Reconnaissance d'entités nommées dans un texte journalistique	2
FIGURE 1.2	Reconnaissance d'entités nommées dans un texte océrisé	4
FIGURE 1.3	Reconnaissance de citations bibliographiques dans un brevet d'invention océrisé	5
FIGURE 2.1	Arbre de dérivation d'une phrase engendré par une grammaire formelle	20
FIGURE 2.2	Diagramme d'états-transitions d'un automate fini	23
FIGURE 2.3	Langage reconnu par un automate fini	24
FIGURE 2.4	Diagramme d'états-transitions d'un NFA	27
FIGURE 2.5	Diagramme d'états-transitions d'un ϵ -NFA	27
FIGURE 2.6	Arbre de dérivation d'une phrase engendrée par une grammaire régulière	30
FIGURE 2.7	Arbre de dérivation d'une phrase engendrée par une grammaire non contextuelle	33
FIGURE 2.8	Étiquette de transition d'un automate à pile	35
FIGURE 2.9	Transition non-déterministe d'un automate à pile	35
FIGURE 2.10	Diagramme d'états-transitions d'un automate à pile	36
FIGURE 2.11	Représentation d'un ruban de lecture/écriture d'une Machine de Turing	39
FIGURE 3.1	Exemple d'un réseau de transitions récursif (RTN)	45
FIGURE 3.2	Reconnaissance de relations filiales	46
FIGURE 3.3	Possibles équivalences entre les grammaires locales (LGs), RTNs, transducteurs finis (FSTs) et automates finis (FSAs)	47
FIGURE 3.4	Automate équivalent au graphe 3.9	54

FIGURE 3.5	Automate équivalent au graphe 3.10	54
FIGURE 3.6	Chaîne de traitement pour l'extraction d'information à l'aide des grammaires locales	58
FIGURE 3.7	Méthode LGFINDER pour la génération de grammaires locales	63
FIGURE 4.1	Transition sur une grammaire locale étendue	77
FIGURE 4.2	Appel à un sous-graphe sur une grammaire locale étendue . .	78
FIGURE 4.3	Appel à une fonction sur une grammaire locale étendue	78
FIGURE 4.4	Transition étiquetée en sortie sur une grammaire locale étendue	79
FIGURE 4.5	Chemin sur une grammaire locale étendue	79
FIGURE 4.6	Chemin accepteur sur une grammaire locale étendue	79
FIGURE 4.7	Automate équivalent au graphe 4.1	80
FIGURE 4.8	Automate fini équivalent au graphe 4.3	84
FIGURE 4.9	Analyse d'une transition étiquetée par une fonction étendue par une machine abstraite	89
FIGURE 5.1	Étapes d'application d'une grammaire locale étendue à un texte	99
FIGURE 5.2	Découpage : <i>tokens</i>	100
FIGURE 5.3	Découpage : <i>tokens</i> indexées	100
FIGURE 5.4	Découpage : représentation sous forme d'indices	100
FIGURE 6.1	Message posté sur Twitter exprimant l'avis sur un produit . .	117
FIGURE 6.2	Agent conversationnel Skyscanner pour la recherche de vols . .	119
FIGURE 6.3	Agent conversationnel avec un message contenant de mots inconnus	124
FIGURE 7.1	Ambiguïté dans l'analyse de la phrase <i>Un seul domestique suffisait à le servir</i>	132
FIGURE 7.2	Reconnaissance d'entités nommées à l'aide des grammaires locales étendues	146
FIGURE 7.3	Dates : grammaire locale étendue	150
FIGURE 7.4	Exemple de dates reconnues à l'aide d'une grammaire locale étendue	151
FIGURE 7.5	Évaluation reconnaissance de dates	151
FIGURE 7.6	Anthroponymes : reconnaître un langage plus grand	152

FIGURE 7.7	Anthroponymes : vérifier les noms à l'aide d'une fonction étendue	153
FIGURE 7.8	Anthroponymes : implémentation de la fonction étendue	153
FIGURE 7.9	Exemple de noms reconnus à l'aide d'une grammaire locale étendue	154
FIGURE 7.10	Évaluation I reconnaissance d'anthroponymes, texte original	154
FIGURE 7.11	Évaluation II reconnaissance d'anthroponymes, changement de casse	154
FIGURE 7.12	Toponymes : grammaire locale étendue	155
FIGURE 7.13	Toponymes : analyse du première cas	156
FIGURE 7.14	Toponymes : analyse du deuxième cas	156
FIGURE 7.15	Toponymes : analyse du troisième cas	157
FIGURE 7.16	Toponymes : analyse du quatrième cas	157
FIGURE 7.17	Exemple d'adresses géolocalisées à l'aide d'une grammaire locale étendue	157
FIGURE 7.18	Évaluation géolocalisation des adresses	158
FIGURE 7.19	Comparaison grammaires locale étendue et Pelias	158
FIGURE A.1	Union, intersection et différence de deux ensembles	163
FIGURE A.2	Relation entre un ensemble de mots et des catégories syntaxiques	167
FIGURE A.3	Diagrammes de graphes à 4 nœuds	169

Liste des graphes

GRAPHE 3.1 Reconnaissance de relations filiales	42
GRAPHE 3.2 Appels de sous-graphes pour chaque sous-motif	49
GRAPHE 3.3 Graphe basique de reconnaissance de noms de personne	50
GRAPHE 3.4 Écriture de sorties :balisage de dates.	50
GRAPHE 3.5 Annotation de noms de personne	50
GRAPHE 3.6 Normalisation de dates	51
GRAPHE 3.7 Sous-graphe de normalisation de mois	52
GRAPHE 3.8 Mode morphologique :reconnaissance et étiquetage de formes ver- bales préfixées	52
GRAPHE 3.9 Grammaire locale qui reconnaît $\mathcal{L}_2 = \{a^+b^+c^+\}$	54
GRAPHE 3.10 Grammaire NooJ qui reconnaît $\mathcal{L}_1 = \{a^n b^n c^n \mid n > 0\}$	54
GRAPHE 3.11 Grammaire (simplifiée) pour ajouter des marqueurs prosodiques aux adjectifs (Monnier et al., 2003, p. 1139)	61
GRAPHE 4.1 Grammaire locale étendue pour la reconnaissance des dates	69
GRAPHE 4.2 Calculatrice	71
GRAPHE 4.3 Évaluation des fonctions étendues	83
GRAPHE 5.1 Convention pour appeler une fonction étendue	92
GRAPHE 5.2 Convention pour appeler une fonction étendue avec un nom différent du fichier où elle est définie	93
GRAPHE 5.3 Variable défini par un seul chemin	95
GRAPHE 5.4 Variable redéfinie et passage par valeur	96
GRAPHE 5.5 Variable temporaire et passage par valeur	96
GRAPHE 5.6 Variable composée par plusieurs <i>tokens</i>	96
GRAPHE 5.7 Variable passée par référence	97

GRAPHE 5.8 Exemple de la construction des sorties avec une grammaire locale étendue (ELG)	103
GRAPHE 7.1 Grammaire locale ambiguë pour le cas d'analyse nom ⟨N⟩ ou adjectif ⟨A⟩	132
GRAPHE 7.2 Grammaire locale ambiguë	133
GRAPHE 7.3 Analyse prédicat–argument passif	134
GRAPHE 7.4 Reconnaissance des dates en anglais	148

Liste des tableaux

TABLEAU I	Symboles pour représenter les LGs et les ELGs sous forme de graphes syntaxiques	xxix
TABLEAU II	Masques lexicaux utilisés dans les LGs et ELGs	xxxii
TABLEAU 2.1	Conventions pour représenter une grammaire formelle	18
TABLEAU 2.2	Hierarchie de Chomsky	20
TABLEAU 4.1	Exemples de dates reconnues par différents systèmes et par une ELG	70
TABLEAU 4.2	Conventions pour représenter les symboles des alphabets des ELGs	75
TABLEAU 5.1	Politiques pour la construction des sorties des ELGs	103
TABLEAU 5.2	Construction de sorties pour le graphe 5.8	103
TABLEAU 6.1	Exemple d’annotation embarquée d’entités nommées	106
TABLEAU 6.2	Exemple d’annotation déportée d’entités nommées (<i>stand-off annotation</i>)	106
TABLEAU 6.3	Catégories d’entités nommées dans MUC(1995)	107
TABLEAU 6.4	Typologie d’entités nommées de Paik et al.	108
TABLEAU 6.5	Campagnes menées entre 1996 et 2015 en relation avec la reconnaissance d’entités nommées	109
TABLEAU 6.6	Taches d’évaluation de MUC-3 à MUC-7 d’après Chinchor (1998)	113
TABLEAU 6.7	Répartition en pourcentage des types d’EN pour chaque partie d’ETAPE dans Nouvel (2012)	115
TABLEAU 7.1	Dictionnaires électroniques	138
TABLEAU 7.2	<i>k</i> -suppressions du mot <i>page</i>	140

TABLEAU 7.3	Nombre de k -suppressions pour un mot de longueur n	141
TABLEAU 7.4	Possibles rotations du mot <i>page</i>	142
TABLEAU 7.5	Génération des k -suppressions du mot <i>page</i>	142
TABLEAU 7.6	Types de dates reconnues par le graphe 7.4	148
TABLEAU 7.7	Exemples de dates provenant d'un corpus océrisé	149
TABLEAU A1	Noms pour des uplets de longueurs spécifiques	164
TABLEAU A2	Notations de Bachmann-Landau	167

Notations et symboles

I. Notations générales

La liste suivante présente quelques notations utilisées dans le corps de la thèse.

Lettres latines

\mathbb{N}	L'ensemble des entiers naturels
\mathbb{R}	L'ensemble des nombres réels
\mathbb{R}^+	L'ensemble des nombres réels non-négatifs
\mathbb{R}^n	Espace euclidien
\mathbb{Z}	L'ensemble des entiers relatifs

Lettres grecques

ε	Le mot vide
Σ	Alphabet
Σ_ε	$\Sigma \cup \{\varepsilon\}$

Notations des ensembles

\in	Appartient à
\notin	N'appartient pas à
\emptyset	L'ensemble vide
\subseteq	Sous-ensemble
\subsetneq	Sous-ensemble propre
$ X $	Cardinalité de X
\overline{X}	Complément de X par rapport à U
U	Ensemble plein
\cap	Intersection
\amalg	Ensemble disjoint
\cup	Union

–	Différence
\wp	Ensemble des parties
\times	Produit cartésien
\circ	Concaténation de langages

Opérateurs logiques

\forall	Pour tout
\exists	Il existe au moins un
\nexists	Il n'existe pas
$\neg a$	non a
$a \wedge b$	a et b
$a \vee b$	a ou b
$a \Rightarrow b$	a implique b
$a \Leftrightarrow b$	a équivalent à b

Notations de Bachmann-Landau

\mathcal{O}	Grand O(micron)
Ω	Grand Omega
Θ	Grand Theta
o	Petit o

Notations générales

$\xRightarrow{*}$	dérive de
\Rightarrow	dérive directement de
\mathfrak{X}	Symbole d'échec
\sqcup	Symbole d'un caractère d'espace explicite
$\langle X \rangle$	X est un méta-symbole





Lettres en indice

i	Variable à la position i
n	Nombre de variables

II. Représentation des grammaires locales/étendues

Dans le tableau I, nous présentons les symboles graphiques utilisés pour schématiser les grammaires locales (LGs) et les grammaires locales étendues (ELGs) décrites dans les chapitres 3 et 4 respectivement. Plus précisément, en plus de la représentation des *fonctions étendues*, que nous introduisons dans cette thèse, nous adoptons les conventions popularisées d'abord par INTEX (Silberztein, 1994) et ensuite étendues par UNITEX (Paumier, 2016) pour construire des graphes syntaxiques. Nous représentons alors ces grammaires par des graphes orientés, lus de gauche à droite¹, avec des nœuds étiquetés dénommés « boîtes », dont une est désignée comme boîte de départ² et une autre comme boîte d'arrivée. En outre, lorsqu'il est pertinent, nous spécifions le lien qui existe entre le symbole décrit et la notation par des diagrammes d'états-transitions (c.f 2.2.1).

TABLEAU I – Symboles pour représenter les LGs et les ELGs sous forme de graphes syntaxiques

SYMBOLE	NOM	DESCRIPTION
	Boîte ²	Les étiquettes des transitions de l'automate qui modélise la grammaire sont représentées à l'intérieur (étiquettes d'entrée) ou en dessous (étiquettes de sortie) des rectangles (appelés boîtes).
	Arc	Des arcs relient les boîtes du graphe. Contrairement à la notation par des diagrammes d'états-transitions, les arcs ne représentent pas les transitions de l'automate sous-jacent (déjà symbolisées par les boîtes), mais servent à décrire quelles sont les transitions entrantes (signalées par des arcs dirigés) et sortantes des états implicitement représentés.
	Boîte de départ	L'entrée du graphe (l'état initial de l'automate sous-jacent), communément le premier symbole à gauche, est représenté par un arc direct vers un carré aux contours en gras.
	Boîte d'arrivée	La fin du graphe (l'état final de l'automate sous-jacent), communément le dernier symbole à droite, est noté par un arc direct vers un double carré.

1. En général, un graphe peut être conçu en accord avec le système d'écriture employé. Par exemple, pour les systèmes qui utilisent un sens droite à gauche, tel que l'arabe, il est plus propice de construire des graphes pouvant être lus dans le même sens.

2. Il est commun de parler des *états* d'une grammaire en faisant rapport aux états d'un diagramme d'états-transitions (cf. section 2.2). Dans ce cas, il est possible de dire que les états de la grammaire sont représentés par des carrés et que, mis à part l'état initial et l'état terminal, les autres états, intermédiaires, ne sont pas explicitement affichés. Cependant, afin de permettre une distinction claire entre la représentation par des graphes et celle par des diagrammes d'états-transitions, nous réservons le terme *état* pour ces dernières. De cette manière, pour se référer à un graphe, nous employons les termes de *boîte de départ/arrivée* et non d'*état initial/final*.

TABLEAU I (suite de la page précédente)

SYMBOLE	NOM	DESCRIPTION
σ	Entrée	Les étiquettes d'entrée non vides (σ) des transitions de l'automate sont explicitement affichées à l'intérieur des boîtes.
■	Entrée vide	Les transitions qui sont étiquetées en entrée uniquement par le mot vide epsilon (ϵ) sont symbolisées par un carré noir de taille réduite.
σ γ	Entrée/Sortie	Les étiquettes de sortie γ des transitions sont placées en dessous des boîtes. Elles sont indiquées en gras. Sauf mention contraire, une boîte sans contenu en dessous désigne une transition qui n'a pas d'étiquette de sortie.
■ γ	Entrée vide/Sortie	Les transitions qui sont étiquetées en entrée uniquement par le mot vide (ϵ) et qui cependant ont une étiquette de sortie ($\gamma \in \Gamma$), sont symbolisées avec l'étiquette de sorti placée en dessous du symbole d'entrée vide.
σ_{t_1} σ_{t_n}	Entrées multiples	L'ensemble des étiquettes d'entrée des transitions qui partent d'un état source pour arriver à un même état de destination, soit $\sigma_t = \{\sigma_{t_1}, \dots, \sigma_{t_n}\}$, sont explicitement affichées à l'intérieur des boîtes. La convention graphique consiste à faire correspondre chaque ligne à une nouvelle étiquette d'entrée.
σ	Boucle	Une boucle sur une ou plusieurs boîtes, symbolisée par un arc dirigé les reliant, indique que les étiquettes en entrée, du chemin formé par la boucle, peuvent être reconnues plusieurs fois en séquence.
ϱ	Sous-graphe	Les symboles d'entrée non-terminaux des transitions sont des appels à d'autres graphes appelés sous-graphes (ϱ). Les noms des sous-graphes attachés aux transitions d'une grammaire sont représentés entourés d'un fond gris foncé. Ils sont explicitement affichés à l'intérieur des boîtes.
$\overset{i}{\rightarrow} \sigma \rightarrow \overset{i}{\leftarrow}$	Variable d'entrée	Les contenus des étiquettes d'entrée d'un ensemble de boîtes entourées par des parenthèses est assignée à un registre local du même nom que celui placé en haut de chaque parenthèse, soit $i = \sigma_{(i..i)}$.
$\overset{\leftarrow}{\sigma} \rightarrow \overset{\leftarrow}{\leftarrow}$	Variable de sortie	Le contenu des étiquettes de sortie entourées par des parenthèses est assignée à un registre local du même nom que celui placé en bas de chaque parenthèse, soit $o = \gamma_{(o..o)}$.

TABLEAU I (suite de la page précédente)

SYMBOLE	NOM	DESCRIPTION
$\leftrightarrow \sigma \rightarrow$	Mode morphologique	Un ensemble de boîtes délimitées par un chevron ouvrant et un chevron fermant se trouve en <i>mode morphologique</i> . Le mode morphologique affecte l'unité minimale d'analyse des étiquettes d'entrée, tandis que par défaut l'analyse se réalise en prenant chaque étiquette d'entrée comme une unité indivisible, lorsque la grammaire se trouve en mode morphologique, l'analyse des étiquettes d'entrée s'effectue lettre par lettre.
$* \rightarrow \sigma$	Contexte gauche	Un contexte gauche, représenté par une étoile, indique que pour un chemin réussi, la sortie du graphe sera produite en tenant juste en compte les entrées et sorties des boîtes qui se trouvent à gauche de l'étoile. Autrement dit, la partie à droite de la séquence reconnue ne sera pris en compte dans le résultat.
$[\rightarrow \sigma \rightarrow]$	Contexte droit	Un ensemble de boîtes délimitées par un crochet ouvrant et un crochet fermant représente un contexte droit. Lorsque la grammaire se trouve dans un contexte droit les symboles d'entrée sont lus mais pas <i>consommés</i> . Si la lecture ne peut pas atteindre la fin du contexte, le chemin n'est pas réussi.
$![\rightarrow \sigma \rightarrow]$	Contexte droit négatif	Un ensemble de boîtes délimitées par un point d'exclamation immédiatement suivi d'un crochet ouvrant et un crochet fermant représente un contexte droit négatif. Lorsque la grammaire se trouve dans un contexte droit négatif les symboles d'entrée sont lus mais pas <i>consommés</i> . Si la lecture peut atteindre la fin du contexte, le chemin n'est pas réussi.
σ $\varphi(\Delta)$	Sortie étendue	Les symboles de sortie non-terminaux des transitions sont des appels à des fonctions externes à la grammaire que nous appelons <i>fonctions étendues</i> et qui ont la forme $\varphi(\Delta)$, où φ est le nom de la fonction et Δ un n -uplet d'arguments en entrée. Les appels aux fonctions étendues attachés aux transitions d'une grammaire sont placés en dessous des boîtes et entourés d'un fond gris foncé.

III. Masques lexicaux utilisés dans les grammaires locales/étendues

Un masque lexical représente une classe de symboles qui reconnaît une ou plusieurs suites d'unités minimales d'analyse, par exemple des *tokens*¹. Le tableau II résume les masques lexicaux utilisés dans la construction des grammaires locales/étendues. Ces masques lexicaux² sont ceux implémentés dans UNITEX (Paumier, 2016) pour la construction des graphes syntaxiques. Le masque # et ceux écrits en majuscules sont standard et correspondent à des méta-symboles. Les autres masques, en minuscules, peuvent prendre différentes valeurs, ainsi que se combiner, afin de faire référence aux informations contenues dans les dictionnaires du texte et établir des contraintes.

TABLEAU II – Masques lexicaux utilisés dans les LGs et ELGs

MASQUE	DESCRIPTION
<CDIC>	N'importe quel mot composé reconnu par un dictionnaire appliqué au texte
< $c_1 \sim c_2$ >	Exclusion, c_1 comme code mais pas c_2
<.c>	c comme code grammatical
<+c>	c comme code sémantique
<DIC>	N'importe quel mot reconnu par un dictionnaire appliqué au texte
<E>	Le mot vide ε
<FIRST>	Premier symbole en capital, n'importe quel token commençant par une majuscule
#	Interdit la présence de l'espace
<m><r>	Masque avec un filtre morphologique, le masque m vérifie r , une expression régulière
<LOWER>	Mot en minuscule, n'importe quel token formé de lettres minuscules
<NB>	Chiffres, reconnaît n'importe quelle suite de chiffres (0-9) contigus
<SDIC>	N'importe quel mot simple reconnu par un dictionnaire appliqué au texte
<TDIC>	N'importe quel token étiqueté sous la forme $\{w, l.c\}$
<TOKEN>	Masque universel, n'importe quel token sauf l'espace et le marqueur {STOP}
<UPPER>	Mot en majuscules, n'importe quel token formé de lettres majuscules
<w.c>	w comme forme canonique et c comme code grammatical

1. Par défaut, un *token* est l'unité minimale qui est pris en compte lors de l'analyse syntaxique réalisée par une grammaire locale/étendue, cependant, lorsqu'une partie de l'analyse se déroule dans le *mode morphologique*, cette unité minimale devient chacune des lettres qui composent un *token*.

2. Observons qu'il est aussi possible de faire référence au complément de certains masques lexicaux en utilisant le symbole ! comme préfixe du nom du masque lexical, par exemple, <!DIC>, permet d'obtenir les mots inconnus des dictionnaires appliqués au texte.

TABLEAU II (*suite de la page précédente*)

MASQUE	DESCRIPTION
$\langle w:f_1:\dots:f_n \rangle$	w comme forme canonique et $f_1 \vee \dots \vee f_n$ comme code flexionnel
$\langle w:f \rangle$	w comme forme canonique et f comme code flexionnel
$\langle w,l,c \rangle$	w : forme canonique, l : forme fléchée et c : code grammatical
$\langle \text{WORD} \rangle$	Mot, n'importe quel token formé de lettres
$\langle w. \rangle$	w comme forme canonique
$\langle w \rangle$	w comme forme canonique, code grammatical ou sémantique

1

Introduction

La prolifération des données numériques que connaissent les sociétés contemporaines a suscité la construction de nouveaux espaces de questionnements et de développements scientifiques et techniques souvent regroupés sous l'appellation de *science des données*. Les méthodes, les outils et les concepts qui émergent dans cet espace bouleversent à la fois les sciences exactes et les sciences humaines et sociales.

Au-delà des controverses épistémologiques soulevées par l'utilisation de ces nouvelles ressources informationnelles et computationnelles, l'essor de cette science se heurte à trois difficultés majeures. Ces trois difficultés sont généralement désignées par les « *trois V* » : – Volume : la quantité de données, – Vitesse : la célérité et la fréquence de création et de mise à jour des données, – Variété : le nombre, le type et l'origine des sources, qui combinent des informations sous différents formats : texte, images, audio, vidéo, contenu multimédia, etc., issues de bases de données, de médias sociaux ou de textes au format brut.

Outre les enjeux liés à la croissance vertigineuse du volume de données, un des principaux défis à relever consiste à transformer ces données en information. Cependant, cette tâche est extrêmement délicate du fait que les données sont, pour la plupart, de nature non structurée, c'est-à-dire, constituées de texte brut écrit en langage naturel. Rappelons qu'à l'inverse des langages artificiels, les langages naturels sont très expressifs et bien connus pour avoir des ambiguïtés et des usages pragmatiques. Dans l'écrit, cette situation est complexifiée par la possible présence de « *bruit* », autrement dit, de perturbations qui empêchent la transmission ou la compréhension de ce qui est écrit, par exemple provenant de problèmes d'encodage, d'erreurs typographiques, de dérogations aux règles syntaxiques ou grammaticales.

Pour parvenir à traiter les données textuelles, des méthodes et des outils informatiques de traitement automatique des langues (TAL) sont mis en œuvre pour identifier et extraire des sous-ensembles pertinents de données ou des connaissances spécifiques. Prenons comme exemple la reconnaissance d'entités nommées (NER), une des sous-tâches de l'extraction de l'information, qui vise à identifier des mots ou des groupes de mots appartenant à une catégorie textuelle ou sémantique spécifique, par exemple (cf. figure 1.1) : de personnes (*Francois Hollande*), de lieux (*Guinea, France*), d'institutions (*World Health Organization*), aussi bien que d'expressions temporelles (dates, durées, etc.), d'expressions numériques (prix, distance, masse, etc), etc.

French President **FUNC** Francois Hollande **PERS** is visiting Guinea **LIEU**, on the first trip by a Western leader to a country at the centre of the latest Ebola outbreak. After arriving in the capital, he told his hosts France **LIEU** had a “duty to support you” in the fight against the virus. More than 1,200 people have died of Ebola in Guinea **LIEU**, which is to trial a test to diagnose Ebola. The outbreak was now “stable” in the West African country, the World Health Organization **ORG** (WHO) said last week.

FIGURE 1.1 – Reconnaissance d'entités nommées dans un texte journalistique ¹

Pour réaliser une telle tâche, le TAL fait appel à la linguistique computationnelle et à l'intelligence artificielle. De façon générale, nous pouvons différencier deux grandes approches : celles des modèles issus de l'ingénierie des grammaires, aussi appelées méthodes symboliques, fondées sur des règles ou bien basées sur des connaissances ; et celles des modèles statistiques, communément associées à l'apprentissage automatique.

L'unité d'analyse de chacune de ces approches est bien différente, la structure de la phrase pour les méthodes symboliques, le voisinage des mots (1, 2, 3 mots) pour les méthodes d'apprentissage automatique. La façon de construire chaque modèle est aussi différente. D'un côté, les méthodes d'apprentissage automatique exigent de larges volumes de données étiquetées pour réaliser l'*apprentissage*, de l'autre côté, les méthodes d'ingénierie des grammaires nécessitent des ingénieurs linguistes pour développer les grammaires. Bien qu'il existe des techniques pour automatiser leur construction, il est indispensable d'avoir un savoir-faire pour maintenir les grammaires ainsi que les autres ressources linguistiques qu'elles mobilisent.

Si les données d'apprentissage sont déjà disponibles, le principal avantage des modèles statistiques est la rapidité de leur mise en place ainsi qu'un haut rappel. En revanche, ils sont difficiles à déboguer et les résultats ne sont pas au rendez-vous dans des textes courts ou lorsque le domaine d'évaluation diffère de celui utilisé pour créer le modèle. En revanche, les grammaires sont propices pour extraire de l'information à un niveau très fin et ont une haute précision. Par contre, leur niveau de rappel est modéré et leur performance est liée aux compétences du développeur linguiste.

1. « Ebola crisis : French President Hollande visits Guinea », 28 novembre 2014, <http://www.bbc.com/news/world-africa-30241374>

Mise à part ces deux types d'approches, il existe des modèles dits *hybrides* de part leur capacité à combiner tant les méthodes symboliques que les méthodes statistiques. Nous pouvons caractériser les types de systèmes hybrides selon trois modes : un *mode pipeline*, où l'entrée d'un module, normalement celui d'apprentissage, repose sur la sortie d'un autre, communément celui à base de règles ; ou bien un *mode tâche*, où des modules spécialisés implémentent une approche spécifique pour reconnaître ou traiter un type défini d'entité ; et finalement, un *mode réseau*, combinant les deux modes précédents.

Il est aussi possible de cataloguer les systèmes hybrides en raison de l'unité minimale utilisée pour faire communiquer les modules. On distingue, d'une part, ceux qui prennent en compte la totalité de l'entrée (chaque module réalise un traitement prédéfini sur son entrée et communique ensuite ces résultats au suivant), et d'autre part, ceux qui peuvent ignorer une partie de l'entrée. Dans ce cas, chaque module réalise un traitement sur la totalité ou sur une partie de l'entrée. Nous remarquerons que la plupart des systèmes hybrides pour la reconnaissance d'entités nommées fonctionnent dans un mode pipeline en effectuant dans chaque module un traitement sur l'intégralité de l'entrée.

Une fois mise en évidence le besoin grandissant de transformer les données textuelles en information, et après avoir présenté un aperçu des techniques issues du TAL pour traiter la problématique de la reconnaissance d'entités nommées, il est important de se poser deux questions : premièrement, pouvons-nous considérer ce problème comme étant résolu par l'utilisation des approches courantes ? deuxièmement, la tâche d'extraction de l'information s'arrête-elle à l'identification des motifs appartenant à une catégorie textuelle ou sémantique spécifique ? La réflexion autour de ces deux questions nous amène à aborder la motivation de notre travail.

1.1 Motivation

Il existe un consensus général pour dire que les techniques proposées dans l'état de l'art pour reconnaître des entités nommées, spécialement celles fondées sur l'apprentissage automatique, produisent des performances proches de la reconnaissance humaine. Cette croyance s'appuie sur les bons résultats des systèmes dans des campagnes d'évaluation et des manifestations scientifiques entreprises au cours des dernières décennies. Cependant, dans la pratique, la réalité est tout autre, prenons le cas de données qui ne sont pas bien formées, par exemple provenant d'un document ocrisé comme celui illustré dans la figure 1.2.

Il n'est pas difficile d'imaginer que dans un texte contenant du bruit, les performances d'un système de reconnaissance, dans l'état de l'art, sont dégradées. En outre, il n'est pas nécessaire que la source du bruit ait pour origine un processus complexe. Il suffit d'une altération de la casse standard, par exemple lors du traitement d'un texte en minuscules, tel que celui issu d'un échange dans des médias sociaux, pour obtenir des résultats moins significatifs. Pouvons alors dire que la reconnaissance d'entités nommées est un problème résolu ?

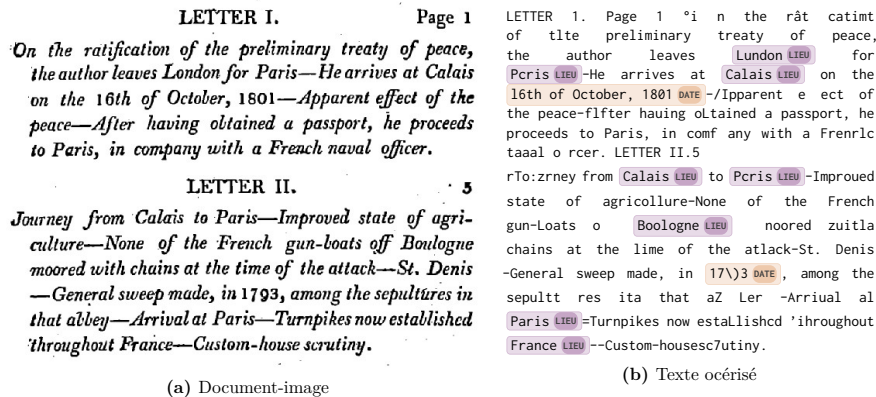


FIGURE 1.2 – Reconnaissance d’entités nommées dans un texte OCRisé

Considérons un argument supplémentaire. Sans tenir compte du fait que les évaluations des systèmes sont, pour la plupart, réalisés sur des textes bien formés, il s’agit d’avoir une réflexion concernant les types d’entités nommées visées. En effet, la tâche est fréquemment reléguée à l’identification des types cités plus haut : personnes, lieux, organisations, dates. Cependant, dans la pratique, les classes d’entités à extraire dépendent directement du besoin. Ce besoin est orienté par la réponse à la question : *dans quel but les entités sont-elles extraites ?* Citons quelques exemples :

- S’il s’agit de caractériser les relations entre la science et la technologie, alors il est important de reconnaître des citations bibliographiques dans des brevets d’invention (cf. figure 1.3).
- S’il est nécessaire de suivre un fait divers, alors il est utile d’identifier des événements dans des textes journalistiques.
- S’il est envisagé de répondre à un appel d’offres, alors il est nécessaire d’obtenir des informations concernant les types d’ouvrage et les budgets prévisionnels.
- S’il est requis de nourrir une base de données d’aide au diagnostic, alors il est primordial d’extraire les noms de symptômes et des médicaments dans un échange médecin/patient.

Les exemples précités nous permettent d’avoir un panorama élargi de la reconnaissance d’entités nommées ainsi que de l’importance de connaître dans quel but les entités sont extraites, nous pouvons affirmer alors que la tâche est plus complexe qu’envisagée. Dans ce contexte, pour être en mesure de parvenir à la reconnaissance, il est impératif de mettre en œuvre des approches qui prennent en compte tant les ambiguïtés que les usages pragmatiques du langage et, en même temps, qui soient capables d’agir en présence de bruit. Néanmoins, est-ce que surmonter ces défis est suffisant pour avoir de l’information pertinente ?

[0002] The cancer suppressor protein p53 is known to be stabilized and activated by diverse cellular stresses such as heat shock, hypoxia, osmotic shock, and DNA damage, leading to the inhibition of cell growth and apoptosis (Ko and Prives, *Genes Dev.* 10: 1054-1072, 1996; Levine, *Cell* 88: 323-331, 1997; Oren, *Cancer Biol.* 5: 221-227, 1994). However, almost nothing is known as to how p53 selects the pathway of cell growth inhibition or apoptosis.

[0003] It has been also known that apoptosis and cell cycle arrest are the major tumor suppressing function of p53 (Levine, *Cell* 88: 323-331, 1997). Therefore, the elucidation of mechanism of p53-induced apoptosis may contribute to the development of anticancer agents having a novel mechanism of action in the cancer treatment. Induction of apoptosis due to the expression of exogenous p53 has been reported in some, though not all, cancer cells with the p53 mutation (Gomez-Manzano et al., *Cancer Res.* 56: 694-699; Kock et al., *Int. J. Cancer* 67: 808-815, 1996). There-

(a) Document-image

[0002] The cancer suppressor protein p53 is known to be stabilized and activated by diverse cellular stresses such as heat shock, hypoxia, osmotic shock, and DNA damage, leading to the inhibition of cell growth and apoptosis (Ko and Prives, *Genes Dev.* 10: 1054-1072, 1996; Levine, *Cell* 88: 323-331, 1997; Oren, *Cancer Biol.* 5: 221-227, 1994). (...) [0003] It has been also known that apoptosis and cell cycle arrest are the major tumor suppressing function of p53 (Levine, *Cell* 88: 323-331, 1997). Therefore, the elucidation of mechanism of p53-induced apoptosis (...) p53 has been reported in some, though not all, cancer cells with the p53 mutation (Gomez-Manzano et al., *Cancer Res.* 56: 694-699; Kock et al., *Int. J. Cancer* 67: 808-815, 1996). There-

(b) Texte océrisé

FIGURE 1.3 – Reconnaissance de citations bibliographiques dans un brevet d'invention océrisé

Malheureusement la réponse est non, les besoins d'extraction d'information vont plus loin que la simple identification des entités reconnues. Énumérons quelques tâches supplémentaires qui peuvent s'avérer nécessaires :

- **Normalisation** : Consiste à associer une forme de référence commune à toutes les variantes d'une entité.
- **Enrichissement** : Consiste à ajouter à l'entité des informations issues de ressources externes ou internes ou même de données issues du texte mais éloignées de l'entité analysée.
- **Validation** : Consiste à vérifier toute question relative à l'entité extraite. Une telle question peut aller de l'existence effective ou potentielle de l'entité en passant par la validation de ses propriétés.
- **Résolution** : Consiste à rapprocher une entité d'une autre référence pertinente. Comme dans le cas de la validation, la référence peut venir d'une ressource endogène ou exogène.

Ainsi l'extraction d'information vue comme une tâche impliquant non simplement la reconnaissance d'entités nommées, mais aussi leur normalisation, enrichissement, validation et résolution, est loin d'être triviale, notamment quand elle vise à traiter des données comportant du bruit. La réflexion autour de la manière de relever ces défis dans un cadre de travail unifié constitue la motivation de notre travail de recherche.

1.2 Problématique de la thèse

Pour délimiter notre problématique, nous précisons tout d’abord la définition de la notion de grammaire locale, ensuite nous donnons un aperçu des limites des grammaires locales, finalement nous nous questionnons sur la faisabilité de surmonter certaines de ces limites.

1.2.1 Grammaire locales : définition utilisée

Les grammaires locales (LGs) (Gross, 1993, 1997) sont un formalisme de description de règles syntaxiques ou sémantiques. Depuis leur conception, leur pertinence a été prouvée pour traiter plusieurs problèmes du TAL liés à la représentation succincte de descriptions linguistiques fines : délimitation des phrases, traitements morphologiques, étiquetage, analyse syntaxique de surface, analyses transformationnelles, résolution des ambiguïtés, extraction de l’information, entre autres (Nam et Choi, 1997, Friburger, 2002, Constant, 2003, Traboulsi, 2004, Laporte, 2005, Geierhos et al., 2008, Martineau et al., 2011, Krstev et al., 2013, Ezzat, 2014).

Le formalisme de LGs est proche de celui des automates à états finis et plus largement associée à des réseaux de transitions récursifs (RTNs) (Woods, 1970, Bates, 1978) et des RTNs comportant des sorties (Sastre et Forcada, 2009). Bien qu’il soit aussi fréquemment assimilé au concept de transducteur fini, de part la capacité d’une grammaire locale à gérer des sorties, observons que même si la notion de sortie est comparable à celle de transduction, les grammaires locales et les transducteurs ne sont pas toujours équivalents. En effet, comme souligné par Blanc (2006, p. 62), l’application d’un transducteur sur un texte en entrée produit comme résultat un nouveau texte composé par la concaténation des sorties, tandis qu’avec une grammaire locale, le résultat peut être en plus égal au texte en entrée combiné avec les séquences de sortie.

En général dans la littérature, la notion de grammaire locale renvoie à des définitions et des mises en œuvre différentes. D’une part, par la variété de types de grammaires locales concernées, ces type peuvent être divisés selon l’analyse effectuée : lexicale (flexion, prétraitement), syntaxique (locale ou structurelle) ou transformationnelle. D’autre part, par les appellations utilisées pour y faire référence, entre autres : transducteurs finis (Fairen et Watrin, 2003, Sætre, 2003, Ranchhod et al., 2004, Kevers, 2006), automates finis (Traboulsi, 2005), automates lexicaux (Blanc et Dister, 2004), automates syntaxiques (Monnier et al., 2003). Finalement, la notion de grammaire locale dépend aussi de l’outil informatique utilisé, nous pouvons distinguer deux groupes :

- Ceux qui ne disposent pas de la capacité de concevoir des grammaires locales à l’aide d’une interface graphique, mais sous la forme d’ensembles de règles de dérivation ou de tableaux de transition d’état. Nous citons par exemple, les grammaires locales du type dag2dag (Sagot et Boullier, 2008) de SXPipe (Sagot et Boullier, 2005), exprimées dans un langage proche de la forme de Backus–Naur (BNF) et analysées à l’aide du système SYNTAX (Boullier et Deschamp, 1991). De plus, des outils comme

OPENFST (Allauzen et al., 2007), pouvant manipuler des RTNs aplatis¹ (Allauzen et Riley, 2012).

- Ceux qui offrent la possibilité de développer des grammaires locales à l'aide d'interfaces graphiques, en faisant appel à l'utilisation de ressources linguistiques, comme des dictionnaires électroniques (Maurel, 1993, Courtois et al., 1997, Chrobot et al., 1999) qui décrivent la syntaxe et la morphologie des entrées lexicales et de leurs attributs sémantiques. Ainsi que d'avoir des sorties, d'utiliser des variables, c'est-à-dire, des registres qui stockent des sous-séquences d'une séquence en entrée, et d'établir de contraintes sur les variables, sous la forme d'opérateurs primitifs. Nous énumérons, par exemple, INTEX (Silberztein, 1994), UNITEX (Paumier, 2003a), NOOJ (Silberztein et Tutin, 2005) ou OUTILEX (Blanc et al., 2006).

Dans notre recherche une grammaire locale concerne des grammaires non contextuelles (cf. sous-section 2.4.1) qui décrivent des langues naturelles. Elles sont représentées par des graphes orientés *lexicalisés* et récursifs. En plus, les grammaires locales peuvent être dérécursivées (cf. section 4.5) et modélisés par des machines à états finis.

Les grammaires locales sont aussi enrichies par des registres et des contraintes qui ne font pas partie du modèle des automates finis, mais en vue de traiter certaines opérations classiques des automates peuvent, avec des ajustements, être considérées comme des symboles d'entrée dépourvus de sens².

Finalement, bien que les grammaires locales soient restreintes à décrire des langages non contextuels (cf. section 2.4), la possibilité d'ajouter des sorties et de contraintes leur donne la capacité de reconnaître des langages contextuels (cf. section 3.5.8) et de réaliser des opérations transformationnelles.

1.2.2 Grammaires locales : leurs limites

Plusieurs auteurs ont évoqué les limites d'utilisation, en termes d'ergonomie, performance et qualité, des grammaires locales.

Selon Friburger (2002), la principale difficulté découle de l'incomplétude des grammaires locales, des ambiguïtés et de l'absence éventuelle de contextes dans les phrases. L'incomplétude des grammaires est relevée aussi par Nakamura (2004) où il est question d'insertions imprévisibles (de motifs) qui ne sont pas prises en compte dans les graphes.

Dister et al. (2004) fait un même constat relatif à la robustesse des grammaires locales, autrement dit, leur capacité à traiter des entrées dégradées ou non standard.

1. L'aplatissement ou la dérécursivation d'un RTN fait référence au processus de remplacement de transitions qui désignent l'appel à un sous-réseau. Afin d'obtenir un automate fini équivalent, il est nécessaire de substituer les transitions étiquetées par des symboles non-terminaux, par une copie exclusive du sous-réseau appelé.

2. Par exemple, un registre qui fait référence au nom d'une variable est considéré comme un symbole d'entrée terminale. C'est lors de l'analyse syntaxique que l'algorithme en charge donne le sens au symbole. Par exemple, en associant un registre en mémoire au prochain symbole lu.

Elle soulève, par exemple, que certains motifs ne sont pas reconnus « *tout simplement parce qu'ils sont mal orthographiés...* », ou comme souligné par [Fairon et Watrin \(2003\)](#), dans le cas des patronymes, simplement parce que les motifs à reconnaître présentent beaucoup de variations.

Le problème des contextes est également relevé dans les travaux de [Sætre \(2003\)](#) relatifs au langage microbiologique, où il est complexe de décrire de motifs contenant de longues séquences. Pour sa part, [Kevers \(2006\)](#) se questionne sur la possibilité d'utiliser des contextes porteurs d'informations mais qui sont éparpillés dans toute une phrase ou paragraphe.

[Constant \(2003\)](#) souligne les difficultés rencontrées pour construire et maintenir des grammaires précises et complètes. La maintenance des graphes est aussi évoquée dans [Pirovani et Silva de Oliveira \(2015\)](#), qui se questionne sur la façon de créer, adapter et transformer des règles dans des graphes afin d'augmenter la performance de la reconnaissance. Par ailleurs, [Ezzat \(2014\)](#) insiste sur le coût de développement des ressources linguistiques nécessaires pour obtenir une couverture satisfaisante.

En outre, pour [Gasiglia \(2004\)](#) le problème réside en « *l'incapacité de discriminer les occurrences verbales des occurrences nominales* », autrement dit, de traiter les ambiguïtés des catégories morphosyntaxiques des entrées lexicales. Selon [Fairon \(1998\)](#), [Calberg \(2003\)](#), [Poibeau et al. \(2003\)](#) il est aussi question des tâches de prétraitement, par exemple, il est indispensable de convertir du HTML en TXT ou prétraiter les textes pour avoir un résultat exploitable à travers les grammaires locales.

Dans le même sens, certaines tâches exigent des modules de post-traitement des résultats, dans [Monnier et al. \(2003\)](#), pour remplacer des marqueurs insérés par les grammaires par des paramètres acoustiques définis dans un tableau. Dans [Kogkitsidou et Antoniadis \(2016\)](#) pour réaliser des transformations au niveau des caractères sur des marqueurs insérés dans des messages SMS.

Le recours à des modules supplémentaires fait aussi partie des approches *hybrides* où la sortie de la grammaire représente l'entrée d'un module fondé sur l'apprentissage automatique : [Watrin et al. \(2014\)](#), [Boujelben et al. \(2014\)](#), [Hkiri et al. \(2016\)](#)

Comme conséquence directe ou indirecte de ces limites, les tâches d'extraction de l'information à l'aide des grammaires locales, telle que la reconnaissance d'entités nommées, bute sur un plafond de performance. Il est devenu coûteux d'améliorer le taux de rappel des systèmes existants, c'est-à-dire le pourcentage des occurrences trouvées parmi les occurrences pertinentes, en ajoutant des règles supplémentaires aux grammaires locales ou en enrichissant les ressources linguistiques qu'elles mobilisent.

En d'autres termes, il est difficile d'augmenter la couverture d'une grammaire locale pour capturer davantage de combinaisons de motifs ou d'accroître les possibilités de traitement des variations attendues et inattendues dans un texte. Ceci résulte du fait de l'apparition des mots inconnus, ou des mots composés, pas nécessairement formés de mots inconnus ([Maurel, 2004](#)), ou mal orthographiés ou encore dérogeant aux règles syntaxiques établies.

1.2.3 Grammaire locales : au-delà des limites

Comme nous venons d'évoquer, les grammaires locales, plus précisément celles assimilées à des graphes syntaxiques, tels qu'ils sont implémentés dans des logiciels comme Intex et NooJ (Silberztein, 1993, 2004), ou Unitex (Paumier, 2003a, 2016), peuvent être utilisées pour la recherche et l'extraction de motifs dans un texte. Néanmoins :

- il est laborieux de construire des grammaires locales capables de capturer toutes les combinaisons possibles de motifs,
- il est parfois impossible de faire face à la complexité du traitement des variations attendues et inattendues : mots inconnus, mots composés inconnus, omis ou mal orthographiés, violations des règles de grammaire standard, etc.

Ainsi en restant sur une description de motifs exhaustive, les approches pour l'extraction d'information fondées sur des grammaires locales conservent une haute précision mais se voient pénalisées en rappel. Dans la pratique, les efforts pour augmenter le rappel finissent par complexifier le développement et la maintenance de la grammaire ou par diminuer leur précision. Le plus souvent, le résultat net des changements, calculé par exemple comme le taux de F-mesure, reste négligeable ou, dans le pire des cas, est non significatif. En outre, ce qui est reconnu par la grammaire lors de l'analyse s'avère très restreint, d'une part par leur nature « locale », d'autre part, par les limites intrinsèques des ressources utilisées :

- L'analyse est locale. Il n'est pas possible, par exemple, de connaître au cours de l'analyse syntaxique les motifs qui ont été reconnus auparavant ou les unités lexicales qui viendront plus loin et de permettre ainsi d'agir d'une manière globale.
- En outre, les ressources linguistiques, tels que les dictionnaires électroniques, même s'ils sont construits dans un effort de complétude et d'exhaustivité, ne peuvent pas toujours répondre à tous les besoins de l'analyse. Il est possible, par exemple, qu'un mot simple ou composé ne soit pas disponible dans un dictionnaire ou, même s'il est présent, qu'il soit dépourvu de certaines informations sémantiques indispensables à l'analyse.
- Il est important aussi de considérer qu'il existe des cas où d'autres ressources, potentiellement utiles pour parvenir à l'analyse, peuvent être disponibles sous une forme différente d'un dictionnaire électronique : fichier texte, base de données, ontologies, etc. Actuellement, cette pluralité de ressources se trouve inaccessible à partir des moteurs d'application des grammaires locales et donc il est impossible de les exploiter sans une transformation directe sous forme de graphes ou de dictionnaires, chose qui n'est pas toujours faisable.

Nous avons mentionné jusqu'à présent deux restrictions des grammaires locales : la première est la localité de l'analyse, la deuxième issue du choix des ressources

utilisées : graphes et dictionnaires électroniques. De plus, les grammaires locales ont une capacité limitée à effectuer des opérations sur les variables, autrement dit, sur les registres qui stockent des sous-séquences d'une séquence, en entrée ou en sortie, ainsi que des informations provenant des entrées des dictionnaires. Bien que, en fonction de l'implémentation, certaines catégories d'opérations primitives soient mises à disposition : tests, transformation et comparaison de variables, ils sont conditionnés à agir selon une forme prédéfinie dans l'algorithme en charge d'appliquer la grammaire locale au texte, ils dépendent ainsi directement de l'implémentation de l'analyseur syntaxique.

Même en apportant une réponse aux restrictions annoncées, il est pertinent de constater que les approches pour l'extraction d'information fondées sur la construction des grammaires locales sont aussi confinées au modèle symbolique. Dans ce sens, dans le formalisme des grammaires locales classiques (représentées par des graphes), il n'existe pas de moyens pour permettre de faire appel, au cours de l'analyse, à des méthodes non-symboliques telles que celles des modèles probabilistes afin d'apporter des informations utiles pour l'analyse.

Compte tenu de ces restrictions la question à l'origine de notre travail de thèse est :

Quels mécanismes peuvent être utilisés pour permettre aux grammaires locales d'intégrer efficacement des ressources externes au cours de l'analyse syntaxique ?

Par ressources, nous entendons à la fois des données, des informations, des opérations, des méthodes, et tout autre élément qui peut s'avérer utile pour les besoins de la tâche qui vise à réaliser la grammaire locale.

1.3 Contributions

Les contributions de cette thèse concernent principalement le formalisme des grammaires locales étendues, sa mise en œuvre ainsi que l'introduction et l'étude des approches pour l'extraction de l'information dans des textes bien formés et des textes bruités.

Le formalisme : Nous introduisons la notion de grammaire locale étendue. Il s'agit d'un formalisme capable d'étendre le modèle classique de grammaires locales. D'une part, par l'ajout de fonctions arbitraires à satisfaire. Ces fonctions, appelées *fonctions étendues*, ne sont pas prédéfinies à l'avance¹, sont évaluées en dehors de la grammaire, peuvent créer des registres globaux ainsi que connaître et manipuler les ressources assignées à l'analyse. D'autre part, en fournissant à l'analyseur syntaxique la possibilité de déclencher des événements, par exemple celui de la lecture d'un symbole d'entrée, qui

1. Autrement dit, l'exécution de ces fonctions est fortement indépendante/découplée de l'analyse syntaxique. Par exemple, en cas de modification d'une fonction étendue, il n'est nécessaire de recompiler la grammaire locale que si la modification impacte l'appel de la fonction (son nom, son nombre de paramètres)

peuvent également être traités sous la forme de fonctions étendues. De plus, les fonctions étendues partagent avec l'analyseur syntaxique des registres globaux concernant l'état de l'analyse et des ressources utilisées, informations qui sont accessibles à partir des autres fonctions étendues. L'intérêt du formalisme proposé est multiple :

- Pallier des problèmes similaires à ceux cités dans la motivation, tels que le traitement des motifs non conformes ou comportant du bruit.
- Dépasser les limites imposées par le contexte « local » des analyses en rendant possible la consultation de l'état global de l'analyse, des ressources utilisées, des variables d'entrée/sortie, du texte passé en entrée.
- Exploiter une vaste variété de ressources difficilement utilisables jusqu'à présent à partir des grammaires locales : fichiers texte, base de données, ontologies, etc. Il est envisageable, par exemple, d'utiliser en cours d'analyse une base de données sémantique, comme le WOLF [Sagot et Fišer \(2008, Wordnet Libre du Français\)](#), pour obtenir les relations d'un mot (pour le mot *chanter* \mapsto *bavarder, dialoguer, entonner, siffler, adresser*).
- Combiner dynamiquement des approches d'analyse hétérogènes, telles que celles issues de l'apprentissage automatique. Il est possible, par exemple, d'employer, au cours de l'analyse, un classifieur bayésien naïf pour prédire le genre d'un prénom inconnu (*Alfnso* \mapsto Masculin) par la grammaire.
- Permettre de décrire efficacement des modèles flous de reconnaissance, par exemple, à travers le rapprochement des mots qui sont inconnus par les ressources linguistiques.

La mise en œuvre : Nous mettons en œuvre le formalisme des grammaires locale étendues dans le cadre d'UNITEX. Pour cela nous adaptons l'analyseur syntaxique descendant d'Unitex en rendant la compilation des grammaires et leur application « *consciente* » de la notion de *sortie étendue*, nous apportons des solutions pour garantir l'atomicité d'évaluation des fonctions étendues, nous contribuons à définir le mode et les types d'arguments qui peuvent être utilisées pour appeler une fonction étendue, nous proposons la mise en place de stratégies permettant de communiquer bidirectionnellement à l'interprète des fonctions étendues et au moteur d'analyse.

Nous développons également des bibliothèques de fonctions pour manipuler, à partir d'une fonction étendue, des chaînes de caractères, des listes, des *tokens*, des motifs reconnus, pour créer des masques lexicaux à la volée, pour consulter le texte en entrée, pour extraire des traits des dictionnaires morphologiques. Nous implémentons les mécanismes nécessaires pour déclencher et traiter des événements lors du début et de la fin de l'analyse, lors de la lecture d'un symbole d'entrée, lors du glissement de la fenêtre d'analyse, lors du chargement initial d'une fonction étendue, lors de leur échec, lors de leur déchargement. En outre, nous développons une nouvelle bibliothèque Unicode, compatible avec la version numéro 9 du standard et avec des performances améliorées par rapport à la version mise en place dans UNITEX.

Les approches : Nous proposons des approches pour identifier, normaliser, enrichir, valider et mettre en relation de motifs reconnus dans des textes bien formés et des textes bruités à l'aide des grammaires locales étendues. Nous focalisons sur les techniques de couplage des ressources externes et des méthodes non-symboliques.

Pour traiter des mots simples ou composées qui ne sont pas reconnus par les grammaires et pour les rapprocher d'entrées connues, nous proposons utiliser une structure de la famille des FM-Index contenant les informations grammaticales et sémantiques sur les entrées. La particularité de la structure proposée, en comparaison avec celles utilisées pour encoder les dictionnaires électroniques de formes fléchies (DELAF), est qu'elle permet d'effectuer la recherche adaptative des motifs.

Par adaptatif, nous entendons la capacité d'effectuer des recherches exactes, des recherches approximatives avec une distance d'édition $d \leq k$ ou également des recherches approximatives par collision de clés, les clés étant stockés comme des attributs sémantiques du type clé-valeur, par exemple Soundex=H367 ou bien Unaccent=heterogeneite¹. Bien que les structures utilisées soient bien connues dans la littérature, leur utilisation pour adopter une approche de recherche adaptative de motifs dans un dictionnaire électronique n'avait pas, jusqu'à présent, été abordée.

1.4 Description des chapitres

Les autres chapitres de cette thèse sont organisés comme suit :

Le chapitre 2 étudie les concepts de base sur les grammaires, les automates et les langages formels.

Le chapitre 3 aborde le formalisme des grammaires locales. Nous étudions leur représentation par des graphes, ces principales caractéristiques, ainsi que nous discutons des travaux portant sur l'utilisation des grammaires locales.

Le chapitre 4 introduit les grammaires locales étendues. Nous présentons les principes concernant leur construction, représentation et analyse, donnons des exemples et définissons les caractéristiques pour garantir l'évaluation des fonctions étendues.

Le chapitre 5 présente la mise en œuvre d'un moteur d'analyse textuelle implémentant le formalisme des grammaires locale étendues. Son développement s'inscrit dans le cadre d'UNITEX, un logiciel libre destiné au traitement linguistique de corpus.

Le chapitre 6 est consacré à l'extraction d'information. Nous détaillons les aspects

1. Il s'agit du mot *hétérogénéité* sans accents.

concernant la reconnaissance d'entités nommées dans des textes formels et bruités et discutons des problèmes de normalisation, enrichissement, validation et mise en relation d'entités.

Le chapitre 7 étudie quelques approches pour l'extraction de l'information dans des textes bien formés et des textes bruités. Nous nous focalisons sur le couplage des méthodes non-symboliques, telles que celles fondées sur l'apprentissage automatique, dans la construction de nos grammaires. Nous montrons que les grammaires locales étendues peuvent avoir recours à ces méthodes au cours de leur application à un texte avec une double utilité : traiter des variations inattendues ainsi que permettre la normalisation, l'enrichissement, la validation et la mise en relation des motifs reconnus.

Le chapitre 8 présente les conclusions du travail réalisé, ainsi qu'une perspective de recherches futures sur le sujet.

1.5 Contenu des annexes

L'annexe A passe en revue certaines notions de base concernant les ensembles et les suites, les fonctions et les relations, les graphes et les arbres, ainsi que les alphabets, les mots et les langages.

2

Grammaires, automates et langages formels

Ce chapitre aborde quelques concepts de base sur les grammaires, les automates et les langages formels ¹ qui sont en lien avec les concepts et représentations des grammaires locales et des grammaires locales étendues étudiées à partir des chapitres 3 et 4 de la thèse. En plus des notions préliminaires disponibles dans l'annexe A, nous introduisons succinctement les concepts des grammaires formelles, des langages réguliers et des automates finis, des langages non contextuels et des automates à pile, des langages contextuels ainsi que des langages sans restriction et des Machines de Turing.

2.1 Grammaires formelles

Une **grammaire formelle** est un *formalisme* pour décrire un langage (cf. définition A.34). Cette description permet à la fois de définir, générer et reconnaître ce langage. Nous donnons une définition plus formelle ci-après.

Définition 2.1 (Grammaire formelle). Une **grammaire formelle** est un quadruplet $\mathcal{G} = (V, T, P, S)$, où :

1. Ces concepts sont largement étudiés dans [Baudot \(1987\)](#), [Perrin \(1990\)](#), [Davis et al. \(1994\)](#), [Hopcroft et al. \(2000\)](#), [Sakarovitch \(2009\)](#), [Sipser \(2012\)](#), [Berstel \(2013\)](#), [Linz \(2016\)](#)

1. V est un ensemble non vide et fini de **variables**, aussi appelées symboles *non-terminaux*,
2. T est un ensemble non vide et fini de symboles **terminaux**, tel que $V \cap T = \emptyset$,
3. P est un ensemble fini de **règles de réécriture**, et
4. $S \in V$ est un symbole initial nommé l'**axiome de la grammaire**

Une grammaire formelle \mathcal{G} , dorénavant *grammaire* ou \mathcal{G} , est alors un système fini composé d'un ensemble *disjoint* de symboles de V et de T , ou $V \cup T$ constitue le **vocabulaire de la grammaire**, noté Σ , d'un ensemble fini P de règles de réécriture, ou simplement *règles*, et d'un seul symbole initial $S \in V$. Les règles de P constituent l'élément central de la définition de toute grammaire et sont de la forme :

$$l \rightarrow r \qquad \text{ou :}$$

- l , la *partie gauche* de la règle, est une suite de mots contenant au moins un symbole de V ou de T , soit $l \in (V \cup T)^+$, autrement dit, $l \in \Sigma^+$, et
- r , la *partie droite* de la règle, est une suite de mots contenant des symboles de V ou de T , soit $r \in (V \cup T)^*$, autrement dit, $r \in \Sigma^*$.

La règle $l \rightarrow r$ se lit « l se réécrit r » et signifie que l peut engendrer r . L'ensemble de règles $P = \{(l, r) \mid l \in \Sigma^+, r \in \Sigma^*\}$ permet ainsi, par leur application successive, de réécrire des suites de mots à partir d'autres suites. Selon la façon dont r peut se dériver de l , soit sous forme directe, ou bien sous forme indirecte, on parle d'une *dérivation directe* ou simplement d'une *dérivation*.

Définition 2.2 (Dérivation directe). Si en appliquant, en *une seule* étape, la règle $l \rightarrow r$ à un mot w de la forme ulv avec $u, v \in \Sigma^*$, il est possible d'obtenir un mot z de la forme urv . Dans ce cas, on dit que w « *engendre directement* » z , ou que z « **dérive directement de** » w et on note :

$$w \Longrightarrow z$$

Définition 2.3 (Dérivation). Si en appliquant, en *zéro ou plus* étapes, un ensemble de règles P à un mot w , il est possible d'obtenir une suite de mots $x_0, x_1, x_2, \dots, x_n$ pour $n \geq 0$, tel que :

$$\begin{aligned} w &= x_0, \\ x_0 &\Longrightarrow x_1 \Longrightarrow x_2 \Longrightarrow \dots \Longrightarrow x_n, \text{ et} \\ z &= x_n \end{aligned}$$

On dit que w « engendre » z , ou que z « *dérive de* » w et on note :

$$w \xRightarrow{*} z$$

En particulier, n est appelé la *longueur de la dérivation*. En outre, si $n = 0$ alors $w \xRightarrow{*} w$.

Définition 2.4 (Forme sententielle). Un mot $w \in \Sigma^*$ est appelé **forme sententielle** de \mathcal{G} s'il peut être engendré par une séquence de dérivations de la forme $S \xRightarrow{*} w$, autrement dit, elle se définit comme tout mot w qui se dérive de l'axiome de la grammaire S , le mot en entrée étant aussi une forme sententielle, la première.

Définition 2.5 (Phrase). Un mot w est appelé une **phrase** de \mathcal{G} s'il est une forme sententielle de \mathcal{G} constituée uniquement de symboles terminaux (zéro ou plus), autrement dit, elle se définit comme tout mot $w \in T^*$ qui se dérive de l'axiome de la grammaire S .

Définition 2.6 (Langage d'une grammaire). Si $\mathcal{G} = (V, T, P, S)$ est une grammaire, alors le langage défini par \mathcal{G} , noté $L(\mathcal{G})$, est l'ensemble de phrases ($w \in T^*$) que la grammaire engendre. Plus formellement :

$$L(\mathcal{G}) = \{w \in T^* \mid S \xRightarrow{*} w\}$$

En particulier, deux grammaires \mathcal{G}_1 et \mathcal{G}_2 sont dites *faiblement équivalentes*, noté $L(\mathcal{G}_1) = L(\mathcal{G}_2)$, si elles définissent le même langage. En outre, elles sont appelées **fortement équivalentes**, si, et seulement si, elles sont faiblement équivalentes et utilisent les mêmes dérivations pour engendrer la même phrase.

Définition 2.7 (Grammaire de génération). Si $L(\mathcal{G}_e)$ est le langage défini par \mathcal{G} tel que $L(\mathcal{G}) = \{w \in T^* \mid S \xRightarrow{*} w\}$, alors \mathcal{G} est appelée **grammaire de génération**.

Définition 2.8 (Grammaire de reconnaissance). Si $L(\mathcal{G}_r)$ est le langage défini par \mathcal{G} tel que $L(\mathcal{G}) = \{w \in T^* \mid w \xRightarrow{*} S\}$, alors \mathcal{G} est appelée *grammaire d'analyse* ou **grammaire de reconnaissance**.

Observons que dans une **grammaire de génération** les phrases ($w \in T^*$) sont engendrées en substituant l'axiome de la grammaire S par des symboles à droite et en répétant ce processus jusqu'à avoir uniquement des symboles terminaux ($S \xRightarrow{*} w$). En revanche, dans une **grammaire de reconnaissance** les règles sont inversées et de la forme $r \rightarrow l$, de sorte qu'une phrase est *analysée* en effectuant la substitution par les symboles qui se trouvaient initialement à gauche et en répétant ce processus jusqu'à atteindre l'axiome de la grammaire ($w \xRightarrow{*} S$). Bien que l'ensemble des règles d'une grammaire puisse avoir la forme $r \rightarrow l$, dite de *composition*, ou $l \rightarrow r$, dite de *décomposition*, il est utile de remarquer que la grammaire définit toujours le même langage.

Nous pouvons à présent résumer les trois aspects caractéristiques d'une grammaire : définitionnel, génératif et analytique :

1. **Définitionnel** : \mathcal{G} définit L , autrement dit, \mathcal{G} définit les phrases qui appartiennent à L ainsi que leurs structures.
2. **Génératif** : \mathcal{G} peut engendrer L , autrement dit, G permet d'engendrer des phrases de L .
3. **Analytique** : \mathcal{G} peut reconnaître L , autrement dit, G permet de reconnaître si une phrase quelconque appartient ou non à L .

Avant de donner quelques exemples de grammaires formelles, nous précisons les conventions utilisées pour représenter ses éléments.

Notation 2.9 (Éléments d'une grammaire). Pour représenter une grammaire, nous utilisons les conventions résumées dans le tableau 2.1 :

N ^o	ÉLÉMENT	CONVENTION
1	V	A, B, C, \dots
2	V	$\langle ABC \rangle$ est une variable
3	T	a, b, c, \dots
4	Σ	$\alpha, \beta, \gamma, \dots$
5	S	à gauche de la première règle
6	P	règles sans ordre imposé
7	P	$\{A \rightarrow aB, A \rightarrow \varepsilon\} = A \rightarrow aB \varepsilon$

TABLEAU 2.1 – Conventions pour représenter une grammaire formelle

1. Les symboles de V (*variables*) sont notés par des lettres latines majuscules du début de l'alphabet : A, B, C, \dots
2. Les symboles de V entourés par des chevrons sont considérés comme uniques, par exemple $\langle ABC \rangle$ est traité de la même manière que la n'importe quelle autre variable comme A
3. Les symboles de T (*terminaux*) sont notés par des lettres latines minuscules du début de l'alphabet : a, b, c, \dots
4. Les symboles de Σ ($V \cup T$) sont notés par des lettres grecques minuscules : $\alpha, \beta, \gamma, \dots$
5. L'axiome de la grammaire (S) est placé à gauche de la première règle
6. L'ensemble de règles (P) n'est pas ordonné, aucun ordre de définition ne s'impose
7. Les règles qui partagent la même partie gauche peuvent s'écrire sous forme abrégée sur une seule ligne. Pour cela, tous les éléments de la partie droite sont regroupés, puis séparés par des traits verticaux $|$, où chaque trait vertical peut se lire « ou bien ». Plus formellement :

Soit $N = \{(A, r_1), (A, r_2), \dots, (A, r_n)\}$, tel que $N \subseteq P$, alors N peut s'abrégé :

$$A \rightarrow r_1|r_2|\dots|r_n$$

Par exemple, les deux règles : $A \rightarrow aB$ et $A \rightarrow \varepsilon$, sont équivalents à $A \rightarrow aB|\varepsilon$, c'est-à-dire, « A se réécrit aB , ou bien, ε ».

Considérons à présent les exemples suivants.

Exemple 2.10.

$$\mathcal{G}_1 = (V, T, P, S),$$

1. $V : \{S, A, B\}$
2. $T : \{a, b\}$
3. $P :$

S	\rightarrow	A
A	\rightarrow	$aB \mid \varepsilon$
B	\rightarrow	bA

Par l'application de P , il est possible d'engendrer, par exemple, la phrase $abab$:

$$S \Rightarrow A \Rightarrow aB \Rightarrow \mathbf{abA} \Rightarrow abaB \Rightarrow ababA \Rightarrow \mathbf{abab}$$

Ce qui équivaut à écrire :

$$S \xRightarrow{*} abab$$

où :

- abA est une *forme sententielle* ($w \in \Sigma^*$)
- $abab$ est une *phrase* ($w \in T^*$)
- $S \xRightarrow{*} abab$ a une longueur de 6, c'est-à-dire, $abab$ est le résultat de 6 dérivations directes

Finalement, observons que les phrases que \mathcal{G}_1 peut engendrer ont toutes la forme :

ε
ab
$abab$
$ababab$
$abababab$
$ababababab$
$abababababab$

Nous pouvons conclure que \mathcal{G}_1 définit le langage :

$$L(\mathcal{G}_1) = \{(ab)^n \mid n \geq 0\}$$

Définition 2.11 (Variable annulable et variable nulle). Une variable $A \in V$ est dite **annulable** s'il existe une dérivation $A \xRightarrow{*} \varepsilon$. De plus, A est dite **nulle** si toutes ses dérivations engendrent le mot vide.

Notation 2.12 (Arbre de dérivation). Étant donnée une grammaire quelconque, le processus de dérivation d'une phrase peut être visualisé plus facilement à l'aide d'une notation arborescente. Il s'agit de construire un arbre où la racine coïncide avec l'axiome de la grammaire (S), les nœuds avec les variables (V) et les feuilles avec les symboles de la phrase (T). Ce type de représentation est connu sous le nom d'**arbre de dérivation** ou *arbre syntaxique* du fait qu'il représente la structure syntaxique d'une phrase engendrée par une grammaire.

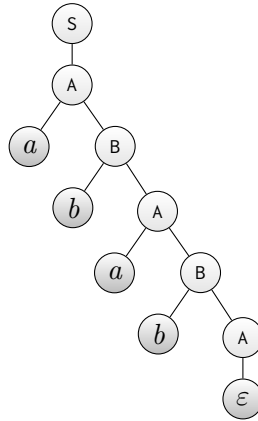


FIGURE 2.1 – Arbre de dérivation de la phrase $abab$ engendré par la grammaire \mathcal{G}_1

Définition 2.13 (Grammaire ambiguë). Une grammaire est dite **ambiguë** lorsqu'il existe plus d'un arbre de dérivation pour une phrase du langage.

Définition 2.14 (Langage ambigu). Un **langage ambigu** est un langage défini par une grammaire ambiguë.

Définition 2.15 (Types de grammaires). Dans les années 1956, Noam Chomsky a proposé une classification des grammaires en 4 catégories, connue désormais comme La **Hiérarchie de Chomsky**, où chaque catégorie a des règles plus restrictives à mesure que le type augmente. La **Hiérarchie de Chomsky** est présentée dans le tableau 2.2

TYPE	GRAMMAIRE	MACHINE
3	Régulière	Automate fini
2	Non contextuelle	Automate à pile
1	Contextuelle	Automate linéairement borné
0	Contextuelle avec effacement	Machine de Turing

TABEAU 2.2 – Hiérarchie de Chomsky

Même si la Hiérarchie de Chomsky est largement utilisée, il est utile de rappeler que mises à part les grammaires régulières (aussi appelées rationnelles), non contextuelles, contextuelles et contextuelles avec effacement (aussi nommées générales ou sans

restriction), il existe d'autres formalismes linguistiques pour décrire de grammaires formelles, citons par exemple :

- La description syntaxique systématique des lexique-grammaires (Gross, 1975)
- Les grammaires lexicales-fonctionnelles (*Lexical functional grammar*, Kaplan et Bresnan, 1982)
- Les grammaires d'arbres adjoints (*Tree-adjoining grammars*, Joshi et Schabes, 1997)
- Les grammaires syntagmatiques guidées par les têtes (*Head-driven phrase structure grammars*, Pollard et Sag, 1994)
- Les grammaires relationnelles (*Relational grammars*, Perlmutter, 1980)
- Les grammaires orientées relationnelles (*Arc pair grammar*, Johnson et Postal, 1981)
- Les grammaires catégorielles avec unification (*categorial unification grammars*, Uszkoreit, 1986)

En outre, en ce qui concerne les automates finis qui reconnaissent chaque type de langage, il est important de dire qu'il est possible de parvenir à reconnaître des langages moins restrictifs (type 0 et 1) à l'aide de langages plus restrictifs (type 2 et 3) et d'opérations qui se trouvent hors du cadre des automates. Par exemple, comme nous les étudions dans le chapitre 3 dédié aux grammaires locales, le fait que *tout langage est inclus dans un langage rationnel* (cf. section 3.5.8) permet à une grammaire locale de mettre en place une approche pour reconnaître des langages contextuels (type 1 ou 0) en reconnaissant d'abord un langage régulier ou non contextuel (type 3 ou 2), et en appliquant par la suite une opération pour exclure les séquences qui n'appartiennent au langage contextuel.

Dans les sections qui suivent nous présentons d'abord les automates finis, ensuite nous étudions les types de langages associés à l'Hierarchie de Chomsky.

2.2 Automates finis

Les automates finis sont des machines abstraites qui vérifient sous forme séquentielle si un mot, passé en entrée, appartient ou non à un langage donné. Ces automates sont dits finis car restreints à un ensemble fini d'états qui mémorisent des informations. Les automates finis ont trouvé des applications directes dans plusieurs domaines qui vont de l'analyse lexicale de langues artificielles et naturelles, en passant par la compression de l'information, la recherche de motifs dans un texte, ou la vérification de systèmes, tel que des circuits électroniques, dotés d'un nombre fini d'états.

2.2.1 Automates finis déterministes

Les automates finis déterministes sont rapides et efficaces pour traiter des problèmes d'analyse des langages tel que la génération automatique de phrases (cf. définition 2.5) ainsi que leur recherche dans des données textuelles. Considérons les phrases :

- (1) *Leonardo was an Italian mathematician*
- (2) ** mathematician an Leonardo Italian was*

Un automate fini \mathcal{A} qui reconnaît un langage quelconque L , est capable alors de dire si les phrases (1) et (2) appartiennent à L (propriété analytique). Dans la suite, nous présentons formellement ces automates ainsi que leur version non-déterministe.

Définition 2.16 (Automate fini déterministe). Un **automate fini déterministe** est un quintuplet $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, où :

1. Q est un ensemble fini d'**états**,
2. Σ est un **alphabet**,
3. $\delta : Q \times \Sigma \longrightarrow Q$ est la **fonction de transition**,
4. $q_0 \in Q$ est l'**état initial**, et
5. $F \subseteq Q$ est l'ensemble des **états finaux**.

Un automate fini \mathcal{A} correspond à un **graphe orienté** (cf. A.16) formé par un ensemble fini d'**états** Q (*les nœuds du graphe*) et de **transitions** (*les arcs du graphe*) qui les relient. Un état est désigné comme **initial** ($q_0 \in Q$) et certains autres sont **finaux** ($F \subseteq Q$). Un automate est à tout instant dans un état unique, la **fonction de transition** δ décrit alors comment l'automate passe à un autre état en lisant un symbole de l'**alphabet**.

Chaque transition est définie comme un triplet (p, σ, q) de $Q \times \Sigma \times Q$ (**l'ensemble de transitions**) où : p est l'**état de départ**, σ est un symbole de l'alphabet, et q est l'**état d'arrivée**. Si \mathcal{A} a une **transition étiquetée** par un symbole σ qui relie p à q , ceci indique que lorsque l'automate est dans l'état p et *lit* σ , il passe à l'état q . Cette **règle de transition** est notée : $\delta(p, \sigma) = q$. Considérons l'exemple 2.17 :

Exemple 2.17.

$$\mathcal{A}_1 = (Q, \Sigma, \delta, q_0, F),$$

1. $Q : \{q_0, q_1, q_2, q_3\}$
2. $\Sigma : \{0, 1\}$

3. δ : est décrite par la **table de transition d'état** suivante :¹

q	$\delta(q, 0)$	$\delta(q, 1)$
q_0 .	q_0	q_1
q_1	q_2	q_1
$q_2,$	q_3	q_3
q_3	q_2	q_2

4. q_0

5. $F : \{q_2\}$

Un automate fini peut être représenté graphiquement par un diagramme **d'états-transitions** comme celui de la figure 2.2. L'état initial, conventionnellement le premier à gauche, est distingué par un arc orienté vers un nœud aux contours en gras. Les états finaux sont indiqués par des nœuds avec double cercle. Les transitions sont des arcs orientés allant d'un nœud de départ vers un nœud d'arrivée. Les étiquettes des transitions se placent sur les arcs. Lorsque un automate a plusieurs transitions² reliant deux états, alors il est possible de placer sur un seul arc l'ensemble des étiquettes sous forme la d'une liste. Pour l'automate \mathcal{A}_1 de l'exemple 2.17, nous avons deux transitions $(q_3, 0, q_2)$ et $(q_3, 1, q_2)$ reliant q_3 à q_2 qui sont représentées par un seul arc étiqueté par 0, 1.

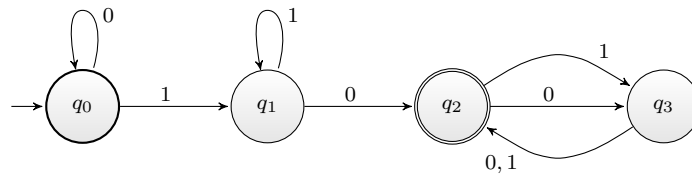


FIGURE 2.2 – Diagramme d'états-transitions d'un automate fini

Définition 2.18 (Chemin sur un automate fini). Un **chemin** sur un automate fini est une suite de transitions menant d'un état p à un état r dont la concaténation des étiquettes successives (**l'étiquette du chemin**) forment un mot $w_i, w_{i+1}, \dots, w_{j-1}, w_j$. Nous notons ceci comme $p \xrightarrow{w_{i,j}} r$. Un **chemin réussi** est nommé ainsi lorsque $p = q_0$ et $r \in F$, c'est-à-dire quand il existe un chemin allant de l'état initial à l'un des états finaux.

Définition 2.19 (Mot accepté par un automate fini). Un mot w de Σ^* est **accepté** par un automate fini \mathcal{A} s'il existe un chemin réussi qui est étiqueté par w , soit $q_0 \xrightarrow{w} r$, tel que $r \in F$.

1. Les tables de transition d'états sont une alternative pour décrire les automates finis. Les lignes représentent les états. L'état initial est marqué par un point après son nom et les états finaux avec une virgule. Les colonnes représentent l'alphabet. Les cellules, c'est-à-dire les intersections ligne/colonne, codent les transitions.

2. Soit un sous-ensemble $T \subseteq Q \times \Sigma \times Q$, tel que $T = \{(p, \sigma, q) \mid p \in Q, \sigma \in \Sigma, q \in Q, \delta(p, \sigma) = q\}$

Définition 2.20 (Langage reconnu par un automate fini). Le **langage reconnu** par un automate fini \mathcal{A} , noté $L(\mathcal{A})$, est égal à l'ensemble des mots acceptés par \mathcal{A} . En particulier, deux automates \mathcal{A}_1 et \mathcal{A}_2 sont dits **équivalents** s'ils reconnaissent le même langage, soit $L(\mathcal{A}_1) = L(\mathcal{A}_2)$. En outre, remarquons qu'un automate fini peut accepter plusieurs mots, mais il reconnaît toujours un seul langage. L'inverse n'est pas vrai, pour un langage donné, il peut exister plusieurs automates qui le reconnaissent.

Considérons l'automate \mathcal{A}_2 de l'exemple 2.21.

Exemple 2.21.

$$\mathcal{A}_2 = (Q, \Sigma, \delta, q_0, F),$$

1. $Q : \{q_n \mid n \leq 15\}$
2. $\Sigma : \{a, c, d, e, i, l, p, r, s\}$
3. $\delta :$

$$\begin{array}{llll} \delta(q_0, p) = q_1 & \delta(q_1, a) = q_2 & \delta(q_2, r) = q_3 & \delta(q_3, a) = q_4 \\ \delta(q_4, d) = q_5 & \delta(q_5, i) = q_6 & \delta(q_6, s) = q_7 & \delta(q_7, e) = q_8 \\ \delta(q_8, s) = q_{11} & \delta(q_7, i) = q_9 & \delta(q_9, c) = q_{13} & \delta(q_{13}, a) = q_{15} \\ \delta(q_{15}, l) = q_{11} & \delta(q_9, a) = q_{14} & \delta(q_{14}, l) = q_{11} & \delta(q_{14}, c) = q_{11} \\ \delta(q_7, a) = q_{10} & \delta(q_{10}, l) = q_{11} & \delta(q_{10}, i) = q_{12} & \delta(q_{12}, c) = q_{11} \end{array}$$

4. q_0
5. $F : \{q_8, q_{11}\}$

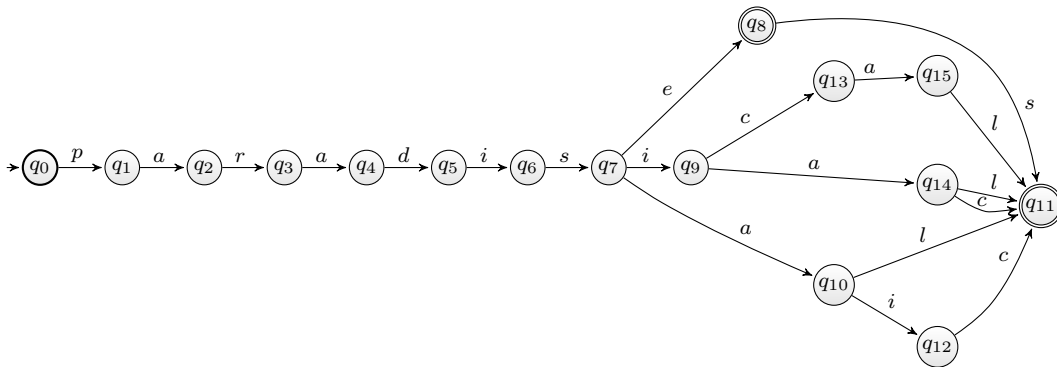


FIGURE 2.3 – Automate fini qui reconnaît le langage $L(\mathcal{A}_2) = \{\text{paradise, paradises, paradisical, paradisial, paradisiac, paradisal, paradisaic}\}$

Quelques exemples de chemins sur l'automate \mathcal{A}_2 sont :

1. $q_0 \xrightarrow{\text{parad}} q_5$
2. $q_2 \xrightarrow{\text{radise}} q_8$

3. $q_0 \xrightarrow{\text{paradise}} q_8$

(1) n'est pas un chemin réussi puisque $q_5 \notin F$, (2) ne l'est pas non-plus car $q_2 \neq q_0$, (3) est un chemin réussi, il part de l'état initial (q_0) et arrive dans un des états finaux $q_8 \in F$. Étant donnée que $q_0 \xrightarrow{\text{paradise}} q_8$ est un chemin réussi, *paradise* est un mot accepté par \mathcal{A}_2 . Les autres mots acceptés sont : *paradises*, *paradisical*, *paradisial*, ..., l'ensemble de mots acceptés par \mathcal{A}_2 , autrement dit, le langage reconnu par \mathcal{A}_2 , est :

$$L(\mathcal{A}_2) = \{\text{paradise}, \text{paradises}, \text{paradisical}, \text{paradisial}, \text{paradisiac}, \text{paradisal}, \text{paradisaic}\}$$

Définition 2.22 (État accessible, co-accessible). Un état $q \in Q$ est dit **accessible** s'il existe un chemin qui part de q_0 et mène à q ; il est dit **co-accessible** s'il existe un chemin qui part de q et mène à r tel que $r \in F$.

Définition 2.23 (Automate émondé). Un automate \mathcal{A} est **émondé** si tous ses états sont accessibles et co-accessibles.

Définition 2.24 (Déterminisme d'un automate). Un automate \mathcal{A} sur un alphabet Σ est un automate fini déterministe (DFA), si et seulement si les deux conditions suivantes sont satisfaites : 1. Il n'existe pas de transitions étiquetées par le mot vide (ε) ; et 2. Pour tout état p , il part vers q au plus une transition étiquetée par chaque symbole $\sigma \in \Sigma$. Les automates des exemples 2.17 et 2.21 sont des DFAs.

Définition 2.25 (Automate fini déterministe complet). Un automate fini déterministe sur un alphabet Σ est un **automate fini déterministe complet** si pour tout état p part une et une seule transition étiquetée de σ vers q . Par exemple, l'automate de l'exemple 2.17 est un DFA complet, celui de l'exemple 2.21 ne l'est pas.

2.2.2 Automates finis non-déterministes

Rappelons qu'un automate fini déterministe est à tout instant dans un état unique et que lorsque il lit un symbole en entrée, il est possible de déterminer avec certitude quel sera l'état suivant. En effet, la fonction de transition d'un DFA est égal à $\delta : Q \times \Sigma \rightarrow Q$, c'est-à-dire, qu'à partir d'un état p et d'un symbole σ dans $Q \times \Sigma$, elle a toujours un seul état q d'arrivée dans Q , soit $\delta(p, \sigma) = q$.

Cependant, lorsqu'à partir d'un symbole donné et d'un état de départ il est possible d'avoir comme destination plusieurs états d'arrivée, ou lorsqu'il est admis de changer d'état sans lire un symbole, ou lorsqu'on a la faculté de choisir quel sera le prochain état, ou de passer dans un autre d'état spontanément, est appelé automate fini non-déterministe (NFA). Au sens informatique, le non-déterminisme peut se comparer à une sorte de parallélisme où pour chaque transition qui comporte plus d'un état d'arrivée, l'automate fini a la capacité de créer simultanément plusieurs copies de lui-même et, pour chaque copie, d'explorer chacun des états d'arrivée possibles.

En outre, pour chaque automate fini non-déterministe il y a un automate fini déterministe équivalent. Autrement dit, tout automate fini non-déterministe peut être converti en un automate fini déterministe équivalent. Ceci est d'un grand intérêt puisque, même si les automates finis non-déterministes sont plus coûteux à utiliser, ils ont un nombre d'états et de transitions réduits, ils sont plus faciles à comprendre et à construire. Dans la suite, nous donnons une définition formelle d'un automate fini non-déterministe.

Définition 2.26 (Automate fini non-déterministe). Un **automate fini non-déterministe** est un quintuplet $\mathcal{N} = (Q, \Sigma, \delta, q_0, F)$, où :

1. Q est un ensemble fini d'états,
2. Σ est un **alphabet**,
3. $\delta : Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)$ est la **fonction de transition**,
4. $q_0 \in Q$ est l'état **initial**, et
5. $F \subseteq Q$ est l'ensemble des **états finaux**.

Les automates finis déterministes et non-déterministes sont similaires à l'exception du type de fonction de transition δ utilisée. Observons que cette fonction de transition se différencie par : 1. l'alphabet d'entrée de la fonction qui n'est pas simplement Σ mais l'union entre cet alphabet et un ensemble ne contenant que le mot vide $\{\varepsilon\}$, noté Σ_ε ¹ ; 2. la sortie de la fonction qui n'est plus constituée par un seul état, mais par un ensemble des parties de Q , noté $\mathcal{P}(Q)$, contenant zéro, un état ou plus. Cela est illustré par l'exemple 2.27.

Exemple 2.27.

$$\mathcal{N}_1 = (Q, \Sigma, \delta, q_0, F),$$

1. $Q : \{q_0, q_1, q_2, q_3\}$
2. $\Sigma : \{0, 1\}$
3. δ : est décrite par la table de transition d'état suivante :

q	$\delta(q, 0)$	$\delta(q, 1)$
q_0 ,	\emptyset	$\{q_1, q_2\}$
q_1 ,	\emptyset	$\{q_1\}$
q_2	$\{q_3\}$	\emptyset
q_3 ,	\emptyset	$\{q_2\}$

4. q_0
5. $F : \{q_0, q_1, q_3\}$

1. Pour éviter toute confusion, il est requis que ε ne soit pas membre de Σ .

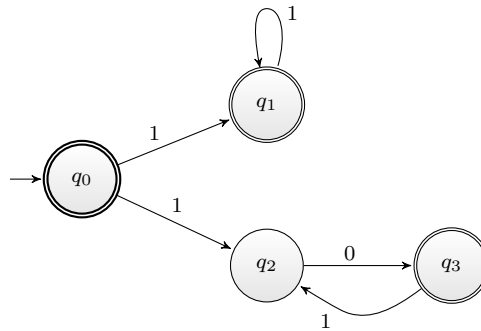


FIGURE 2.4 – Diagramme d'états-transitions d'un NFA

Définition 2.28 (ε -transitions). Les transitions qui font passer un automate fini non-déterministe d'un état dans un autre sans qu'aucune opération de lecture d'un symbole ne soit effectuée, sont nommées des transitions spontanées ou ε -**transitions**. De même, un automate fini non-déterministe avec ε -transitions est abrégé en ε -NFA. Remarquons que ce type de transitions est possible car la fonction de transition accepte $\varepsilon \in \Sigma_\varepsilon$ en entrée.

Exemple 2.29. Un automate équivalent à \mathcal{N}_1 (exemple 2.27) qui utilise des transitions spontanées est donné à figure 2.5.

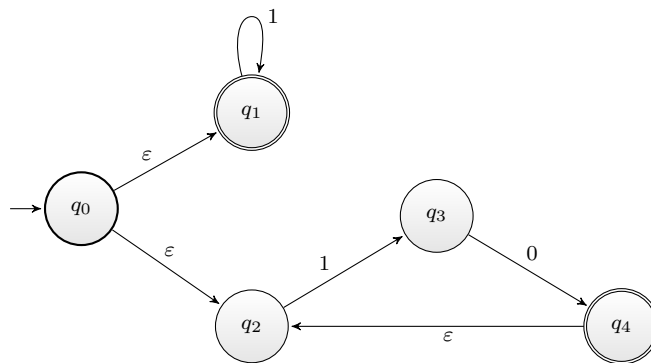


FIGURE 2.5 – Diagramme d'états-transitions d'un ε -NFA

Enfin, il est important de mentionner que tout automate fini non-déterministe a un automate fini déterministe équivalent. La démonstration de ceci est due à [Rabin et Scott \(1959\)](#) et consiste à construire à partir d'un automate fini *non-déterministe* $\mathcal{N} = (Q, \Sigma, \delta, q_0, F)$ qui reconnaît un langage L , un automate fini *déterministe* $\mathcal{A} = (Q', \Sigma, \delta', q'_0, F')$ qui reconnaît aussi L .

2.3 Langages réguliers

Les **langages réguliers**, aussi appelés *langages rationnels*, sont les langages les plus restrictifs, mais à la fois les plus faciles à manipuler. Ces langages peuvent être reconnus

par les automates finis (cf. section 2.2) et les expressions régulières (cf. section 2.3.2), tous les deux équivalents quant à leur puissance descriptive¹. Nous pouvons définir un langage régulier sous la forme suivante :

Définition 2.30 (Langage régulier). Étant donné un langage L sur Σ , L est un **langage régulier** si, et seulement si, il existe un automate \mathcal{A} qui reconnaît ce langage.

Nous avons déjà présenté les automates finis. Considérons à présent les grammaires régulières et les expressions régulières.

2.3.1 Grammaires régulières

Une **grammaire régulière**, à droite ou à gauche, est une **grammaire formelle** qui décrit un langage régulier². Les grammaires régulières sont explicitées par la définition ci-après :

Définition 2.31 (Grammaire régulière à droite). Une **grammaire régulière à droite** est un quadruplet $\mathcal{G} = (V, T, P, S)$, où (1) V et T sont des ensembles non vides et finis de symboles, (2) $V \cap T = \emptyset$, $\Sigma = V \cup T$, (3) P est un ensemble fini de règles de réécriture et (4) $S \in V$ est l'axiome de la grammaire.

L'ensemble fini P des règles constitue l'élément distinctif d'une grammaire régulière à droite par rapport aux autres grammaires formelles. Les règles de P ont l'une des formes suivantes :

$$A \rightarrow aB$$

$$A \rightarrow a$$

où $A, B \in V$ sont des variables et $a \in T$ est un symbole terminal. Les grammaires régulières à droite permettent alors de remplacer une variable soit par un symbole terminal, soit par un symbole terminal suivi d'une variable. Si à la place de la règle $A \rightarrow aB$ on utilise la règle $A \rightarrow Ba$, c'est-à-dire, que B est placé à gauche de a , la grammaire est alors dénommée **grammaire régulière à gauche**.

Définition 2.32 (Grammaire régulière à gauche). Une **grammaire régulière à gauche** est une grammaire dans laquelle toutes les règles ont la forme :

1. Toute expression régulière R peut être convertie en un automate fini \mathcal{A} et vice versa. Voir à ce sujet (Hopcroft et al., 2000, p. 66)

2. Remarquons que même si une grammaire régulière est limitée à décrire un langage régulier, ce dernier peut aussi être défini par une grammaire non-régulière.

$$A \rightarrow Ba$$

$$A \rightarrow a$$

où $A, B \in V$ et $a \in T$.

N.B. 2.33. Il est important de remarquer qu'une grammaire régulière est soit régulière à droite, soit régulière à gauche, mais pas les deux à la fois. Dorénavant, sans autre précision, nous utiliserons le terme **grammaire régulière** pour designer une *grammaire régulière à droite*.

À présent, considérons la grammaire régulière de l'exemple 2.34.

Exemple 2.34.

$$\mathcal{G}_2 = (V, T, P, S),$$

1. $V : \{S, A\}$
2. $T : \{a, b, c\}$
3. $P :$

S	\rightarrow	$aA \mid c$
A	\rightarrow	bS

Par l'application de P , il est possible d'engendrer, les phrases :

	c
	abc
	ababc
	abababc
	ababababc
	abababababc

Soit le langage :

$$L(\mathcal{G}_2) = \{(ab)^n c \mid n \geq 0\}$$

En ce qui concerne la représentation d'une phrase engendrée par une grammaire régulière sous la forme d'un arbre de dérivation, il est utile de constater que le fait de limiter le remplacement d'une variable par une autre au plus, génère des arbres où, tout fils gauche existant est une feuille. La figure 2.6 illustre l'arbre de la phrase abc de $L(\mathcal{G}_2)$ engendrée par la dérivation :

$$S \Longrightarrow aA \Longrightarrow abS \Longrightarrow abc, \text{ soit } S \xRightarrow{*} abc$$

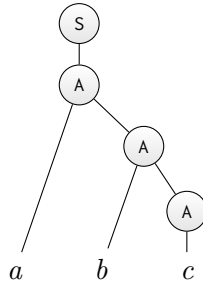


FIGURE 2.6 – Arbre de dérivation de la phrase abc engendrée par \mathcal{G}_2

Mis à part les automates finis et les grammaires régulières¹, les langages réguliers peuvent être définis par des expressions régulières. Nous donnons ensuite leur définition.

2.3.2 Expressions régulières

Une **expression régulière**, aussi appelée expression rationnelle, est un formalisme qui définit selon une notation précise un langage régulier. Les expressions régulières peuvent être définies de façon inductive comme suit :

Définition 2.35 (Expression régulière). Une **expression régulière** R sur un alphabet Σ est soit :

1. Une expression régulière a pour tout $a \in \Sigma$, \emptyset ou ε ,
2. Une expression régulière formée à partir d'autres expressions régulières R_1 et R_2 par les opérations d'union ($R_1 \cup R_2$), de concaténation ($R_1 \circ R_2$) ou d'étoile de Kleene (R_1^*).

Dans (1), l'expression régulière a représente le langage $\{a\}$, l'expression régulière \emptyset indique le langage vide $\{\}$, autrement dit, le langage qui ne contient pas de mots, et l'expression régulière ε est le langage $\{\varepsilon\}$, soit le langage qui contient juste le mot vide.

Dans (2), l'expression régulière $(R_1 \cup R_2)$ indique l'union des langages $L(R_1) \cup L(R_2)$, l'expression régulière $(R_1 \circ R_2)$ est égale à la concaténation des langages $L(R_1) \circ L(R_2)$, et l'expression régulière (R_1^*) est l'étoile de Kleene du langage $L(R_1)^*$.

À partir de l'application d'un nombre fini d'opérations données en (2), il est possible de construire des expressions régulières plus complexes que celles décrites en (1). Il est aussi utile de définir R^+ ² comme la forme abrégée de RR^* . En outre, il est propice de

1. Ou non-régulières, remarquons de nouveau que même si une grammaire régulière est limitée à décrire un langage régulier, ce dernier peut aussi être défini par une grammaire non-régulière.

2. Soit l'ensemble de toutes les concaténations non-vides sur les membres d'un langage L . Notons que L^+ contient ε si, et seulement si, L contient ε et L^+ contient un mot non-vide w si, et seulement si, $w \in L^*$

définir un ordre de priorité pour les opérateurs : d'abord l'étoile de Kleene, ensuite la concaténation et finalement l'union. Nous donnons ci-après quelques exemples d'expressions régulières.

Exemple 2.36. Soit R une expression régulière sur un alphabet $\Sigma = \{0, 1\}$, nous pouvons définir R comme :

1. $0 \cup 1(0 \cup 1)^* = \{w \mid w \text{ est un numéro binaire}\}$
2. $(01)^*$, dont la grammaire est similaire à celle de l'exemple 2.10
3. $1^* \cup (10)^*$, dont le NFA équivalent correspond à celui de l'exemple 2.27

2.4 Langages non contextuels

Les expressions régulières et les automates finis sont équivalents quant à leur puissance descriptive. Cependant, il existe certains langages qui ne peuvent pas être décrits via ces deux formalismes. Un exemple est le langage sur $\Sigma = \{0, 1\}$ défini par $\{0^n 1^n \mid n \geq 0\}$, qui est le langage contenant des mots formés par une suite de plusieurs 0s suivi immédiatement du même nombre de 1s.

Lorsque un langage a une nature récursive il est alors utile d'avoir des méthodes formelles pour définir, engendrer et reconnaître le dit langage. Les **langages non-contextuels (CFLs)** sont une classe plus large que les langages réguliers. Ils permettent de formaliser des récursions et des structures imbriquées propres à certains langages. De plus, ils contiennent les langages réguliers, dans ce sens, tout langage régulier est aussi un langage non contextuel. Bien qu'il existe des langages non contextuel qui ne sont pas réguliers.

Un langage non contextuel peut être défini par une **grammaire non contextuelle** et, à l'instar des grammaires régulières, ces grammaires correspondent à une famille d'automates connue sous le nom des **automates à pile**. Dans la suite, nous explicitons ces deux notions.

2.4.1 Grammaires non contextuelles

Une grammaire non-contextuelle (CFG), aussi appelée grammaire algébrique ou grammaire indépendante du contexte, est une grammaire dotée de règles de réécriture plus expressives que celles d'une grammaire régulière. En effet, les CFGs peuvent définir certaines caractéristiques des langages qui ont une nature récursive, les rendant utiles, par exemple, pour étudier un langage naturel. C'est dans ce cadre que les grammaires non contextuelles ont trouvé une première utilité comme moyen pour comprendre et organiser les relations entre les parties du discours telles que les noms, verbes, pronoms, prépositions. Considérons à présent une définition plus formelle.

Définition 2.37 (Grammaire non contextuelle). Une **grammaire non contextuelle** est une **grammaire formelle**, c'est-à-dire un quadruplet $\mathcal{G} = (V, T, P, S)$, où (1) V et T

sont des ensembles non vides et finis de symboles, (2) $V \cap T = \emptyset$, $\Sigma = V \cup T$, (3) P est un ensemble fini de règles de réécriture et (4) $S \in V$ est l'axiome de la grammaire.

L'ensemble fini P des règles constitue l'élément distinctif d'une grammaire non contextuelle par rapport aux autres grammaires formelles. Il est défini comme ¹ :

$$P = \{(A, \omega) \mid A \in V, \omega \in \Sigma^+\}$$

Les règles sont alors des paires ordonnées (A, ω) de la forme :

$$A \rightarrow \omega$$

où A est une variable et ω une suite de mots contenant n variables et/ou symboles terminaux ($n > 0$). Remarquons que la différence entre une grammaire qui décrit un langage régulier et celle qui décrit un langage non contextuel se situe dans la partie droite de la règle. En effet, tandis que dans une grammaire régulière il est possible de remplacer une variable par un symbole terminal, suivi ou non d'une variable, dans une grammaire non contextuelle, il est permis de remplacer une variable par n'importe quelle suite non vide d'éléments du vocabulaire de la grammaire.

Étudions l'exemple 2.38 qui décrit de façon simplifiée quelques relations entre les parties du discours de la langue anglaise.

Exemple 2.38.

$$\mathcal{H}_1 = (V^2, T, P, S),$$

1. $V : \{\langle S \rangle, \langle NP \rangle, \langle VP \rangle, \langle NNP \rangle, \langle VBD \rangle, \langle DT \rangle, \langle JJ \rangle, \langle NN \rangle\}$ ³
2. $T : \{a, \dots, z, _ \}$
3. $P :$

$\langle S \rangle$	\rightarrow	$\langle NP \rangle \langle VP \rangle$
$\langle NP \rangle$	\rightarrow	$\langle NNP \rangle \mid \langle DT \rangle \langle JJ \rangle \langle NN \rangle$
$\langle VP \rangle$	\rightarrow	$\langle VBD \rangle \langle NP \rangle$
$\langle NNP \rangle$	\rightarrow	Maurice \mid Johannes \mid Niels
$\langle VBD \rangle$	\rightarrow	was \mid wrote \mid saw
$\langle DT \rangle$	\rightarrow	a \mid the
$\langle JJ \rangle$	\rightarrow	French \mid German \mid Danish
$\langle NN \rangle$	\rightarrow	linguist \mid book \mid garden

1. Nous utilisons la définition de Baudot (1987, pp.44–45) qui restreint l'utilisation de variables nulles (cf. définition 2.5). Ceci dit, il est utile de remarquer que si l'on a une CFG avec ε -règles, il est parfois possible de la convertir dans une CFG où on a enlevé le mot vide.

2. Rappelons que les symboles V entourés par des chevrons sont considérés comme des symboles uniques.

3. Nous utilisons pour cet exemple un sous-ensemble du jeu d'étiquettes du Penn Treebank (Marcus et al., 1993). $\langle S \rangle$: phrase ; $\langle NP \rangle$: syntagme nominal ; $\langle VP \rangle$: syntagme verbal ; $\langle NNP \rangle$: nom propre, singulier ; $\langle VBD \rangle$: verbe au passé ; $\langle DT \rangle$: déterminant ; $\langle JJ \rangle$: adjective ; $\langle NN \rangle$: nom singulier.

La grammaire \mathcal{H}_1 possède : 8 variables ($|V| = 8$) ; 27 symboles terminaux correspondant aux lettres de l'alphabet anglais plus l'espace ($|T| = 27$) et 18 règles de réécriture ($|P| = 18$). À partir de cette grammaire il est possible d'engendrer, par exemple, les phrases suivantes :

- (3) *Maurice was a French linguist*
- (4) *Johannes wrote a Danish book*
- (5) *Niels saw the German garden*

Notons que la grammaire \mathcal{H}_1 décrit les phrases de $L(\mathcal{H}_1)$ au niveau syntaxique, représentant la façon dont leurs parties sont agencées et hiérarchisées. Ceci est plus facile à constater à l'aide d'un arbre de dérivation. La figure 2.7 illustre celui de la phrase (3).

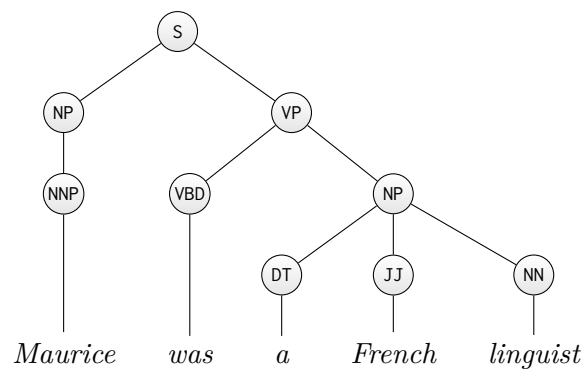


FIGURE 2.7 – Arbre de dérivation de la phrase « *Maurice was a French linguist* » engendrée par la grammaire \mathcal{H}_1

Observons qu'à la différence des arbres de dérivation issus des grammaires régulières, où tout fils gauche est nécessairement une feuille, un arbre de dérivation issu d'une grammaire non contextuelle comprend pour chaque fils, soit un symbole terminal, soit une variable. Ceci de telle sorte que si un nœud est défini par une variable, il constitue à la fois un sous-arbre, lui aussi contenant éventuellement d'autres sous-arbres, où chaque sous-arbre définit une partie de la phrase. Dans la figure 2.7, les sous-arbres de racine $\langle NP \rangle$ définissent les parties de la phrase « *Maurice* » et « *a French linguist* ».

Finalement, il est important de rappeler que la description réalisée par une grammaire non contextuelle est limitée au plan syntaxique. Considérons les phrases :

- (6) * *Johannes was a French garden*
- (7) * *the Danish book saw Maurice*

les phrases (6) et (7) ont été aussi engendrées par \mathcal{H}_1 et par conséquent elles ont une structure syntaxique identique à celle des autres phrases de $L(\mathcal{H}_1)$. Cependant, dans

la phrase (6) le verbe *was* produit une relation grammaticale entre *Johannes* et *a French garden* inattendu (un homme peut-il être un jardin ?), il en est de même pour la phrase (7) où le nom inanimé *the Danish book* est le sujet du verbe de perception *saw* (un livre peut-il regarder un homme ?).

2.4.2 Automates à pile

Un automate à pile (PDA) est une machine abstraite équivalente à un automate fini non-déterministe¹ muni d'une **pile** infinie. Il peut alors écrire des symboles dans la pile et les récupérer plus tard, ce qui lui confère une mémoire supplémentaire au-delà de la quantité finie disponible dans les NFAs. Ainsi, nous définissons formellement le concept d'**automate à pile non-déterministe (NPDA)** de la façon suivante :

Définition 2.39 (Automate à pile non-déterministe). Un **automate à pile non-déterministe** est un sextuplet $\mathcal{S} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, où :

1. Q est un ensemble fini d'états,
2. Σ est l'**alphabet d'entrée**,
3. Γ est l'**alphabet de la pile**,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ est la **fonction de transition**,
5. $q_0 \in Q$ est l'**état initial**, et
6. $F \subseteq Q$ est l'ensemble des **états finaux**.

Notons que la fonction de transition δ d'un automate à pile définit en entrée un état de départ ($q_i \in Q$), un symbole d'entrée ($a \in \Sigma_\varepsilon$) et un symbole au sommet de pile ($x \in \Gamma_\varepsilon$), soit un triplet de la forme (q_i, a, x) .

De même, observons que la sortie de la fonction de transition est un ensemble de paires ordonnées de la forme (q_j, y) de $\mathcal{P}(Q \times \Gamma_\varepsilon)$. Ceci indique qu'étant donnée une entrée (q_i, a, x) , la fonction de transition peut définir en sortie : 1. plusieurs états d'arrivée parmi les états de Q , 2. pour tout état d'arrivée $q_j \in Q$, un symbole $y \in \Gamma_\varepsilon$ à mettre à jour au sommet de pile lorsque le changement d'état est réalisé.

Par exemple, la règle de transition $\delta(q_i, a, x) = (q_j, y)$, précise que lorsque l'automate se trouve à l'état q_i et lit a sur l'entrée, il peut remplacer le symbole x au sommet de pile par y et passer dans l'état q_j . L'étiquette d'une telle transition est notée $a, x \rightarrow y$ est illustrée à la figure 2.8.

1. Rappelons que tout automate fini déterministe est aussi un automate fini non-déterministe

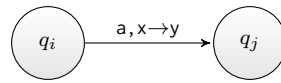


FIGURE 2.8 – Étiquette de transition d'un automate à pile

Définition 2.40 (Transition d'un automate à pile non-déterministe). Si $\delta(q_i, a, x) = (q_j, y)$ est une règle de transition de \mathcal{S} , alors n'importe quel terme a , x ou y peut être égal à ε :

1. Si a est ε , soit $\varepsilon, x \rightarrow y$: aucune opération de *lecture* d'un symbole en entrée est effectuée.
2. Si x est ε , soit $a, \varepsilon \rightarrow y$: aucune opération d'*empilement et dépilement* d'un symbole dans la pile est effectuée.
3. Si y est ε , soit $a, x \rightarrow \varepsilon$: aucune opération d'*empilement* d'un symbole dans la pile est effectuée.

De plus, si à partir d'un état de départ q_i , en lisant un symbole a , l'automate a plusieurs états d'arrivée, par exemple dans une règle de transition de la forme $\delta(q_i, a, x) = \{(q_j, y), (q_k, z)\}$, cela indique que deux choses peuvent arriver, soit l'automate remplace le symbole x en haut de la pile par y et passe à l'état q_j , soit il remplace le symbole x en haut de la pile par z et passe à l'état q_k . Une telle configuration est illustrée à la figure 2.9.

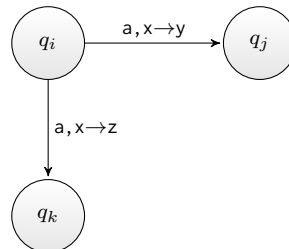


FIGURE 2.9 – Transition non-déterministe d'un automate à pile

Exemple 2.41.

$$\mathcal{S}_1 = (Q, \Sigma, \Gamma, \delta, q_0, F),$$

1. $Q : \{q_0, q_1, q_2, q_3\}$
2. $\Sigma : \{0, 1\}$
3. $\Gamma : \{0, 1\}$
4. δ : est décrite par la table de transition d'état suivante :
5. q_0
6. $F : \{q_0, q_3\}$

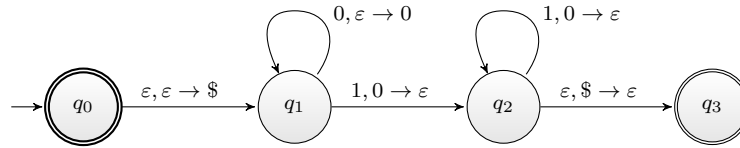


FIGURE 2.10 – Diagramme d'états-transitions d'un automate à pile qui reconnaît le langage $L = \{0^n 1^n \mid n \geq 0\}$

Définition 2.42 (Automate à pile déterministe). Un automate à pile \mathcal{A} est un **automate à pile déterministe** si la fonction de transition est définie par $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow (Q \times \Gamma_\varepsilon) \cup \{\emptyset\}^1$ et elle satisfait la condition suivante : Pour chaque $q \in Q$, $a \in \Sigma$ et $x \in \Gamma$, une et une seule des sorties parmi les configurations suivantes n'est pas égal à \emptyset : $\delta(q, a, x)$, $\delta(q, a, \varepsilon)$, $\delta(q, \varepsilon, x)$ et $\delta(q, \varepsilon, \varepsilon)$

Définition 2.43 (Langage reconnu par un automate à pile). Le **langage reconnu** par un automate à pile \mathcal{A} , noté $L(\mathcal{A})$, est égal à l'ensemble des mots acceptés par \mathcal{A} .

2.5 Langages contextuels

Les langages contextuels sont les langages engendrés par les grammaires contextuelles et reconnus par les automates linéairement bornés. Ces derniers sont équivalents à de machines de Turing (cf. sous-section 2.6.2) non déterministes. Nous définissons ci-dessous les grammaires contextuelles.

2.5.1 Grammaires contextuelles

Une **grammaire contextuelle**, aussi appelée grammaire dépendante du contexte, est une grammaire qui définit un langage contextuel. Considérons à présent une définition plus formelle.

Définition 2.44 (Grammaire contextuelle). Une **grammaire contextuelle** est une **grammaire formelle**, c'est-à-dire un quadruplet $\mathcal{G} = (V, T, P, S)$, où (1) V et T sont des ensembles non vides et finis de symboles, (2) $V \cap T = \emptyset$, $\Sigma = V \cup T$, (3) P est un ensemble fini de règles de réécriture et (4) $S \in V$ est l'axiome de la grammaire.

Comme pour le cas des grammaires régulières et non contextuelles, l'ensemble fini P des règles constitue l'élément central d'une grammaire contextuelle. Il est défini comme :

$$P = \{(\varphi A \psi, \varphi \omega \psi) \mid A \in V, \varphi, \psi \in \Sigma^*, \omega \in \Sigma^+\}$$

1. Ceci indique que l'automate peut soit réaliser un seul mouvement ou ne pas réaliser d'actions en renvoyant \emptyset .

Les règles sont alors des paires ordonnées $(\varphi A\psi, \varphi\omega\psi)$ de la forme :

$$\varphi A\psi \rightarrow \varphi\omega\psi$$

où A est une variable, φ, ψ sont des suites de mots contenant n variables et/ou symboles terminaux ($n \geq 0$) et ω est une suite de mots contenant au moins une variable ou un symbole terminal. Remarquons que comme dans le cas d'une CFG, dans une grammaire contextuelle il est permis de remplacer une variable A par n'importe quelle suite non vide d'éléments ω du vocabulaire de la grammaire Σ . Cependant, pour que la réécriture soit acceptée, ω doit apparaître entre deux suites de mots, éventuellement vides, φ et ψ . Autrement dit, ω doit se trouver dans le *contexte* défini par $\varphi \dots \psi$.

2.6 Langages récursivement énumérables

Les langages récursivement énumérables, aussi dénommés généraux ou sans restriction, sont les langages engendrés par les grammaires contextuelles avec effacement et reconnus par les machines de Turing. Tous les types de langages présentés ci-dessus : réguliers, non contextuels, contextuels sont récursivement énumérables. Bien qu'il existe des langages récursivement énumérables qui ne sont d'aucun des types (1, 2 et 3). Cela étant dit, il n'existe pas d'algorithme pour dire si une chaîne w appartient à un langage défini pour une grammaire contextuelle avec effacement. Nous définissons ci-dessous les grammaires contextuelles avec effacement.

2.6.1 Grammaires contextuelles avec effacement

Une **grammaire contextuelle avec effacement**, aussi appelée grammaire sans restriction, est une grammaire qui définit un langage récursivement énumérable. Considérons à présent une définition plus formelle.

Définition 2.45 (Grammaire contextuelle avec effacement). Une **grammaire contextuelle avec effacement** est une **grammaire formelle**, c'est-à-dire un quadruplet $\mathcal{G} = (V, T, P, S)$, où (1) V et T sont des ensembles non vides et finis de symboles, (2) $V \cap T = \emptyset$, $\Sigma = V \cup T$, (3) P est un ensemble fini de règles de réécriture et (4) $S \in V$ est l'axiome de la grammaire.

À la différence de toutes les autres grammaires vues jusqu'à présent : régulières, non contextuelles et contextuelles, les grammaires contextuelles avec effacement n'ont pas d'autre restriction que d'avoir au moins une variable à gauche. L'ensemble fini P des règles est défini comme :

$$P = \{(\varphi A\psi, \omega) \mid A \in V, \varphi, \psi, \omega \in \Sigma^*\}$$

Les règles sont alors des paires ordonnées $(\varphi A\psi, \omega)$ de la forme :

$$\varphi A\psi \rightarrow \omega$$

où A est une variable, et φ, ψ, ω sont des suites de mots contenant n variables et/ou symboles terminaux ($n \geq 0$). Remarquons que comme dans le cas d'une grammaire contextuelle, il est permis d'avoir n'importe quel élément à gauche de la règle. Cependant, il n'existe pas de restriction de contexte $\varphi \dots \psi$ dans l'élément à droite.

2.6.2 Machines de Turing

Une Machine de Turing (TM) est une machine abstraite qui est modélisée comme une extension d'automates à pile. Nous présentons brièvement ci-dessous leur définition ¹.

Définition 2.46 (Machine de Turing). Une **Machine de Turing** est un septuplet $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \mathbf{b}, F)$, où :

1. Q est l'ensemble d'**états**,
2. $\Sigma \subseteq \Gamma$ est l'**alphabet d'entrée** qui ne contient pas \mathbf{b} ²,
3. Γ est l'alphabet des symboles du **ruban de lecture/écriture**,
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ est la **fonction de transition**,
5. $q_0 \in Q$ est l'**état initial**,
6. $\mathbf{b} \in \Gamma$ est un symbole spécial nommé **symbole blanc**, et
7. $F \subseteq Q$ est l'ensemble des **états finaux**.

Chaque transition de \mathcal{M} est définie comme une paire ordonnée (q_i, γ_i) de $Q \times \Gamma$ où : q_i est l'état de départ, et γ_i est le symbole lu sur le ruban. La sortie de la fonction de transition $\delta(q_i, \gamma_i)$ est égale à un triplet (q_j, γ_j, D) de $Q \times \Gamma \times \{L, R\}$, dont le premier terme est l'état d'arrivée $q_j \in Q$, le deuxième est un symbole $\gamma_j \in \Gamma$ qui remplace γ_i et le troisième est un symbole de direction, L ou R , afin d'indiquer vers quelle position, gauche (L) ou droite (R), doit se déplacer la tête de lecture/écriture. Ceci est illustré dans la figure 2.11.

1. Nous invitons le lecteur à consulter Linz (2016, pp. 232–242) pour plus d'information et d'exemples détaillés concernant les machines de Turing.

2. Soit $\Sigma \subseteq \Gamma - \{\mathbf{b}\}$

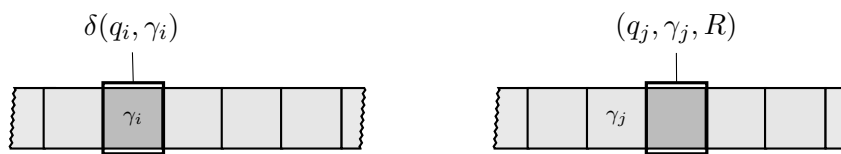


FIGURE 2.11 – Représentation d'un ruban de lecture/écriture d'une Machine de Turing qui est dans un état de départ q_i et qui lit le symbole γ_i pour passer dans un état d'arrivée q_j , en remplaçant γ_i par γ_j et en déplaçant la tête de lecture vers la droite

3

Grammaires locales

Est-il possible de concevoir un modèle abstrait et universel du langage ? Tel est le projet de Noam Chomsky qui jusqu'à présent n'a pas pu être atteint. En revanche, les travaux initiés par Maurice Gross fixent pour objectif de réaliser une description satisfaisante des langues naturelles, ceci en contraste avec les modèles formels des langages qui ne s'attardent pas sur les descriptions, à l'instar des grammaires génératives transformationnelles proposées par Chomsky.

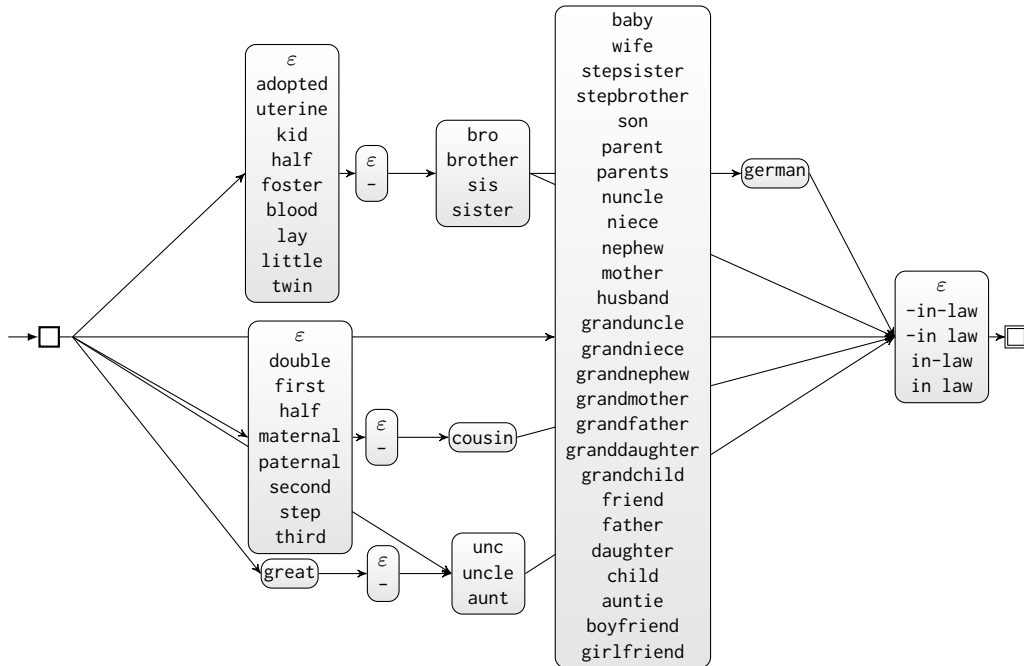
Pour parvenir à une description satisfaisante des langues naturelles Maurice Gross préconise, à la place de concevoir un modèle qui s'appliquerait à toutes les langues, plutôt un de nature strictement locale où la phrase se constitue comme l'unité élémentaire de base au niveau syntaxique et sémantique.

Ce chapitre est dédié aux grammaires locales (LGs). Les grammaires locales sont un formalisme, proche des automates finis, pour décrire, sous forme de graphes, des contraintes syntaxiques ou sémantiques d'une langue. Le concept de grammaire locale est issu des travaux menés par Maurice Gross (1993, 1996, 1997). Nous présentons d'abord un aperçu général des grammaires locales pour étudier ensuite leur définition et principales caractéristiques.

3.1 Aperçu général

Le graphe 3.1 représente une grammaire locale pour reconnaître de forme simplifiée quelques relations filiales (père, mère, fis, etc.) dans un texte en langue anglaise. La façon classique de représenter une grammaire locale est d'utiliser des graphes connectés

Silberztein (1993), lus de gauche à droite. Le graphe se lit de gauche à droite, les nœuds sont dénommés « boîtes », dont celle carrée le plus à gauche est désignée comme boîte de départ et celle en double carrée plus à droite est désignée comme boîte d'arrivée. Une liste complète de symboles utilisées pour schématiser les LGs est disponible au tableau I.



Graphe 3.1 – Reconnaissance de relations filiales

Une chaîne textuelle est reconnue, s'il existe un chemin (cf. définition 2.18) allant de la boîte initiale à la boîte finale où chacune des boîtes parcourues reconnaît une partie de la chaîne présentée en entrée. Le graphe 3.1 acceptera la phrase (8) mais non la (9) :

- (8) Hemings was the **half brother-in-law** of Thomas Jefferson
 (9) Hemings was the **first African-American graduate of Vassar College**

Les étiquettes d'entrée, composées de symboles appartenant à l'alphabet d'entrée, peuvent contenir des mots, des références à des classes de mots, des traits lexicaux sur les mots, le mot vide. De plus, les symboles d'entrée peuvent appartenir à un vocabulaire préétabli peuvent influencer l'analyse, certains concernent la mémorisation des symboles d'entrée ou de sortie : variables d'entrée, variables de sortie. D'autres permettent de lire des symboles d'entrée sans être « consommées » : contextes droits (pré-lecture) et contextes gauches (exclusion). En fin, d'autres permettent de changer l'unité minimale d'analyse : *mode morphologique*.

De plus, les arcs d'un graphe peuvent être chaînés pour former des boucles. Aussi, un graphe peut appeler d'autres sous-graphes, ce qui leur confère une grande puissance

expressive pour caractériser des structures lexicales dans un texte en langue naturelle. Nous reviendrons sur ces aspects plus tard.

Dans la pratique, les graphes peuvent s'organiser dans des bibliothèques de graphes facilitant par la suite leur réutilisation.

3.2 Définition

Les grammaires locales (Gross, 1993, 1997) sont un formalisme de description de règles syntaxiques ou sémantiques. Depuis leur conception, leur pertinence a été prouvée pour traiter plusieurs problèmes du traitement automatique des langues (TAL) liés à la représentation succincte de descriptions linguistiques fines : découpage de phrases, traitements morphologiques, étiquetage, analyse syntaxique de surface, analyses transformationnelles, résolution des ambiguïtés, extraction de l'information, entre autres (Nam et Choi, 1997, Friburger, 2002, Constant, 2003, Traboulsi, 2004, Laporte, 2005, Geierhos et al., 2008, Martineau et al., 2011, Krstev et al., 2013, Ezzat, 2014).

Le formalisme de LGs est proche de celui des automates à états finis et plus largement associée à des réseaux de transitions récursifs (RTNs) (Woods, 1970, Bates, 1978) et des RTNs comportant des sorties (Sastre et Forcada, 2009). Bien qu'il soit aussi fréquemment assimilé au concept de transducteur fini, de part la capacité d'une grammaire locale à gérer des sorties, observons que même si la notion de sortie est comparable à celle de transduction, les grammaires locales et les transducteurs ne sont pas toujours équivalents. En effet, comme souligné par Blanc (2006, p. 62), l'application d'un transducteur sur un texte en entrée produit comme résultat un nouveau texte composé par la concaténation des sorties, tandis qu'avec une grammaire locale, le résultat peut être en plus égal au texte en entrée combiné avec les séquences de sortie.

En général dans la littérature, la notion de grammaire locale renvoie à des définitions et des mises en œuvre différentes. D'une part, par la variété de types de grammaires locales concernées, ces type peuvent être divisées selon l'analyse effectuée : lexicale (flexion, prétraitement), syntaxique (locale ou structurelle) ou transformationnelle. D'autre part, par les appellations utilisées pour y faire référence, entre autres : transducteurs finis (Fairon et Watrin, 2003, Sætre, 2003, Ranchhod et al., 2004, Kevers, 2006), automates finis (Traboulsi, 2005), automates lexicaux (Blanc et Dister, 2004), automates syntaxiques (Monnier et al., 2003). Finalement, la notion de grammaire locale dépend aussi de l'outil informatique utilisée, nous pouvons distinguer deux groupes :

- Ceux qui ne disposent pas de la capacité de concevoir des grammaires locales à l'aide d'une interface graphique, mais sous la forme d'ensembles de règles de dérivation ou de tableaux de transition d'état. Nous citons par exemple, les grammaires locales du type dag2dag (Sagot et Boullier, 2008) de SXPipe (Sagot et Boullier, 2005), exprimées dans un langage proche de la forme de Backus-Naur (BNF) et analysées à l'aide du système SYNTAX (Boullier et Deschamp, 1991). De plus, des outils comme

OPENFST (Allauzen et al., 2007), pouvant manipuler des RTNs aplatis¹ (Allauzen et Riley, 2012).

- Ceux qui offrent la possibilité de développer des grammaires locales à l'aide d'interfaces graphiques, en faisant appel à l'utilisation de ressources linguistiques, comme des dictionnaires électroniques (Maurel, 1993, Courtois et al., 1997, Chrobot et al., 1999) qui décrivent la syntaxe et la morphologie des entrées lexicales et de leurs attributs sémantiques. Ainsi que d'avoir des sorties, d'utiliser des variables, c'est-à-dire, des registres qui stockent des sous-séquences d'une séquence en entrée, et d'établir de contraintes sur les variables, sous la forme d'opérateurs primitifs. Nous énumérons, par exemple, INTEX (Silberztein, 1994), UNITEX (Paumier, 2003a), NOOJ (Silberztein et Tutin, 2005) ou OUTILEX (Blanc et al., 2006).

Dans notre recherche une grammaire locale concerne des grammaires non contextuelles (cf. sous-section 2.4.1) qui décrivent des langues naturelles. Elles sont représentées par des graphes orientés *lexicalisés* et récursifs. En plus, les grammaires locales peuvent être dérécursivées (cf. section 4.5) et modélisés par des machines à états finis.

Les grammaires locales sont aussi enrichies par des registres et des contraintes qui ne font pas partie du modèle des automates finis, mais qu'en vue de traiter certaines opérations classiques des automates peuvent, avec des ajustements, être considérées comme des symboles d'entrée dépourvus de sens².

Finalement, bien que les grammaires locales sont restreintes à décrire des langages non contextuels (cf. section 2.4), la possibilité d'ajouter des sorties et de contraintes leur donne la capacité de reconnaître des langages contextuels (cf. section 3.5.8) et de réaliser des opérations transformationnelles.

Malgré la puissance inhérente aux *automates finis* (FSAs), ceux-ci ne possèdent pas de mécanisme leur permettant faire appel à d'autres FSAs. Pour palier ce problème, tout en conservant certains avantages des modèles d'automates finis classiques, les grammaires locales sont représentées par des RTNs (Woods, 1970, Bates, 1978).

1. L'aplatissement ou la dérécursivation d'un RTN fait référence au processus de remplacement de transitions qui désignent l'appel à un sous-réseau. Afin d'obtenir un automate fini équivalent, il est nécessaire de substituer les transitions étiquetées par des symboles non-terminaux, par une copie exclusive du sous-réseau appelé.

2. Par exemple, un registre qui fait référence au nom d'une variable est considéré comme un symbole d'entrée terminale. C'est lors de l'analyse syntaxique que l'algorithme en charge donne le sens au symbole. Par exemple, en associant un registre en mémoire au prochain symbole lu.

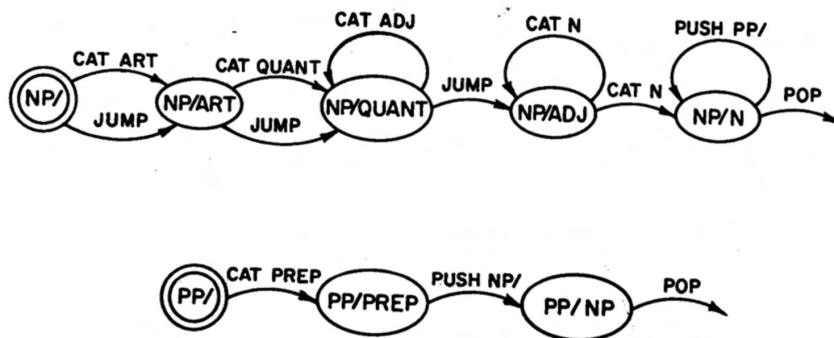


FIGURE 3.1 – Exemple d'un RTN pour la reconnaissance de phrases nominales (Bates, 1978, p. 193)

Un réseau de transitions récursif est une grammaire non contextuelle où on admet que les membres droits des règles soient des automates finis au lieu de simples séquences de symboles, lorsqu'ils modélisent des grammaires locales ils sont définis comme un graphe de nœuds et d'arêtes (les liens entre les nœuds). Les arêtes sont étiquetées avec des symboles de sortie ; les nœuds, l'alphabet d'entrée, peuvent contenir des mots, dont le mot vide epsilon (ε), des traits lexicaux sur les mots, ou faire appel à un sous-réseau (état non-terminal). Le graphe est lu de gauche à droite, c'est-à-dire, l'un des nœuds est désigné comme nœud initial et un autre comme nœud final. Une chaîne textuelle est reconnue, si un chemin existe allant du nœud initial au nœud final où chacun des nœuds parcourus accepte une sous-partie de la chaîne présentée en entrée. Les arêtes d'un RTN peuvent être connectées, pour former des cycles dans le graphe. Les réseaux sont alors nommés récursifs, une caractéristique, qui leur confère une grande puissance expressive pour caractériser des structures lexicales dans un texte en langue naturelle.

3.3 Modélisation

Formellement, une grammaire locale peut être définie sous la forme d'automate fini comme étant un 7-tuple $G = (Q, \Sigma, \Gamma, I, F, E, s)$, où :

- Q est un ensemble fini d'états
- Σ est l'alphabet d'entrée constitué d'un ensemble fini de symboles
- Γ est l'alphabet de sortie constitué d'un ensemble fini de symboles
- I est l'ensemble sous-initial des états qui étiquettent des transitions (*soit les appels à des sous-graphes*)
- $F \subseteq Q$ est l'ensemble des états accepteurs ($F \neq \emptyset$)

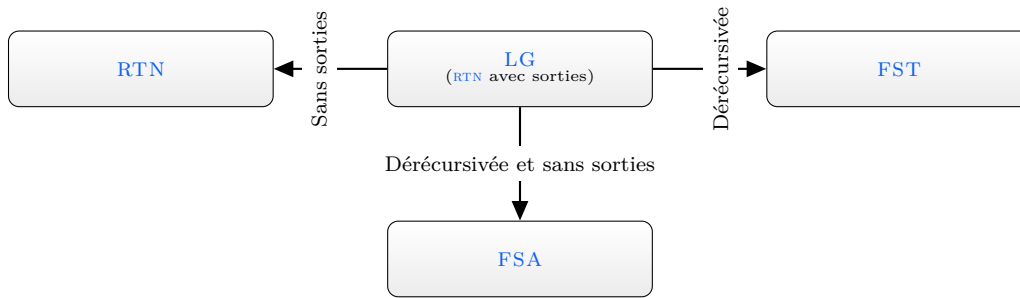


FIGURE 3.3 – Possibles équivalences entre les LGs, RTNs, transducteurs finis (FSTs) et FSAs

3.4 Outils existants

Le formalisme des grammaires locales à été implémenté par différents outils pour traiter des problèmes liés au traitement automatique des langues. Par la suite, nous passons brièvement en revue certains de ces outils.

3.4.1 INTEX

INTEX (Silberztein, 1993, 1994, 2000) est un environnement de développement de dictionnaires électroniques, de grammaires et de lexiques-grammaires pour le traitement de corpus : recherche d'expressions dans des textes, indexation des motifs trouvés, concordances et statistique des résultats. INTEX est à l'origine de plusieurs caractéristiques utilisées par ses successeurs :

- Les opérations de prétraitement du texte dont le découpage en unités lexicales, le découpage en phrases, la normalisation des formes non-ambiguës, et l'application des dictionnaires électroniques du LADL (DELA) (Courtois et Silberztein, 1990, Courtois et al., 1997).
- L'interface graphique pour construire des graphes. Notamment par l'utilisation de boîtes (les nœuds du graphe) pour associer les étiquettes de transition de l'automate sous-jacent. Ceci en reprenant les conventions proposées originalement par Maurice Gross (Gross, 1987).
- L'utilisation de méta-symboles pour reconnaître, ou non, des classes de caractères, par exemple ⟨MOT⟩, ⟨MAJ⟩, ⟨!DIC⟩; ainsi que des masques lexicaux afin de faire référence aux informations fournies par les dictionnaires, par exemple, ⟨N⟩, ⟨V⟩, ⟨pomme.N⟩.
- Les politiques pour traiter les sorties de la grammaire : soit par remplacement (mode *replace*) où les sorties remplacent les entrées reconnues, soit par combinaison (mode *merge*) où les sorties produites sont insérées dans le texte. Également les

politiques permettant de sélectionner le mode de reconnaissance en donnant priorité aux longueurs des séquences reconnues : soit les plus courtes, soit les plus longues.

INTEX été développé en C++ et distribué avec une licence propriétaire sans possibilité de connaître les détails sur les algorithmes utilisés.

3.4.2 UNITEX

UNITEX¹ [Paumier \(2003a\)](#), est un environnement logiciel libre dédié à l'analyse du langage naturel. Il a été développé par Sébastien Paumier et l'équipe d'informatique linguistique du [Laboratoire d'Informatique Gaspard-Monge \(LIGM\)](#). À partir de la version 3.0 UNITEEX est devenu UNITEEX/GRAMLAB et continue à être développé au tour d'une communauté regroupant des linguistes et des développeurs. UNITEEX débute comme une alternative libre à INTEX.

3.4.3 NOOJ

NOOJ² ([Silberztein, 2004](#), [Silberztein et Tutin, 2005](#)) représente l'évolution d'INTEX.

3.4.4 OUTILEX

L'analyseur syntaxique d'OUTILEX³ ([Blanc et al., 2006](#)) est de la famille des analyseurs Earley.

3.4.5 SXPIPE

les grammaires locales du type dag2dag ([Sagot et Boullier, 2008](#)) de SXPIPE⁴ ([Sagot et Boullier, 2005](#)), exprimées dans un langage proche de la BNF et analysées à l'aide du système SYNTAX ([Boullier et Deschamp, 1991](#)).

3.4.6 OPENFST

OPENFST⁵ ([Allauzen et al., 2007](#)) est une bibliothèque logicielle libre et modulaire orientée vers la manipulation des transducteurs pondérés (WFSTs). Elle fournit des algorithmes efficaces en temps et en espace pour la construction, la combinaison, l'optimisation et la recherche sur les automates. Mise à part un cœur d'opérations

-
1. <http://unitexgramlab.org/>
 2. <http://www.nooj-association.org>
 3. <http://igm.univ-mlv.fr/~mconstan/outilex>
 4. <http://alpage.inria.fr/~sagot/sxpipe.html>
 5. <http://www.openfst.org>

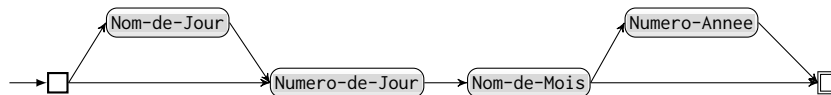
classiques, OPENFST met à disposition des *extensions* avec de nouveaux algorithmes et des automates équivalents aux WFSTs, dans ces derniers, une extension permettant de manipuler des transducteur à pile (PDT) est fournie. Une des opérations disponibles sur les PDTs, est celle de l'aplatissement (Allauzen et Riley, 2012, p. 8) d'un RTN, c'est-à-dire, le remplacement des transitions étiquetées par des symboles non-terminaux qui désignent l'appel à un sous-graphe par une copie exclusive du même graphe, afin d'obtenir un PDTs équivalent.

3.5 Caractéristiques principales

Nous présentons quelques caractéristiques principales des grammaires locales, non sans rappeler que dans le contexte de notre travail, nous nous concentrons sur les graphes syntaxiques.

3.5.1 Faire appel à des sous-graphes

Les grammaires locales comportent la notion de sous-graphe. Cette notion permet de décomposer une analyse en différentes analyses plus simples qui s'attachent chacune à la reconnaissance d'une sous-partie du motif recherché. Considérons par exemple une version simplifiée de la reconnaissance de dates. Elle peut être exprimée par la reconnaissance des quatre sous-motifs suivants : Nom de Jour, Numéro de Jour, Nom de Mois, Numéro d'Année.



Graph 3.2 – Appels de sous-graphes pour chaque sous-motif

Le graphe 3.2 fait appel à un sous graphe pour chacun des constituants potentiels d'une date complète. Comme nous le constatons dans la figure, certains d'entre-eux sont optionnels. Les sous graphes peuvent être de simples graphes ou bien, eux-mêmes faire appel à d'autres sous-graphes. Il serait possible par exemple de créer un sous-graphe pour traiter le sous-motif lorsqu'il est écrit en chiffre et un autre pour le traitement de la version en lettres. Cette notion de sous-graphe « en réseau » proche de celle des RTNs rend possible une représentation aisée et structurée de motifs complexes.

3.5.2 Faire référence à des masques lexicaux

Dans une grammaire locale, il est possible d'utiliser des masques lexicaux. Ces masques se présentent sous la forme d'une requête à des dictionnaires entourée de chevrons (<>). Le format de cette requête est conforme au format des informations présentes dans les entrées de dictionnaires. Par exemple, <N+Hum:fp> reconnaît un nom humain féminin

pluriel. Il est également possible d'utiliser le symbole \sim pour indiquer qu'un trait ne doit pas figurer dans l'entrée. $\langle N \sim \text{Hum} \rangle$ fait référence à un nom non humain.

3.5.3 Faire référence à des méta-symboles

Les grammaires locales disposent de méta-symboles capables de représenter certaines séquences de caractères selon leur structure ou leur appartenance ou non à un dictionnaire. La liste de méta-symboles est disponible au tableau II.

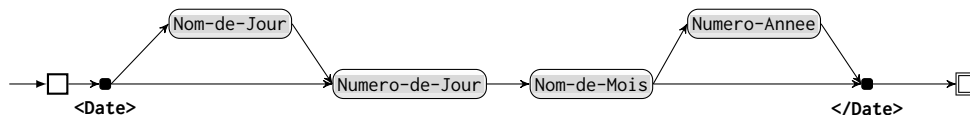
En combinant un masque lexical $\langle N + \text{Given} \rangle$ et un méta-symbole $\langle \text{FIRST} \rangle$ (reconnait un mot qui commence par une majuscule) le graphe 3.3 constitue une grammaire basique capable de reconnaître des noms de personnes.



Graph 3.3 – Graphe basique de reconnaissance de noms de personne

3.5.4 Gérer des sorties

Les grammaires locales ne se limitent pas à pouvoir lire le texte en entrée. Elles permettent également d'écrire dans le texte traité et de produire un nouveau texte enrichi. Cette fonctionnalité est très utilisée pour annoter un texte.

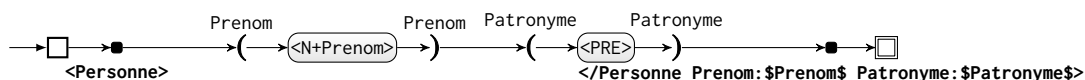


Graph 3.4 – Écriture de sorties : balisage de dates.

Le graphe 3.4 reprend celui présenté au graphe 3.2. Il comporte en plus deux sorties $\langle E \rangle / \langle \text{Date} \rangle$ et $\langle E \rangle / \langle / \text{Date} \rangle$ qui entourent la date reconnue. L'application de cette grammaire ajoute donc des balises autour de toutes les occurrences de date trouvées dans le texte. Un nouveau texte annoté peut être ainsi généré.

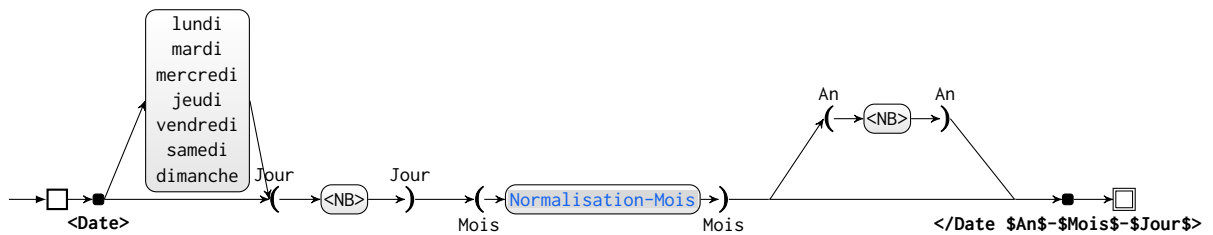
3.5.5 Stocker des variables locales

Dans certaines applications il est souhaitable de produire une annotation qui ne se limite pas à une annotation globale du motif trouvé. Le graphe 3.3 de reconnaissance de nom de personne peut être enrichi en ce sens afin de produire une annotation qui distingue les deux constituants : prénom, patronyme. Dans ce but, nous utilisons deux variables qui captent chacun d'eux.



Graph 3.5 – Annotation de noms de personne

Le graphe 3.5 stocke le patronyme et le prénom dans des variables et délimite le nom complet reconnu au moyen de balises d'étiquette « Personne ». La balise fermante comporte la séquence `Prénom:$Prenom$ Patronyme:$Patronyme$` dans laquelle les symboles \$ sont utilisés pour indiquer que nous voulons afficher le contenu de la variable qu'ils entourent. Ainsi, si dans un texte nous rencontrons le nom *Emmanuel Macron* le graphe produit : `<Personne>Emmanuel Macron</Personne Prénom:Emmanuel Patronyme:Macron>`. Dans ce graphe les variables mémorisent des éléments issus du texte analysé en entrée. Il est également possible de stocker des données issues des sorties d'une grammaire.



Graph 3.6 – Normalisation de dates

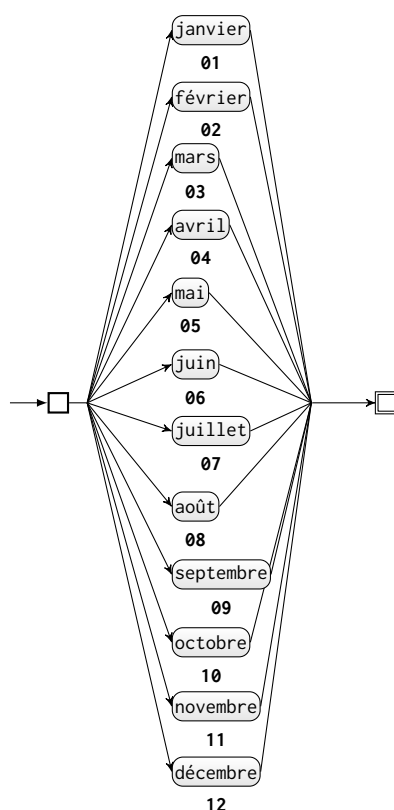
Le graphe 3.6 reconnaît une date comportant au moins le numéro de jour et le mois : la reconnaissance des noms de jour et de l'année sont optionnels. La date reconnue est balisée ; la balise fermante comporte la forme normalisée de la date reconnue. Le graphe utilise le méta-symbole `<NB>` pour reconnaître le numéro du jour et de l'année qui sont captés dans des variables qui mémorisent des entrées issues du texte (parenthèses rouges). Il fait appel à un sous-graphe *Normalisation_Mois* (graphe 3.7) qui transforme le nom du mois en son numéro. Le résultat de cette transformation est mémorisé dans la variable *Mois* qui stocke une sortie du graphe (parenthèses bleues). Enfin les variables contenant les parties normalisées de la date sont utilisées dans la balise fermante afin de produire la forme normalisée complète souhaitée. Ainsi l'application de la grammaire à la date *mercredi 18 octobre 2017* produit : `<Date>mercredi 18 octobre 2017</Date 2017-10-18>`.

3.5.6 Utiliser des filtres morphologiques

Nous avons évoqué précédemment les notions de masques lexicaux et de méta-symboles. Les recherches effectuées grâce à eux peuvent être affinées (restreintes) en leur adjoignant un filtre morphologique entre double chevrons `<<>>` qui ajoute une contrainte supplémentaire. Il est possible par exemple de rechercher des adverbes qui se terminent par *ment* `<ADV><<ment$>>` (le dollar représente la fin du mot), des verbes qui commencent par l'un des préfixes *re*, *ré*, *auto* `<V><<(auto|re|ré)>>` (le symbole ^ représente le début du mot et le pipe « | » le ou logique) ou bien encore des mots en majuscules qui finissent par *s* `<UPPER><<s$>>`.

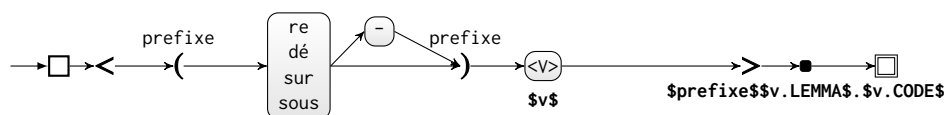
3.5.7 Définir l'unité minimale d'analyse

Pour analyser un texte de manière efficace en terme de temps d'exécution il est habituel de le décomposer en une unité plus grande que le caractère : le token. Un token est



Graph 3.7 – Sous-graphe de normalisation de mois

soit une suite de lettres d'une longueur supérieure ou égale à 1, soit un caractère seul. Le texte est alors vu comme une suite de numéros de tokens. Les tokens proprement dits sont listés dans un lexique construit au cours de la lecture du texte. L'indice d'un token dans cette liste est le numéro utilisé dans la représentation du texte évoquée précédemment. Dans la plupart des analyses l'utilisation du token comme unité minimale est satisfaisante. Cependant, il est utile de pouvoir avoir accès au contenu d'un token pendant l'analyse. Les filtres morphologiques permettent de filtrer les tokens selon leur contenu mais ne permettent pas d'analyser des suites de lettres dynamiquement construites pendant l'analyse. Pour avoir un accès plus fin au contenu d'un token il faut utiliser une analyse caractère par caractère. Ceci est rendu possible au sein des grammaires locales en utilisant le mode morphologique.



Graph 3.8 – Mode morphologique : reconnaissance et étiquetage de formes verbales préfixées

Le graph 3.8 permet de détecter dans un texte des verbes ayant comme préfixe *re*, *dé*, *sur*, *sous* à partir de formes non préfixées déjà présentes dans le dictionnaire.

Les chevrons violets indiquent la zone de la grammaire dans laquelle l'analyse est effectuée caractère par caractère. La première boîte reconnaît les préfixes recherchés et mémorise celui effectivement trouvé. La deuxième boîte qui reconnaît un verbe ($\langle V \rangle$) comporte en dessous la variable $\$v\$$ (variable morphologique). Cette dernière est utilisée afin d'étiqueter entre accolades la forme verbale reconnue. Cette étiquette est construite d'une part, grâce à l'adjonction du préfixe au lemme de la forme verbale non préfixée, d'autre part grâce à la récupération des informations syntaxico-sémantique de cette même forme dont hérite ainsi la forme verbale préfixée. Si par exemple, nous rencontrons dans un texte les formes verbales *sousalimentait* ou *sous-alimentait* et si le verbe alimenté est présent dans le dictionnaire utilisé, la grammaire étudiée produit respectivement les annotations *sousalimentait*{*sousalimenter.V+z1 :I3s*}, *sous-alimentait*{*sous-alimenter.V+z1 :I3s*}.

3.5.8 Capacité à reconnaître des langages contextuels

Le principe pour concevoir des grammaires locales capables de reconnaître des langages contextuels (cf. section 2.5), repose sur le fait, tel que décrit par Silberztein (2015), que *tout langage est inclus dans un langage rationnel*. Pour illustrer ceci, nous reprenons l'exemple de deux langages \mathcal{L}_1 et \mathcal{L}_2 disponible dans Donabédian et al. (2013, p. 8) et Silberztein (2015, p. 229).

Le langage \mathcal{L}_1 est égal à $\{a^n b^n c^n \mid n > 0\}$, autrement dit, celui des mots qui sont formés par une suite de n lettres a , suivi par n lettres b , suivi par n lettres c . \mathcal{L}_1 est alors contextuel (cf. section 2.5), les mots qui appartiennent à ce langage sont $\{abc, aabbcc, aaabbccc, aaaabbbbcccc, \dots\}$.

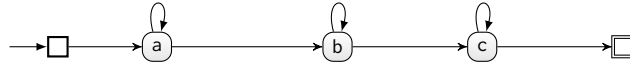
Le langage \mathcal{L}_2 est égal à $\{a^+ b^+ c^+\}$, c'est-à-dire, celui des mots qui sont formés par une suite arbitraire de lettres a , suivi par une suite arbitraire de lettres b , suivi par une suite arbitraire de lettres c . \mathcal{L}_2 est régulier et les mots qui appartiennent au langage sont $\{a, b, ab, abc, aa, ab, bb, aab, abc, aabb, \dots, aabbcc, \dots, aaabbbccc, \dots\}$.

Il n'est pas difficile de constater que les mots engendrés par le langage contextuel \mathcal{L}_1 , sont aussi des mots du langage régulier \mathcal{L}_2 . Nous pouvons généraliser ce fait en prenant en compte que pour un alphabet Σ , les mots engendrés par n'importe quel type de langage L dans l'hierarchie de Chomsky : régulier (type 3), non contextuel (type 2), contextuel (type 1) ou sans restriction (type 0), sont inclus dans Σ^* . Du fait que Σ^* est le langage rationnel formé par l'ensemble de tous les mots sur Σ , nous avons alors que $|L \cap \Sigma^*| = |L|$.

Étant donné que tout langage est inclus dans un langage rationnel, la stratégie mise en place dans une grammaire locale pour reconnaître \mathcal{L}_1 , le langage contextuel, est tout d'abord de reconnaître \mathcal{L}_2 , le langage régulier, et par la suite d'exclure les séquences qui n'appartiennent pas à \mathcal{L}_1 ¹. Pour la reconnaissance de $\mathcal{L}_2 = \{a^+ b^+ c^+\}$, la première

1. Un théorème connexe abordé dans Grune (2014) et prouvé par Ginsburg et al. (1967) est que *tout langage sans restriction (type 0), peut être engendré en intersectant deux langages non contextuels (type 2) et en appliquant un homomorphisme d'effacement au résultat* : $\mathcal{L}_0 = h_e(\mathcal{L}_2 \cap \mathcal{L}_2)$

partie, il est possible d'utiliser une grammaire locale comme celle du graphe 3.9, dont l'automate équivalent est celui de la figure 3.4.



Graph 3.9 – Grammaire locale qui reconnaît $\mathcal{L}_2 = \{a^+b^+c^+\}$

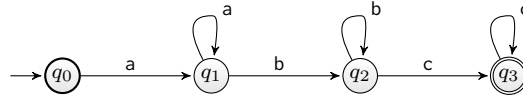
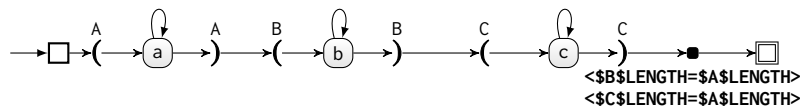


FIGURE 3.4 – Automate équivalent au graphe 3.9

Une fois une séquence $w \in \mathcal{L}_2$ reconnue, il est requis de définir la façon de vérifier si w appartient également à $\mathcal{L}_1 = \{a^n b^n c^n \mid n > 0\}$. Dans cet exemple, il est donc nécessaire de vérifier si w est bien formé par une suite de n lettres a , suivi par n lettres b , suivi par n lettres c . Pour réaliser une telle opération, les grammaires locales doivent être dotées de variables d'entrée, des registres qui stockent des sous-séquences d'une séquence en entrée, ainsi que de contraintes, sous la forme d'opérateurs, permettant de vérifier si deux variables d'entrée ont la même longueur. Par exemple, dans NOOJ, la longueur d'une variable d'entrée peut être obtenue à l'aide de l'opérateur LENGTH et leur équivalence avec l'opérateur égal (=) (Silberztein, 2015, p. 230). La grammaire locale NOOJ du graphe 3.10, dont l'automate équivalent est celui de la figure 3.5¹, illustre comment le graphe 3.9 peut être doté de variables et de contraintes afin de vérifier si $w \in \mathcal{L}_1$.



Graph 3.10 – Grammaire NooJ qui reconnaît $\mathcal{L}_1 = \{a^n b^n c^n \mid n > 0\}$

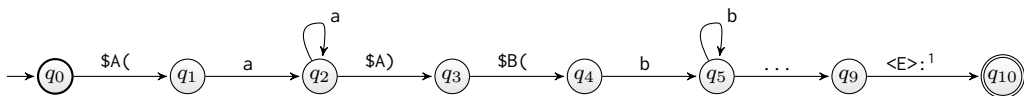


FIGURE 3.5 – Automate équivalent au graphe 3.10

Le graphe 3.10 constitue un des exemples de la façon dont une grammaire locale peut reconnaître un langage contextuel (ici \mathcal{L}_2) tout en reconnaissant un langage régulier (ici \mathcal{L}_1) et en excluant les séquences qui n'appartiennent pas au langage contextuel. En général, cette approche peut s'étendre en gardant le principe d'utiliser des grammaires

1. En raison de l'espace réduit, la transition $q_9 \xrightarrow{\langle E \rangle : \langle B\$LENGTH=\$A\$LENGTH \rangle \langle C\$LENGTH=\$A\$LENGTH \rangle} q_{10}$ est désignée de la forme $q_9 \xrightarrow{\langle E \rangle : 1} q_{10}$.

qui peuvent être analysées plus efficacement, telles que les grammaires régulières et non contextuelles, munies des contraintes pour exclure les séquences qui n'appartiennent pas à un langage défini par une grammaire contextuelle.

3.6 Grammaires locales et extraction de l'information

Les approches d'extraction automatique d'information basées sur la construction de grammaires locales modélisent le problème sous forme ascendante, c'est-à-dire en tenant compte d'abord les dépendances entre groupes voisins de mots et ensuite entre les groupes plus distants. En effet, il est souhaitable d'analyser tout d'abord les contextes immédiats des mots afin d'identifier les éléments d'indice qui peuvent, ou non, déclencher l'identification des motifs recherchés pour étudier ensuite les contextes plus éloignés ou complexes.

Les contextes trouvés peuvent alors se diviser en deux groupes : **internes** ou **externes**. Les contextes internes font partie des motifs recherchés et constituent alors une *preuve interne* pour l'identification, en revanche, les contextes externes n'appartiennent pas aux motifs recherchés et représentent alors une *preuve externe* pour l'identification.

Une fois les contextes caractérisés, ils sont utilisés pour construire des graphes simples au niveau des mots, aussi nommés graphes de premier niveau, qui éventuellement sont appelés par des graphes de niveau supérieur, capables d'identifier des séquences de plus en plus complexes.

Cette approche est utilisée dans les travaux de Sætre (2003) pour le développement d'un système d'extraction automatique d'information à partir de textes biomédicaux. En partant d'un ensemble de graphes qui reconnaissent des motifs simples (par exemple, des noms de gènes), des graphes d'un niveau supérieur sont construits pour décrire des séquences de plus en plus complexes (par exemple, des interactions entre les gènes).

Les grammaires locales sont aussi exploitées dans le cadre d'un système d'indexation de noms de personnalités développé par Fairon et Watrin (2003). À l'opposé d'autres travaux, le corpus à analyser n'est pas prédéfini, mais il s'agit d'un flux continu de textes issue du web qui est généré par un service tiers¹.

Les grammaires développées utilisent un dictionnaire de noms de personnes et autre de noms de professions et décrivent des motifs qui se trouvent autour du nom d'une personnalité. Ces motifs doivent nécessairement² être accompagnés par des informations bibliographiques comme l'âge ou la profession, par exemple, comme dans : *Mr. X, director of Y – The director of Y, Mr. X*, où *X* est un nom de personne et *Y* le nom d'une organisation. À part la tâche d'identification, les graphes sont utilisés pour annoter les entités reconnues à l'aide de balises. Une fois le texte annoté, une

1. GLOSSANET, Fairon (1998) : <http://glossa.fltr.ucl.ac.be>

2. La démarche choisie par les auteurs et mise en œuvre par le système est alors de privilégier la précision en donnant moins d'importance au taux de rappel

dernière phase, basée sur l'analyse des *collocations* des motifs extraits, est entamée afin de synthétiser et d'identifier les clés d'indexation les plus pertinentes.

Les travaux de *Aubin et Barbier (2003)* autour de l'extraction d'information à partir de documents contractuels font appel à l'utilisation des grammaires locales dans le but de remplir une base de données. Ces grammaires servent à identifier les dates ainsi que les noms de personnes ou d'organismes qui sont cités dans des contrats passés entre un centre de recherche public français et ses partenaires. Le processus de construction de grammaires commence par mettre en évidence des motifs lexicaux qui sont fréquemment associés aux entités recherchées. Il est suivi par la constitution de dictionnaires électroniques spécialisés qui sont ultérieurement utilisés par des graphes nécessaires à l'extraction des motifs recherchés. Même si l'étude ne révèle pas de résultats quantitatifs concernant les performances, les auteurs constatent des *taux de bruit relativement faibles* ce qui s'avère pertinent pour la tâche requise.

Poibeau et al. (2003) utilisent les grammaires locales comme partie d'un système de reconnaissance d'entités nommées dans des documents multilingues. Pour cela, des dictionnaires de noms propres et des grammaires locales sont conçues spécifiquement pour chacune des langues visées, les graphes sont développés en prenant en compte les mots et les contextes qui constituent des déclencheurs pour l'identification d'une entité. Une fois un texte analysé par les grammaires, le traitement se poursuit par un module destiné à améliorer les performances globales par la reconnaissance de nouvelles entités jusqu'alors inconnues et négligées.

La technique pour étendre l'ensemble des expressions identifiées est fondée sur l'apprentissage supervisé (*Cornuéjols et Miclet, 2011*, p. 41). Tandis que la connaissance du système est représentée par les lexiques et les grammaires, l'ensemble d'exemples d'apprentissage est constitué par les entités, inconnues au départ, qui ont été positivement identifiées par l'application des grammaires locales. Par exemple, si *Newton* est un mot inconnu au départ et si lors du passage des grammaires sur l'expression *Sir Isaac Newton*, il est reconnu comme étant un nom de personne, alors cette information est mémorisée. La méthode d'apprentissage utilisera cette dernière pour construire une *fonction de prédiction* permettant de déterminer que les occurrences isolées du mot *Newton* peuvent également être étiquetées comme un nom de personne.

Suite à l'application des grammaires locales et à l'identification supplémentaire des entités qui ont été négligées, un dernier module est utilisé pour, à partir d'un contexte donné, résoudre les conflits d'étiquetage. En suivant l'exemple précédent, les occurrences isolées du mot *Newton* qui sont étiquetées erronément comme un nom de personne dans la phrase *The Second Law of Newton* sont corrigées grâce au mot *Law*.

Dans *Saidi (2004)*, les grammaires locales sont utilisées pour l'identification et l'extraction de motifs dans des annonces de séminaires, tels que le lieu, l'adresse, la date ou l'heure.

Nakamura (2004) propose d'analyser automatiquement des bulletins boursiers au moyen de grammaires locales. L'approche utilisée consiste à faire une caractérisation linguistique fine (*analyse linguistique, syntaxico-sémantique et lexicale*) des schémas de phrase pour ensuite transposer chaque composant de cette description sous la forme

d'une grammaire locale.

[Dister et al. \(2004\)](#) décrivent une technique de repérage et d'extraction de mots inconnus permettant leur indexation dans un dictionnaire électronique. La méthode est développée dans le cadre du GLOSSANET ([Fairon, 1998](#)), un service proposant de créer des corpus dynamiques à partir du web¹. En premier lieu, une grammaire permettant d'extraire les formes commençant par une majuscule et qui ne sont pas reconnues par un dictionnaire des mots simples permet de repérer les mots à analyser. Ensuite, une caractérisation et classification manuelle des types de mots inconnus permet la création de nouveaux dictionnaires électroniques ou la mise à jour des dictionnaires déjà existants.

[Ranchhod et al. \(2004\)](#) présentent des méthodes appliquées pour développer des ressources linguistiques en portugais pour le traitement automatique du langage. Les ressources sont divisées en dictionnaires et grammaires. D'une part, des dictionnaires généraux de mots simples et de mots composés ont été créés, ainsi que des dictionnaires spécialisés de noms propres (organisations, toponymes, et acronymes). D'autre part, en utilisant de l'information linguistique contenue dans les dictionnaires, des grammaires de désambiguïsation ([Laporte et Monceaux, 1999](#)) et des graphes syntaxiques ont été développés. Tandis que les grammaires de désambiguïsation visent à lever l'ambiguïté entre des noms et des adjectifs qui sont homographes, les grammaires locales sont utilisées pour décrire des dates, des expressions temporelles, des pourcentages et des structures prédicatives plus complexes.

Afin d'extraire des données biographiques contenues dans des dépêches de presse et de les organiser dans une base de connaissances, [Kevers \(2006\)](#) propose premièrement de créer une définition fonctionnelle des événements biographiques (par exemple : naissance, décès, profession, etc.) sous forme de triplets ($\{ \textit{ sujet, relation, objet } \}$) permettant d'exprimer les relations entre entités (par exemple : X est né à L , ou X est un patronyme et L un lieu). Une fois la liste des relations constituée, l'information est utilisée pour développer des cascades de grammaires ([Friburger et Maurel, 2004](#)) capables de décrire et aussi d'annoter les données biographiques.

3.7 Chaîne de traitement pour l'extraction de l'information à l'aide des grammaires locales

Étant donné un texte en entrée, nous énumérons une suite d'étapes (c.f. figure 3.6) afin d'appliquer une grammaire locale (possiblement en cascade) en vue d'effectuer une tâche d'extraction de l'information. Ces étapes, dont certaines facultatives, sont associées à plusieurs sous-problèmes traditionnels dans le traitement du langage naturel. Pour chaque problème, nous exposons également les approches les plus répandues qui permettent leur résolution.

1. Cette approche est à l'opposé de la constitution des corpus statiques provenant, par exemple, des banques de données

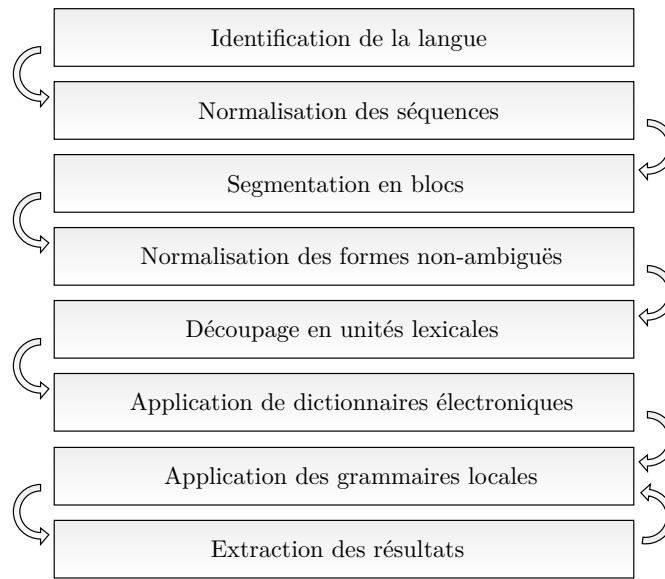


FIGURE 3.6 – Chaîne de traitement pour l'extraction d'information à l'aide des grammaires locales

- a) **Identification de la langue** : correspond à la tâche de déterminer la langue principale du texte : anglais, français, espagnol, grecque, etc. Plusieurs techniques peuvent être mise en œuvre : classification fondée sur les n-grammes de caractères (Cavnar et al., 1994), prédiction par reconnaissance partielle (PPM, *Prediction by Partial Matching*, Teahan et Harper, 2003), apprentissage profond par réseaux de neurones (*Deep Neural Networks*, Lopez-Moreno et al., 2014), etc.
- b) **Normalisation des séquences** : elle mobilise des règles d'insertion, de remplacement ou d'identification qui ont pour fonction, d'une part, d'insérer ou de remplacer des séquences de caractères par d'autres séquences et, d'autre part, de délimiter les séquences du texte qui ne seront pas prises en compte lors des analyses. Prenons comme exemple, les séparateurs (tabulations, sauts de ligne, etc.) ou les balises des langages de définition ou de description de documents (TEI, HTML, XML, etc.). Cette étape de normalisation comprend l'utilisation d'approches pour remplacer efficacement des motifs ou pour dépouiller la mise en forme d'un texte : expressions rationnelles Karttunen et al. (1996), translittération par des transducteurs finis Irvine et al. (2010), détection de la mise en forme en utilisant des caractéristiques textuelles peu profondes Kohlschütter et al. (2010), etc.
- c) **Segmentation en blocs** : elle consiste à découper le texte en blocs tels que des phrases en utilisant, par exemple, des modèles fondés sur des transducteurs finis (Friburger et al., 2000), ou sur l'entropie maximale (*MaxEnt Classifier*, Agarwal et al., 2005), ou encore sur l'apprentissage non-supervisé (Kiss et Strunk, 2006), etc.
- d) **Normalisation des formes non-ambiguës** : elle vise la normalisation des formes lexicales non-ambiguës. Cela consiste à remplacer les séquences du texte qui peuvent

être normalisées sans ambiguïté, par exemple, en anglais, « *she's* » par « *she is* » ou « *won't* » par « *will not* ». Cette étape peut mobiliser toute technique permettant de remplacer efficacement des séquences d'un texte, telles que celles fondées sur l'utilisation des transducteurs finis (Hassan et al., 2008) ou les techniques citées dans l'étape b).

- e) **Découpage en unités lexicales** : elle consiste à séparer et regrouper dans un index les séquences de lettres contiguës qui ont une signification conjointe. Cette opération est effectuée en utilisant les séparateurs d'unités lexicales adaptés à la langue du texte, par exemple, des espaces pour les langues indo-européennes ou d'autres critères pour traiter des langues agglutinantes. Cette étape peut faire appel à toute technique pour découper un texte en unités lexicales : découpage par des transducteurs finis (Paumier, 2003b), par construction d'arbres binaires complets (PAT-tries) au niveau des caractères Carpenter (2005), ou par des modèles statistiques Darwish et al. (2014), etc.
- f) **Application de dictionnaires électroniques** : elle consiste à rechercher l'association entre les unités lexicales du texte et les entrées lexicales des dictionnaires électroniques. Les dictionnaires contiennent des mots simples ou composés, associant à chaque entrée un lemme et des codes grammaticaux, sémantiques, flexionnels, ou n'importe quel autre code, relatif ou non à l'entrée. Cette étape peut mobiliser des approches permettant de stocker, charger, rechercher des entrées et appliquer efficacement un dictionnaire électronique à un texte : par exemple, via la construction d'un ou plusieurs automates acycliques minimaux (ADFA, *acyclic deterministic finite automaton*, Revuz, 1992, Watson, 2003), la construction d'index inversés Heinz et Zobel (2003), la construction de tables de hachage Lam et Huo (2005) ou encore la construction des RTN d'affixes (Berlocher et al., 2006) pour traiter les langues agglutinantes.
- g) **Application des grammaires locales** : correspond à l'application au texte d'une ou de plusieurs grammaires locales, possiblement en cascade. Le cœur de cette tâche est prise en charge par un analyseur syntaxique. Ce dernier permet non seulement de vérifier que le texte en entrée correspond aux règles définies par la grammaire locale, mais aussi de stocker de manière indexée les motifs repérés. Les algorithmes d'analyse syntaxique les plus utilisés dans cette étape sont ceux fondés sur la première recherche en profondeur (*top-down depth-first*) comme celui d'UNITEX (Paumier, 2003a) ou bien du type Earley utilisé, par exemple, dans OUTILEX (Blanc et al., 2006).
- h) **Extraction des résultats** : une fois les motifs repérés et indexés, il ne reste qu'à produire un résultat en sortie. En suivant la finalité pour laquelle la grammaire a été développée, la sortie peut, par exemple, prendre la forme de concordances, d'arbres d'analyse syntaxique, du texte en entrée muni des balises correspondantes aux motifs repérés, ou même de fichiers répertoriant les positions et, alternativement, les noms balises associés aux motifs.

Chacune des étapes listées précédemment doit être considérée d'un point de vue

général toute en prenant en compte la nature des textes à analyser. De ce fait, certaines sont facultatives, d'autres peuvent disparaître ou être fusionnées, ainsi que de nouvelles étapes, non énumérées, peuvent apparaître.

Par exemple, par rapport l'étape a) d'*identification de la langue*, elle est normalement facultative, les grammaires locales étant développées pour décrire un sous-langage appartenant à une langue ciblée, il est inhabituel de passer par une telle étape. Ceci étant dit, dans une tâche d'extraction d'information orientée à traiter plusieurs langues (une à la fois), des grammaires locales peuvent être conçues pour chaque une d'entre elles et alors reconnaître à l'avance la langue visée s'avère indispensable.

Pour ce qui est une éventuelle fusion d'étapes, nous pouvons citer le cas du traitement de l'arabe ou des langues agglutinantes. En effet, les étapes e) du *découpage du texte en unités lexicales* et g) d'*application de dictionnaires électroniques* sont parfois fusionnées car interdépendantes : les limites de chaque unité lexicale peuvent dépendre des limites des unités voisines, mais aussi des informations trouvées dans le dictionnaire.

Finalement, concernant des étapes qui peuvent apparaître, nous pouvons citer par exemple la *désambiguïsation des catégories grammaticales*¹ : elle consiste à sélectionner une seule catégorie grammaticale parmi celles associées à une unité lexicale par l'application de dictionnaires (après l'étape f). La désambiguïsation des catégories grammaticales peut utiliser toute technique permettant d'effectuer l'étiquetage morpho-syntaxique d'un texte : utilisation des grammaires par contraintes (Karlsson, 1990), utilisation de modèles de Markov cachés (HMMs, *Hidden Markov Models*, Church, 1988), utilisation des champs aléatoires conditionnels (CRFs, *Conditional Random Fields*, Lafferty et al., 2001), utilisation des machines à vecteurs de support (SVMs, *Support Vector Machines*, Nietzjo, 2002), utilisation des approches fondées sur l'apprentissage profond par réseaux de neurones (Collobert et Weston, 2008, Andor et al., 2016) ou bien des approches hybrides combinant des approches symboliques et fondées sur l'apprentissage automatique (Sigogne, 2010).

3.8 Autres travaux autour des grammaires locales

Dans Calberg (2003) un modèle de génération morphologique à deux niveaux fondé sur des grammaires locales est utilisé pour le traitement du finnois. Au premier niveau, les morphèmes sont décomposés par des graphes indépendants permettant de découper les entrées du lexique en unités de base et de leur associer des informations sémantiques. Au deuxième niveau, des graphes dits de *linéarisation* sont appliqués sur les résultats précédents afin de produire des unités lexicales valides.

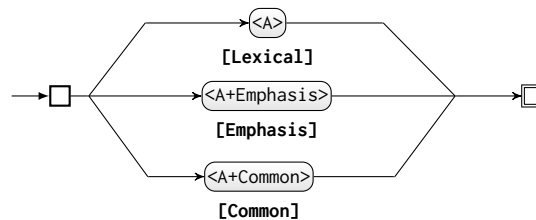
1. Observons que pour l'application des grammaires locales, que la désambiguïsation des catégories grammaticales est normalement absente, autrement dit, les grammaires sont souvent appliquées à des textes dont presque aucune ambiguïté lexicale n'a été levée. En effet, une des particularités de l'approche fondée sur l'utilisation des grammaires locales est de permettre un traitement du texte même en présence d'ambiguïtés lexicales.

Dans [Monnier et al. \(2003\)](#), les grammaires locales sont utilisées pour introduire dans les textes des marqueurs destinés à l'étude de la [prosodie](#) de la langue anglaise. Le travail consiste d'abord à classer la prosodie en 4 niveaux : l'accent tonique (ou *stress* en anglais), l'intonation, les frontières prosodiques ([Bolinger, 1986](#), citée par [Monnier et al.](#)) et l'intensification énonciative. Ensuite, des dictionnaires électroniques et des grammaires locales sont créés pour identifier et marquer les structures prosodiques selon les niveaux précités. Finalement, à l'aide d'un script de post-traitement, une mise en correspondance entre les marqueurs trouvés et des paramètres acoustiques est effectuée pour générer un texte susceptible d'être interprété par un logiciel de synthèse vocale nommé KALI¹.

Concernant les dictionnaires, il s'agit des dictionnaires électroniques du [LADL](#) pour l'anglais ([Klarsfeld et Carthy, 1991](#), [Monceaux, 1995](#), [Chrobot et al., 1999](#)) qui ont été enrichis par l'ajout de noms propres, de néologismes et de toute autre nouvelle entrée porteuse d'informations prosodiques. En outre, les anciennes entrées comme les nouvelles ont été munies de traits supplémentaires. Nous présentons ci-après des exemples d'entrées enrichies :

- (10) **carnival**, .N+1:s
 (11) **extremely**, .ADV+Emphasis

Dans l'exemple 10, le trait +1 est créé pour indiquer que dans *carnival* ['kɑrnəvəl] la syllabe qui porte l'accent tonique est la première. Dans l'exemple 11, le trait +Emphasis est utilisé comme un *marqueur d'intensité*. D'autres traits sont également créés pour indiquer des caractéristiques prosodiques accidentels.



Graphe 3.11 – Grammaire (simplifiée) pour ajouter des marqueurs prosodiques aux adjectifs ([Monnier et al., 2003](#), p. 1139)

En ce qui concerne les grammaires locales, il est tout d'abord question de construire un graphe pour chaque catégorie syntaxique, par exemple comme le graphe 3.11 pour les adjectifs. En faisant référence aux entrées des dictionnaires, ces graphes sont utilisés pour étiqueter, au niveau des mots, toute occurrence d'une catégorie syntaxique. Ensuite, au niveau des phrases, des structures plus complexes comme les propositions incises sont décrites à l'aide de graphes d'un niveau supérieur, c'est-à-dire des graphes

1. KALI ([Morel et Lacheret-Dujour, 2001](#)) est un logiciel payant de synthèse vocale développé par le Laboratoire Crisco de l'Université de Caen.

composées de plusieurs sous-graphes, qui serviront à ajouter davantage de marqueurs prosodiques dans le texte.

Dans le cadre d'une étude linguistique des emplois des verbes *donner* et *passer* dans un corpus portant sur le football, [Gasiglia \(2004\)](#) fait appel à CORDIAL et UNITEX. Les deux outils ont été utilisés conjointement pour extraire les occurrences des verbes et ainsi permettre d'analyser leurs structure syntaxiques et la nature de leurs arguments.

Dans [Blanc et Dister \(2004\)](#) est présenté une description des automates lexicaux avec structures de traits, c'est-à-dire des automates sur les mots dans lesquels les transitions peuvent être étiquetées par des masques lexicaux. Dans cette étude, une nouvelle représentation des masques lexicaux est utilisée afin d'adapter les algorithmes généraux sur les automates finis aux automates lexicaux. La démarche entreprise permet alors d'effectuer efficacement des opérations classiques sur les automates telles que la détermination, la minimisation, l'intersection ou le complément sur les automates lexicaux. À partir d'un état qui comporte une transition sortante étiquetée par un masque lexical, la technique qui permet l'ajustement des algorithmes réside à *découper les transitions de telle sorte que les ensembles décrits soient deux à deux disjoints ou égaux*. En suivant cette stratégie, le nombre d'opérations de comparaison et de découpage à réaliser peut être, dans le pire des cas, exponentiel par rapport au nombre total de transitions. Ceci est à prendre en compte lors de la construction des automates, mais pas lors de leur application.

3.9 Défis liés à la gestion et construction des grammaires locales

3.9.1 Gestion des grammaires

Afin de gérer le volume important des grammaires locales nécessaires pour les applications du TAL et de faciliter leur diffusion, accessibilité et partage, [Constant \(2004\)](#) proposait la création d'un outil de gestion de graphes sous la forme d'une bibliothèque en-ligne. Cette plateforme visait d'une part, l'hébergement des catalogues des grammaires documentées, d'autre part, la possibilité de rechercher ces grammaires en utilisant des filtres sur les composants ou sur les métadonnées indexées.

Dans [Constant \(2007\)](#), un tel système de partage de grammaires locales nommé **GRAAL** est présenté. Il est constitué par un ensemble de dépôts décentralisés, composés de paquets comprenant des métadonnées et une collection de grammaires. Une interface visuelle en-ligne appelée **GRAALWEB**, permet d'avoir un accès centralisé aux informations ainsi qu'à des fonctions d'exploration et de recherche des grammaires.

Bien que l'initiative de *Graal* avait une visée essentiellement pratique, dans les faits, peu d'utilisateurs ont partagé leurs grammaires au moyen du système, moins encore de dépôts de grammaires ont vu le jour. De surcroît, la décision des principaux navigateurs web (Firefox version 52, Chrome version 35, Opera version 37, entre autres) de limiter la technologie pour exécuter des applets Java, rendra inaccessible l'accès à

*GraalWeb*¹, tout au moins dans son implémentation actuelle.

3.9.2 Génération automatique de grammaires locales

Traboulsi (2005) décrit le prototype d'un système appelé *Local Grammar Finder* (LGFINDER) pour la construction semi-automatique de grammaires locales visant l'identification d'entités nommées dans un corpus de textes d'actualité financière en anglais. À partir d'un corpus de texte non étiqueté, la méthode est divisée en trois étapes (voir figure 3.7) : 1. Identifier la fréquence de mots lexicaux (noms, verbes, adjectifs, adverbes) ; 2. Analyser les collocations dans le texte afin de trouver les plus fréquentes et 3. Générer des grammaires locales à partir du traitement des concordances de chacune des collocations trouvées.

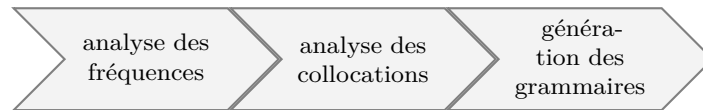


FIGURE 3.7 – Méthode LGFINDER pour la génération de grammaires locales

Après l'identification des collocations les plus fréquentes, le cœur de la méthode repose tout d'abord dans la génération d'une concordance à partir de chaque collocation. Ensuite, les mots de chaque concordance sont collectés et représentés sous la forme de vecteurs géométriques (\mathbb{R}^n , avec $n = 14$). Les 7 premières coordonnées des vecteurs sont respectivement codés selon un ensemble prédéfini de caractéristiques lexicales : mots des catégories ouvertes (*open-class words* – noms, verbes lexicaux, adjectifs et adverbes), mots de la classe fermée (*close-class words* – pronoms, déterminants, conjonctions et prépositions), mots hors dictionnaire, nombres et signes de ponctuation. Les 7 autres servent à encoder les positions (normalisées) de chacune des caractéristiques lexicales mentionnées ci-dessus. Une fois constitués, les vecteurs sont ordonnés selon leur fréquence et comparés en utilisant la distance euclidienne afin de les rapprocher, c'est-à-dire de regrouper les motifs linguistiques qui se ressemblent le plus. En dernier lieu, les groupes des motifs linguistiques les plus fréquentes servent à construire les grammaires locales à l'aide d'UNITEX.

En ce qui concerne la démarche implémentée par LGFINDER, nous pouvons remarquer deux limitations : En premier lieu, l'algorithme présenté pour regrouper les vecteurs (Traboulsi, p. 16), recherche les vecteurs les plus proches de chaque vecteur et a une complexité en temps quadratique $\mathcal{O}(n^2)$. En effet, étant donné n vecteurs, on a $n \cdot \frac{(n-1)}{2}$ paires de vecteurs et autant de distances euclidiennes à calculer. Enfin, lors de la dernière étape, une intervention manuelle est requise pour définir les catégories sémantiques correctes (patronymes, organisations, toponymes, etc) des motifs linguistiques codés par les vecteurs. En somme, ces deux aspects font obstacle

1. *GraalWeb* est disponible sur http://igm.univ-mlv.fr/~mconstan/library/index_graalweb.html

à l'analyse de grands volumes de textes et à la mise en pratique d'un algorithme de génération complètement automatique de grammaires locales.

3.9.3 Limitations

Toutes les approches qui utilisent les grammaires locales fournissent un taux de précision satisfaisant. Cependant, il reste difficile de construire des grammaires locales capables de capter toutes les combinaisons possibles et de faire face à la complexité du traitement des variations langagières attendues et inattendues : mots inconnus, omis ou mal orthographiés, violations des règles de grammaire standard.

Plusieurs auteurs ont évoqué les limites d'utilisation, en termes d'ergonomie, performance et qualité, des grammaires locales.

Selon [Friburger \(2002\)](#), la principale difficulté découle de l'incomplétude des grammaires locales, des ambiguïtés et de l'absence éventuelle de contextes dans les phrases. L'incomplétude des grammaires est relevée aussi par [Nakamura \(2004\)](#) où il est question d'insertions imprévisibles (de motifs) qui ne sont pas prises en compte dans les graphes.

[Dister et al. \(2004\)](#) fait un même constat relatif aux motifs non reconnus « *tout simplement parce qu'ils sont mal orthographiés...* », ou comme souligné par [Fairon et Watrin \(2003\)](#), dans le cas des patronymes, simplement parce que les motifs à reconnaître présentent patronymes beaucoup de variations.

Le problème de contextes est également soulevé dans les travaux de [Sætre \(2003\)](#) relatifs au langage microbiologique, où il est complexe de décrire de motifs contenant de longues séquences. Pour sa part, [Kevers \(2006\)](#) se questionne sur la possibilité d'utiliser des contextes porteurs d'informations mais qui sont éparpillées dans toute une phrase ou paragraphe.

[Constant \(2003\)](#) souligne les difficultés rencontrées pour construire et maintenir des grammaires précises et complètes. La maintenance des graphes est aussi évoquée dans [Pirovani et Silva de Oliveira \(2015\)](#), qui se questionne sur la façon de créer, adapter et transformer des règles dans des graphes afin d'augmenter la performance de la reconnaissance. Par ailleurs, [Ezzat \(2014\)](#) insiste sur le coût de développement des ressources linguistiques nécessaires pour obtenir une couverture satisfaisante.

En outre, pour [Gasiglia \(2004\)](#) le problème réside en « *l'incapacité de discriminer les occurrences verbales des occurrences nominales* », autrement dit, de traiter les ambiguïtés des catégories morphosyntaxiques des entrées lexicales. Selon [Fairon \(1998\)](#), [Calberg \(2003\)](#), [Poibeau et al. \(2003\)](#) il est aussi question des tâches de prétraitement, par exemple, il est indispensable de convertir du HTML en TXT ou prétraiter les textes pour avoir un résultat exploitable à travers les grammaires locales.

Dans le même sens, certaines tâches exigent des modules de post-traitement des résultats, dans [Monnier et al. \(2003\)](#), pour remplacer des marqueurs insérés par les grammaires avec des paramètres acoustiques définis dans un tableau. Dans [Kogkitsidou et Antoniadis \(2016\)](#) pour réaliser des transformations au niveau des caractères sur des

marqueurs insérés dans des messages SMS.

Le recours à des modules supplémentaires fait aussi partie des approches *hybrides* ou la sortie de la grammaire représente l'entrée d'un module fondé sur l'apprentissage automatique : [Watrín et al. \(2014\)](#), [Boujelben et al. \(2014\)](#), [Hkiri et al. \(2016\)](#)

Comme conséquence directe ou indirecte de ces limites, les tâches d'extraction de l'information à l'aide des grammaires locales, telle que la reconnaissance d'entités nommées, bute sur un plafond de performance. Il est devenu coûteux d'améliorer le taux de rappel, autrement dit, le pourcentage de motifs retrouvés au regard des motifs pertinents, des systèmes existants en ajoutant des règles supplémentaires aux grammaires locales ou en enrichissant les ressources linguistiques qu'elles mobilisent.

En d'autres termes, il est difficile d'augmenter la couverture d'une grammaire locale pour capturer davantage de combinaisons de motifs ou d'accroître les possibilités de traitement des variations attendues et inattendues dans un texte. Ceci résulte du fait de l'apparition des mots inconnus, ou des mots composés, pas nécessairement formés par de mots inconnus ([Maurel, 2004](#)), ou mal orthographiés ou encore dérogeant aux règles syntaxiques établies.

4

Grammaires locales étendues : principes

Dans le chapitre 3 nous avons étudié les grammaires locales et leur pertinence pour traiter des problèmes liés au traitement automatique des langues (TAL). Nous avons défini une grammaire locale (LG) comme un formalisme de description de règles syntaxiques ou sémantiques sous forme de graphe. Nous avons également étudié la représentation d'une grammaire locale par des réseaux de transitions récursifs (RTNs) qui peuvent être vus comme des grammaires non-contextuelles (CFGs) dans lesquelles on admet que les membres droits des règles soient des automates finis au lieu de simples séquences de symboles.

De plus, nous avons évoqué le fait que ces grammaires peuvent également reconnaître des langages contextuels (type 1) et sans restriction (type 0) en reconnaissant tout d'abord un langage régulier (type 3) ou non-contextuel (type 2) et en appliquant par la suite une opération pour exclure les séquences qui n'appartiennent pas au langage à reconnaître (0 ou 1).

Nous avons vu que le formalisme des LGs a été implémenté dans des outils ([Silberztein, 1994](#), [Paumier, 2003a](#), [Silberztein et Tutin, 2005](#), [Blanc et al., 2006](#)) destinés aux tâches du traitement automatique des langues telle que la recherche de motifs dans un texte qui a été éventuellement prétraité par des dictionnaires électroniques. Enfin, nous avons également soulevés certains inconvénients lors de leur utilisation.

Du point de vue de la mise en œuvre des grammaires locales, la possibilité de

simuler des grammaires non contextuelles est une conséquence de leur modélisation par des automates finis. En revanche, la gamme de contraintes représentables dépend de la mise en œuvre. Dans les outils disponibles, elle va d'une absence totale de contraintes jusqu'à un nombre prédéterminé de types de tests sur les variables : longueur, état (initialisée ou non), comparaison, etc, tous ceux-ci définis à l'avance et reconnus par l'analyseur de la grammaire.

Bien que la possibilité d'établir des contraintes sur les variables reste un atout indispensable pour augmenter le pouvoir de reconnaissance des grammaires locales, la non-standarisation des opérateurs parmi les outils disponibles, leur nombre réduit, le fait qu'il soit complexe de mettre en œuvre de nouvelles opérations sans modifier l'algorithme d'analyse syntaxique sous-jacent, ainsi que l'impossibilité de les paramétrer, de les adapter et de les modifier pour s'ajuster à différents types de problèmes, continuent à limiter la puissance de reconnaissance des grammaires locales. Entre autres, est en jeu la capacité d'une grammaire locale à conserver un bon niveau de précision, tout en améliorant leur rappel.

Dans ce chapitre nous introduisons la notion de grammaire locale étendue (ELG) comme une extension des grammaires locales classiques, plus particulièrement nous nous concentrons sur les grammaires locales qui sont représentées par des graphes syntaxiques et qui sont utilisés pour la recherche de motifs. Dans ce sens, il s'agit également d'un formalisme permettant de décrire formellement des ensembles de séquences grammaticales acceptées. Cependant, à la différence d'une grammaire locale, dans une grammaire locale étendue, il est possible d'associer aux étiquettes de sortie des transitions, des symboles non-terminaux qui représentent des appels à des actions conditionnelles, appelées *fonctions étendues*. Ces fonctions sont évaluées à l'extérieur de la grammaire, c'est-à-dire, en dehors de l'analyseur syntaxique.

Ainsi, alors qu'une grammaire locale permet uniquement de décrire des motifs syntaxiques ou sémantiques, une ELG peut en plus effectuer, au cours de l'analyse, des opérations arbitraires basées, par exemple, sur les motifs et sur l'état de l'analyse. Les fonctions étendues ont la possibilité de recevoir des paramètres, de réaliser des traitements, et de renvoyer comme résultat des symboles terminaux à concaténer à la sortie de la transition. Comme telles, les fonctions étendues peuvent être utilisées pour augmenter le pouvoir transformationnel des grammaires locales : repérer – en tolérant les fautes inattendues – des motifs dans un texte, puis les normaliser, les enrichir, les valider, les mettre en relation ou les baliser à la volée.

Au-delà de l'implémentation des grammaires traditionnelles, l'analyseur syntaxique d'une grammaire locale étendue peut être doté de la capacité de déclencher des événements, tel que la lecture d'un symbole d'entrée ou l'échec d'une reconnaissance. Ces actions peuvent également être implémentées sous la forme de fonctions étendues à l'extérieur de la grammaire locale, plus précisément, en dehors de la représentation des graphes.

Une telle possibilité implique un lien fort entre la modélisation des grammaires par des automates finis et l'analyseur syntaxique. Cependant, nous montrons comment ce lien peut à la fois garder une forte cohésion et se caractériser par un faible couplage.

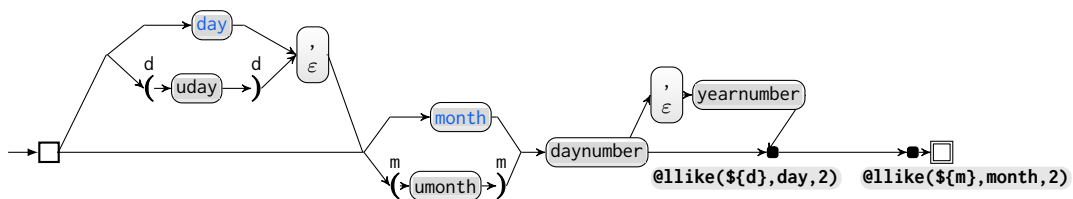
Autrement dit, l'analyseur peut considérer les fonctions étendues, dont les événements, comme des opérations abstraites et arbitraires qui n'ont pas besoin d'être définies à l'avance et qui ne changent pas la logique d'analyse.

Par exemple, les fonctions étendues mises en œuvre peuvent faire référence à des opérations sur les unités lexicales, si l'unité lexicale est 'book', nous avons des opérations comme '@reverse(book)' = 'koob', '@length(book)' = '5', '@equal(book,koob)' = false, '@upper(book)' = 'BOOK'; sur les unités lexicales numériques comme '@greater(10,2)' = true, 'times(2,3)' = 6; sur des listes d'unités lexicales comme '@front(book;koob)' = book, '@push(book;koob,BOOK)' = book;koob;BOOK, '@get(book;koob;BOOK,2)' = koob, sur les masques lexicaux '@isverb(<manger.V>)' = true, '@similarity(Bordaux, <N+Toponym>, 0.9)' ainsi que les positions particulières (absolues et relatives) de la séquence d'entrée '@begin()', '@end()', '@seek()'.

Parmi les avantages des grammaires locales étendues on peut citer la possibilité de réutiliser des fonctions externes à la grammaire afin d'améliorer ou agrandir les capacités de reconnaissance. Ces fonctions externes peuvent mettre en œuvre des approches hybrides au lieu de se limiter à des stratégies fondées sur des règles. Finalement, les fonctions étendues peuvent être organisées dans des bibliothèques réutilisables et partagées pour être intégrées à d'autres grammaires.

4.1 Aperçu général

Le graphe 4.1 représente une grammaire locale étendue pour la reconnaissance de dates. La fonction étendue `llike` (*looks like*) vérifie si une séquence inconnue, stockée dans la variable `d`, est similaire (avec un seuil maximal de 2) à un nom de jour, de même, elle teste, dans la deuxième appelle, si la séquence stockée sur la variable `m` est similaire à un nom de mois.



Graph 4.1 – Grammaire locale étendue pour la reconnaissance des dates

Des exemples de séquences reconnues par cette grammaire, colonne ELG, sont présentés dans le tableau 4.1.

SÉQUENCES	COGITO ¹	LG ²	TXRZR ³	STNER ⁴	ELG ⁵	VALIDE ? ⁶
March 14th	✓	✓	✓	✓	✓	✓
Friday, October 7, 1966	✓	✓	✓	✓	✓	✓
Dceember 21, 1985	×	×	×	✓	✓	✓

SÉQUENCES	COGITO	LG	TXRZR	STNER	ELG	VALIDE ?
Monda Yanary 15, 2007	×	×	✓	×	✓	✓
Jueves, Septiembre 30, 2004	×	×	×	×	×	✓
Championships 2nd 2004	×	×	×	×	×	×

TABLEAU 4.1 – Exemples de dates reconnues par différents systèmes et par une ELG

L'implémentation de la fonction étendue `llike`, et par conséquent de la notion de ressemblance, est arbitraire et peut dépendre de la nature du problème à traiter. Nous pouvons définir, par exemple, comme présentée plus bas, qu'une séquence ressemble au nom d'un jour si la distance d'édition (Levenshtein, 1966) entre le nom d'un jour et la séquence est inférieure ou égal 2. Par exemple, $\text{distance}(\text{Wednesday}, \text{Wednesdy}) = 1$ mais $\text{distance}(\text{Wednesday}, \text{Wedgelize}) = 5$.

```
function llike(input,entity,threshold)
  if input == nil then return true end
  if entity == 'month' then
    if levenshtein({'January','February',...}, input) <= threshold then
      return true
    end
  elseif entity == 'day' then
    if levenshtein({'Sunday','Monday',...}, input) <= threshold then
      return true
    end
  end
end
end
```

La fonction étendue `llike` utilise une approche naïve pour rechercher approximativement un motif `p` (`input`) dans une liste `W` (e.g. 'January', 'February',...). Des autres approches sont plus adéquates à mettre en œuvre. Cependant, ce qui est important à retenir est que l'analyseur syntaxique ne connaît pas à l'avance cette fonction étendue et n'est pas responsable de son évaluation. Cela implique que la fonction peut être modifiée ou améliorée sans changer l'algorithme d'analyse syntaxique, et même sans apporter des changements au graphe graphe 4.1, et donc sans le recompiler.

Le graphe 4.2 illustre un autre exemple de l'utilisation des grammaires locales étendues, il s'agit de la reconnaissance des expressions du type :

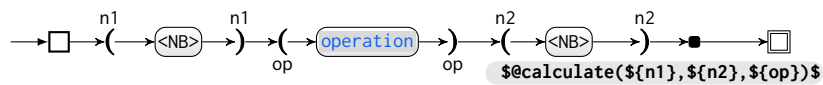
- *2 fois 13*
- *16 divisé par 4*

6. Cogito demo API : <http://cogitoapi.com/demo,context:kernel>.
6. Grammaire locale classique
6. TextRazor demo : <https://www.textrazor.com/demo>
6. Stanford NER demo : <http://nlp.stanford.edu:8080/ner>, classifier : english.muc.7class.
6. Grammaire locale étendu

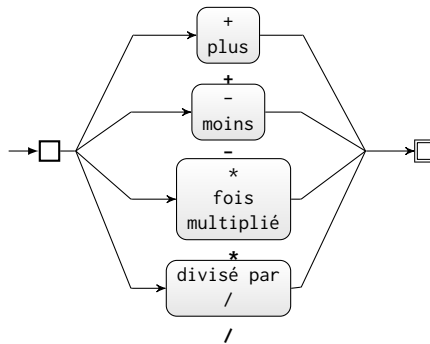
- 25 moins 15
- 45 + 32

Afin de produire des sorties comme :

- 2 fois 13 = 26
- 16 divisé par 4 = 4
- 25 moins 15 = 10
- 45 + 32 = 77



(a) graphe principal



(b) sous-graphe operation

Graphe 4.2 – Calculatrice

Pour parvenir à produire ce résultat, la fonction étendue `calculate` est utilisée, un exemple de sa mise en œuvre est illustrée ci-après :

```
function calculate (op1, op2, op)
  if not uMatch.start_with_space() then return false end
  if op == "+" then return tostring(op1 + op2)
  elseif op == "-" then return tostring(op1 - op2)
  elseif op == "*" then return tostring(op1 * op2)
  elseif op == "/" then return tostring(op1 / op2)
  else return false
end
end
```

Dans l'exemple `uMatch.start_with_space()` fait partie des opérations accessibles aux fonctions étendues pour connaître l'état de l'analyse, dans le cas présent, il est requis que le premier *token* du motif qui viens d'être reconnu se trouve au début de l'entrée ou après l'espace.

4.2 Principes

En premier lieu, une grammaire locale étendue inclut les caractéristiques classiques d'une grammaire locale traditionnelle ¹ : appel à des sous-graphes, utilisation de variables, de contextes, de filtres morphologiques, de masques lexicaux (Gross, 1993, 1997, Silberztein, 2003, Silberztein et Tutin, 2005, Paumier, 2016), etc.

De plus, une grammaire locale étendue enrichit le modèle des grammaires locales par un nouveau formalisme qui permet d'ajouter, par des transitions étiquetées, des fonctions conditionnelles qui sont externes à l'analyseur syntaxique. Ces fonctions permettent d'inclure ainsi, à la volée, dans les analyses effectuées, la combinaison d'une ou plusieurs approches, comme des calculs mathématiques, des manipulations de chaînes de caractères, des analyses statistiques, l'interrogation de bases de données, ou l'exploitation d'autres ressources utiles.

Dans leur représentation, le graphe est composé également par un groupe de boîtes (les transitions du diagramme d'états-transitions) reliées par des arcs avec une unique boîte désignée comme initiale (l'état initial du diagramme d'états-transitions) et une seule boîte désignée comme finale (l'état final du diagramme d'états-transitions), chaque boîte, à l'exception de celle associée à l'état final, est étiquetée en entrée (contenu de la boîte) et facultativement en sortie (contenu en dessous de la boîte).

Comme dans le cas des LGs, les étiquettes d'entrée peuvent contenir des mots, des méta-symboles, des traits sur les mots (contraintes lexicales sur les mots), le mot vide, l'appel à un sous-graphe (en indiquant son nom), etc.

Les étiquettes de sortie peuvent contenir des suites de symboles, comportant zéro ou plus sous-séquences de symboles non-terminaux toujours sous la forme $\varphi(\Delta)$ représentant chacun l'appel à une fonction externe à la grammaire. φ est le nom de la fonction étendue, Δ est un n -uplet (a_1, a_2, \dots, a_n) où chaque a_i est dénommé argument de φ . Un argument est soit une suite finie de symboles terminaux, désormais appelé argument littéral, soit un registre de la grammaire, désormais appelé argument variable.

L'entrée de la grammaire locale étendue est une séquence d'unités telle que des caractères ou des tokens. Lors de leur analyse, c'est-à-dire, l'application d'une ELG sur la séquence d'unités lexicales en entrée, une sous-séquence est reconnue si, en lisant le graphe de gauche à droite, il existe un chemin allant de l'état initial à l'état final où chacune des transitions parcourues reconnaît la sous-séquence présentée en entrée tout en satisfaisant les fonctions étendues rencontrées.

De la même manière que la notion de grammaire locale est comparable à celle d'un RTN, la notion de grammaire locale étendue que nous introduisons peut être rapprochée du modèle du réseaux de transitions augmenté (ATN, Woods, 1970, Bates, 1978). Le modèle d'ATN a été introduit indépendamment par Thorne et al. (1968,

1. Il est utile de rappeler que dans le périmètre de nos travaux, nous parlons des grammaires locales classiques en faisant référence au formalisme des graphes syntaxiques disponibles dans des outils comme UNITEX ou NOOJ

citée par Kaplan, 1972, p. 5) et généralisé par Woods (1970) comme une approche du traitement automatique du langage. Un ATN est une version augmentée du modèle des RTNs par la possibilité qu'offre un tel réseau d'ajouter 1) des registres, 2) des conditions aux arêtes et 3) d'associer à celles-ci des actions. Néanmoins, nous pouvons mettre en avant deux sortes de différences entre les ELG et les ATN. D'une part, on retrouve les mêmes différences que celles existant entre les grammaires locales et les RTNs :

- Les grammaires locales peuvent produire des sorties.
- Les grammaires locales peuvent faire référence à des ressources linguistiques.

D'autre part, on note des différences supplémentaires entre les ATN et les ELG :

- La notion de fonction étendue diffère de celle d'action dans les ATN, d'un part parce que les fonctions étendues peuvent communiquer de forme bidirectionnelle avec l'analyseur syntaxique et aussi entre elles, d'autre part, parce qu'elles peuvent retourner des résultats à concaténer à la sortie de la grammaire.
- Dans une ATN le cycle de vie des actions est limité à leur temps d'exécution, les fonctions étendues sont évaluées à chaque appel, mais peuvent conserver des registres globaux.
- Dans une ATN il n'est pas possible de faire face à des erreurs inattendues, par exemple, des phrases qui échouent à être analysées pour contenir un mot inconnu. En revanche, la notion de fonction étendue et l'ajout des événements permet aux grammaires locales étendues de mettre en place des techniques de correction locale au niveau de la lecture de symboles d'entrée ou des opérations appelées à partir du graphe.
- Les fonctions étendues sont évaluées par une machine abstraite et indépendantes de l'algorithme d'analyse syntaxique, autrement dit, il n'est pas nécessaire que l'analyseur connaisse à l'avance la fonction pour parvenir à la traiter.
- Les fonctions étendues ont accès aux mêmes ressources utilisées par l'analyseur syntaxique, tels que les dictionnaires morphologiques ou les tokens et peuvent les consulter et les modifier. En outre, les fonctions étendues ont aussi accès à des ressources externes, tels que des fichiers ou des bases de données.

4.3 Préliminaires

En plus des notions décrites dans l'annexe A et des définitions données au chapitre 2, nous présentons quelques définitions et conventions supplémentaires que nous utilisons dans la description des ELGs.

Symboles

Nous avons défini un alphabet (cf. définition A.25) comme un ensemble fini et *non-vide* d'éléments appelés symboles. Dans la présentation des ELGs, nous distinguons deux classes de symboles appartenant à un même alphabet :

1. **des symboles terminaux**, et
2. **des symboles non-terminaux**

Alphabets

Nous considérons un alphabet d'entrée (Σ) et un alphabet de sortie Γ . Chacun des alphabets est constitué par deux sous-alphabets disjoints, l'un de symboles terminaux, l'autre de symboles non-terminaux.

Alphabet d'entrée

L'alphabet d'entrée Σ d'une ELG est constitué par l'union de deux sous-alphabets disjoints :

1. $\bar{\Sigma}$: l'alphabet des symboles **terminaux** d'entrée, et
2. Ω : l'alphabet des symboles **non-terminaux** d'entrée.

Nous symbolisons par Σ_ε l'union entre l'alphabet d'entrée Σ et un ensemble ne contenant que le mot vide $\{\varepsilon\}$, soit $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.

Alphabet de sortie

L'alphabet d'entrée Γ d'une ELG est constitué par l'union de deux sous-alphabets disjoints :

1. $\bar{\Gamma}$: l'alphabet des symboles **terminaux** de sortie, et
2. Φ : l'alphabet des symboles **non-terminaux** de sortie, aussi appelé **alphabet étendu**.

Il est utile de rappeler que nous symbolisons par Γ_ε l'union entre l'alphabet de sortie Γ et un ensemble ne contenant que le mot vide $\{\varepsilon\}$, soit $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$.

Enfin, le tableau 4.2 résume les conventions pour représenter les symboles des alphabets des ELGs.

ALPHABET	SYMBOLES	MOTS
Σ	$\{\sigma_1, \sigma_2, \dots, \sigma_{ \Sigma }\}$	
$\bar{\Sigma}$	$\{\bar{\sigma}_1, \bar{\sigma}_2, \dots, \bar{\sigma}_{ \Sigma }\}$	
Ω	$\{\varrho_1, \varrho_2, \dots, \varrho_{ \Omega }\}$	
Γ	$\{\gamma_1, \gamma_2, \dots, \gamma_{ \Gamma }\}$	
$\bar{\Gamma}$	$\{\bar{\gamma}_1, \bar{\gamma}_2, \dots, \bar{\gamma}_{ \bar{\Gamma} }\}$	
Φ	$\{\phi_1, \phi_2, \dots, \phi_{ \Phi }\}$	

TABLEAU 4.2 – Conventions pour représenter les symboles des alphabets des ELGs

Étiquettes

Au chapitre 3 nous avons vu que dans une LG, les étiquettes des transitions, en plus des symboles terminaux, peuvent contenir des symboles non-terminaux. Dans le cas des ELG que nous présentons par la suite, les **étiquettes de sortie** des transitions peuvent aussi être constituées de symboles terminaux et non-terminaux (vus comme symboles terminaux au regard des opérations sur les automates).

Machines abstraites

Nous définissons une machine abstraite \mathcal{M} comme un 5-uplet comprenant 1) des registres permettant de stocker des valeurs 2) de constantes 3) de variables et 4) un état global, et 5) des opérations. Une opération reçoit des **entrées**, aussi appelées **variables**, et retourne des **sorties**. Plus précisément, une opération est une fonction de X dans Y qui définit une relation entrée-sortie en permettant de faire correspondre aux entrées ($x \in X$) des sorties ($y \in Y$) selon une règle ou un algorithme.

4.4 Modélisation

Une **grammaire locale étendue** définie sur un alphabet d'entrée $\Sigma = \bar{\Sigma} \cup \Omega$, tel que $\bar{\Sigma} \cap \Omega = \emptyset$, et un alphabet de sortie $\Gamma = \bar{\Gamma} \cup \Phi$, tel que $\bar{\Gamma} \cap \Phi = \emptyset$, est un n -uplet $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, \psi, \theta, \mathfrak{X}, q_0, F)$, où :

1. Q est un ensemble fini d'**états**,
2. Σ est l'alphabet d'**entrée**,
3. Γ est l'alphabet de **sortie**,
4. $\delta : Q \times (\bar{\Sigma} \cup \Omega) \times (\bar{\Gamma} \cup \Phi) \rightarrow \mathcal{P}(Q \times (\bar{\Gamma} \cup \Phi))$ est la **fonction de transition**,
5. \mathfrak{X} est le symbole d'échec,
6. $q_0 \in Q$ est l'**état initial**, et
7. $F \subseteq Q$ est l'ensemble des **états finaux**.

Fonction de transition

Rappelons que la fonction de transition d'un automate fini (FSA) décrit comment un automate passe à un autre état en « lisant » un symbole de l'alphabet. Dans les automates finis non-déterministes avec ε -transitions (ε -NFAs), la fonction de transition δ est égale à :

$$Q \times \Sigma_\varepsilon \longrightarrow \wp(Q)$$

Les arguments de la fonction transition sont ainsi définis par un état de départ ($q_i \in Q$) et par un symbole d'entrée égal au mot vide ε ou à un symbole σ de l'alphabet d'entrée ($\sigma \in \Sigma$), donc par une paire ordonnée de la forme (q_i, ε) ou (q_i, σ) , tel que :

1. Si c'est (q_i, ε) , la transition est spontanée, autrement dit, qu'elle fait passer l'automate de l'état q_i dans un autre état sans qu'un symbole d'entrée ne soit consommé.
2. Si c'est $(q_i, \bar{\sigma})$, la transition est réalisée en consommant le symbole d'entrée terminal $\bar{\sigma}$ ¹.
3. Si c'est (q_i, ϱ) , la transition est effectuée en consommant, ou pas, un symbole d'entrée, ceci dépend de la sémantique associée au symbole d'entrée non-terminal ϱ ².

En outre, étant donnée (q_i, σ) , la fonction de transition δ renvoie un ensemble de parties (cf. définition A.7) de Q , donc des états d'arrivée parmi les états de Q ou l'ensemble vide \emptyset .

Deux classes d'arguments de la fonction de transition sont alors possibles :

- a. (q_i, ϕ)
- b. (ε, ϕ)

Notons que les cas 2.a (ε -cycle) n'est pas accepté. En effet, le cas $\varepsilon : \phi$ signifierait qu'il est possible d'appeler infiniment une fonction sans avoir lu d'entrée.

De même, observons que la sortie de la fonction de transition est un ensemble de paires ordonnées de la forme (q_j, y) de $\wp(Q \times \Gamma_\varepsilon)$. Ceci indique qu'étant donnée une entrée (q_i, a, x) , la fonction de transition peut définir en sortie : 1. plusieurs états d'arrivée parmi les états de Q , 2. pour tout état d'arrivée $q_j \in Q$, un symbole $y \in \Gamma_\varepsilon$ à mettre à jour au sommet de pile lorsque le changement d'état est réalisé.

Par exemple, la règle de transition $\delta(q_i, a, x) = (q_j, y)$, précise que lorsque l'automate se trouve à l'état q_i et lit a sur l'entrée, il peut remplacer le symbole x au sommet de pile par y et passer dans l'état q_j . L'étiquette d'une telle transition est notée $a, x \rightarrow y$ et est illustré à la figure 2.8.

1. Notons que $\bar{\sigma} \in \bar{\Sigma}$ et $\bar{\Sigma} \subseteq \Sigma$
 2. Notons que $\varrho \in \Omega$ et $\Omega \subseteq \Sigma$

- $\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$. La fonction de transition δ reçoit un état Q et un symbole d'entrée Σ_ε et retourne un ensemble de parties $\mathcal{P}(Q)$,
- $\psi : Q \times \Sigma_\varepsilon \times Q \longrightarrow \Omega^*$. La fonction de sortie étendue ψ reçoit un état courant Q , un symbole d'entrée Σ_ε et un état suivant Q et retourne une séquence finie de symboles, appelée sortie étendue, de l'alphabet étendu $\Omega = \Phi_\varepsilon \cup \Gamma_\varepsilon$,
- $\mathcal{M} : \Omega^* \longrightarrow \Gamma^* \cup \{\mathfrak{X}\}$. \mathcal{M} est une machine abstraite qui prend en entrée une sortie étendue $\Omega^* = \{\Phi_\varepsilon \cup \Gamma_\varepsilon\}^*$ et qui retourne une suite finie de symboles de l'alphabet de sortie Γ_ε ou le symbole d'échec \mathfrak{X} ,
- $\theta : Q \times \Sigma_\varepsilon \times \mathcal{M}(\Omega^*) \times Q \longrightarrow \{\Sigma_\varepsilon \cup \Gamma_\varepsilon\}^*$. La fonction de sortie θ reçoit un état courant Q , un symbole d'entrée Σ_ε , la sortie de $\mathcal{M}(\Omega^*)$, et un état suivant Q . Elle retourne l'étiquette de sortie de la transition constituée d'une suite finie de symboles de l'alphabet résultant de l'union de l'alphabet de sortie et d'entrée $\{\Sigma_\varepsilon \cup \Gamma_\varepsilon\}^*$.

Les états (Q) de l'automate sont reliés par des arcs orientés représentant les transitions d'un état à un autre. Chaque transition $t = (p[t], li[t], f[t], lo[t], n[t]) \in E$ est ainsi un arc qui part d'un état source (ou état précédent p) vers un état destination (ou état postérieur n), avec une étiquette d'entrée $li[t]$, une fonction étendue $f[t]$, et une étiquette de sortie $lo[t]$ (cf. figure 4.1).

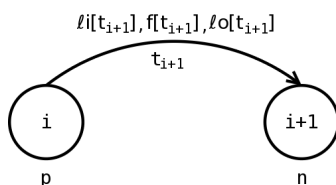


FIGURE 4.1 – Transition sur une grammaire locale étendue

Une transition peut être étiquetée en entrée soit par le symbole vide (ε), pour indiquer qu'elle ne « consomme » pas d'entrée ; soit par des symboles terminaux, par exemple, des mots (portions du texte en entrée), des méta-symboles, des traits sur les mots (contraintes lexicales sur les portions du texte en entrée), etc ; soit par le nom d'un autre automate (symbol non-terminal) dont l'état associé est l'état initial du sous-automate appelé (cf. figure 4.2). Dans le cas d'un symbole terminal, le critère de transition de i vers $i + 1$ est que le mot courant qui est analysé en entrée (x_i) satisfait l'étiquette d'entrée de la transition, par exemple si $li[ti+1] = red, < MIN >, < ADJ >$, un changement à l'état $i + 1$ sera possible si x_i est égal soit au mot « red », soit à un mot en minuscules (par exemple, « of »), soit à un adjectif (par exemple, « strong »). Si la transition est satisfaite, alors le mot x_i est consommé, s'il existe encore des séquences en entrée, le mot courant devient alors le mot suivant en entrée (x_{i+1}). Dans le cas d'un appel récursif, la transition ne consomme pas des séquences d'entrée et le critère de transition de i vers $i + 1$ dépendra alors de l'existence d'au moins un chemin accepteur dans le sous-automate appelé.

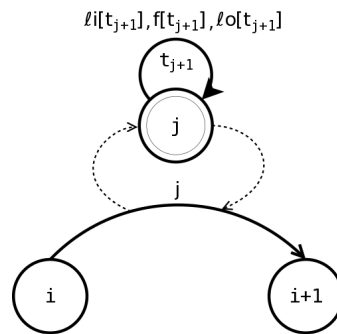


FIGURE 4.2 – Appel à un sous-graphe sur une grammaire locale étendue

Une fonction $f[t]$ est représentée par une paire « $i : o$ », i étant un ensemble de zéro ou plus paramètres d'entrée et $o = f(i)$ étant égal à la valeur qui renvoie la fonction après évaluation. Le critère d'acceptation de « f » est que la valeur retournée soit une sous-étiquette ($l[ft]$) différente du symbole d'échec (\mathfrak{X}) ou une valeur booléenne « vrai », cela implique que la fonction « attachée » à la transition t_i doit être satisfaite pour avancer à l'état i . Par convention, les fonctions sont associées aux transitions en écrivant « $@f(i)$ » où f est le nom de la fonction à appeler et i l'ensemble paramètres d'entrée (cf. figure 4.3).

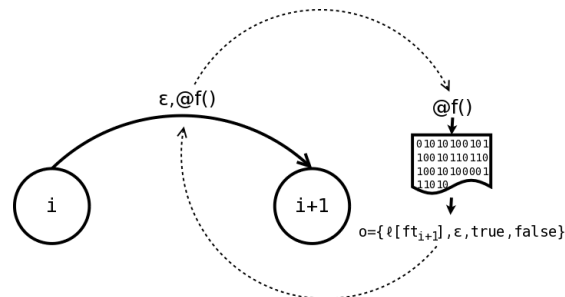


FIGURE 4.3 – Appel à une fonction sur une grammaire locale étendue

Dans la figure 4.3 la fonction $f[t_{i+1}] = @f(i)$ associée à la transition t_{i+1} est satisfaite si, après être évaluée, elle renvoie une valeur différente de \mathfrak{X} ou « faux ». En outre, la transition t_{i+1} est étiquetée en entrée par le symbole vide (ε), ce qui indique qu'elle ne consomme pas d'entrée.

Une transition peut être étiquetée en sortie, sous la forme d'une séquence de caractères, par la concaténation de $l[ft]$ et $l o[t]$, c'est-à-dire, par la sous étiquette retournée après évaluation de ft et l'étiquette de sortie associée à la transition t (cf. figure 4.4).

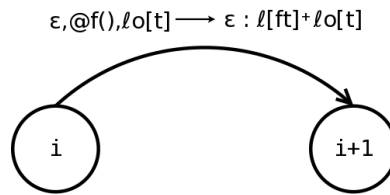


FIGURE 4.4 – Transition étiquetée en sortie sur une grammaire locale étendue

Un chemin de l'automate est une séquence de transitions $t_1, t_2, t_3 \dots t_n$ avec $n[t_i] = p[t_{i+1}]$ pour $i=1, \dots, n-1$ (cf. figure 4.5). L'étiquette d'entrée d'un chemin est le résultat de la concaténation des étiquettes d'entrée des transitions associées au chemin, soit $li[\pi] = li[t_1] \dots li[t_n]$. L'étiquette de sortie d'un chemin est le résultat de la concaténation des étiquettes de sortie des transitions associées au chemin, soit $lo[\pi] = lo[t_1] \dots lo[t_n]$.

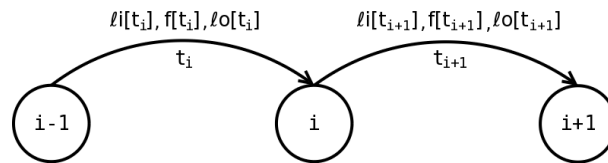


FIGURE 4.5 – Chemin sur une grammaire locale étendue

L'automate qui modélise la grammaire locale étendue débute dans un état initial (premier cercle en gras à gauche) et change d'état lorsque l'ensemble des conditions attachées à la transition t sont satisfaites. Si au cours de chaque changement l'automate vérifie toutes les conditions, il finira pour aboutir à l'état final (dernier double cercle à droite). Alors, un chemin accepteur $\pi = t_1, t_2, t_3 \dots t_n$ sera le chemin partant de l'état initial vers l'état final $f \in F$ (cf. figure 4.6).

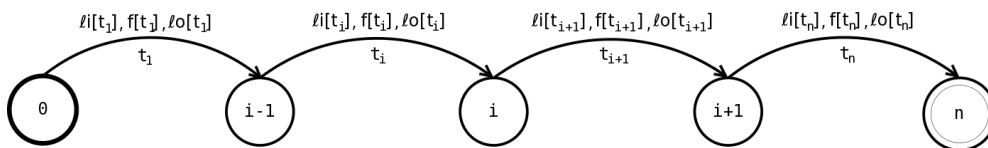


FIGURE 4.6 – Chemin accepteur sur une grammaire locale étendue

Une sous-séquence $x = u_1, u_2, \dots, u_n$ de caractères ou de tokens donnés en entrée de la grammaire, est acceptée par l'automate s'il existe un chemin accepteur π partant de s et qui est étiqueté comme $x : li[\pi] = x$ (cf. figure 4.7). Comme pour les grammaires locales, l'automate résultant est localement efficace mais ne satisfait pas, par exemple, un critère de minimisation globale.

Exemple :

1. $Q : \{q_0, q_1, q_2, q_3, q_4, q_5\}$
2. $\Sigma : \{aperçoit, Juliette, Roméo\}$
3. $\Phi : \{inc(), fail()\}$
4. $\Gamma : \{(\ , \ , a, b, c, \dots, z)\}$
5. $\delta :$

$\delta(q_0, \varepsilon) = q_1,$	$\delta(q_1, \langle \text{WORD} \rangle) = q_2,$	$\delta(q_1, \langle \text{TOKEN} \rangle) = q_3,$	$\delta(q_2, \varepsilon) = q_5,$
$\delta(q_3, \langle \text{TOKEN} \rangle) = q_4,$	$\delta(q_4, \varepsilon) = q_6,$	$\delta(q_6, \varepsilon) = q_5,$	
6. $\psi :$

$\psi(q_0, \varepsilon, q_1) = inc(),$	$\psi(q_2, \varepsilon, q_5) = inc(),$	$\psi(q_4, \varepsilon, q_6) = inc(),$	$\psi(q_6, \varepsilon, q_5) = fail(),$
--	--	--	---
7. $\theta : \varepsilon$
8. \mathfrak{X}
9. q_0
10. $F : \{q_5\}$

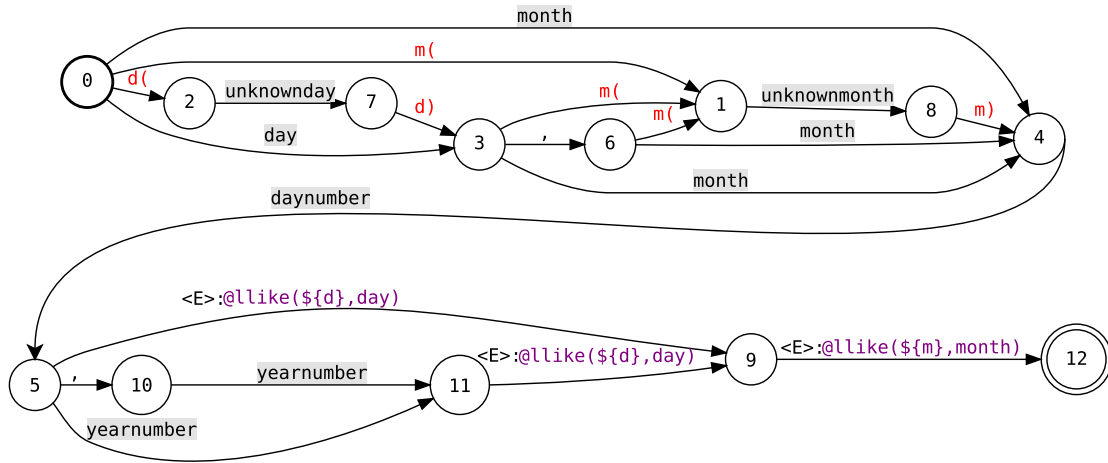


FIGURE 4.7 – Automate équivalent au graphe 4.1

4.5 Aplatissement

Une grammaire non contextuelle est récursive lorsque une variable à gauche d'une règle se réfère à une autre dans la partie à droite. La dérécursivation est une opération qui permet d'interdire les cycles et donc de transformer une grammaire locale en une machines à états finis. Une des techniques de dérécursivation est l'aplatissement, il consiste à remplacer une transition t étiquetée en entrée par un symbole non-terminal qui désigne l'appel à un sous-graphe par une copie exclusive du sous-graphe. L'automate résultant est obtenu en parcourant la grammaire à partir du graphe principal et, pour chaque étiquette non-terminale qui fait appel à un sous-graphe, en remplaçant récursivement la transition par le sous-graphe appelé. Lors du processus, il est possible d'aboutir à une dépendance cyclique parmi un sous-ensemble de transitions qui appellent un sous-graphe et boucler indéfiniment. La stratégie pour palier à ce problème peut être abordée de façon passive ou active :

- De manière passive, il est nécessaire de vérifier les dépendances cycliques avant l'aplatissement, si des dépendances cycliques sont détectées, alors produire une erreur. Cette démarche est par exemple l'implémentée dans OPENFST pour la commande *replace* en charge de l'aplatissement.
- De manière active, est possible de démarrer l'algorithme en fixant un seuil d'imbrication n au-delà duquel les appels à un sous-graphe sont tronqués et remplacés par des transitions vides. Dans les cas où le niveau maximal de récursion est atteint, l'automate résultant n'est pas strictement équivalent à la grammaire d'origine. Cette approche est aussi dénommée par *aplatissement contrôlé* (Paumier, 2003a, p. 138) et est utilisée dans UNITEX et OUTILEX pour les commandes *flatten* et *wrtn-flatten*.

4.6 Caractéristiques principales

En plus d'hériter des caractéristiques des LGs, les ELGs ont les caractéristiques principales suivantes :

1. Créer des conditions implicites et explicites sur les transitions.
2. Faire appel à des fonctions arbitraires.
3. Faire référence à des ressources externes.
4. Utiliser les sorties des fonctions pour générer des sorties.
5. Stocker des variables globales.
6. Utiliser des contextes éloignés.
7. Déclencher des événements.
8. Modifier la fenêtre d'analyse.
9. Consulter les dictionnaires morphologiques.

4.7 Évaluation des fonctions étendues

Rappelons qu'une fonction étendue φ reçoit un n -uplet d'arguments Δ et retourne, soit une suite finie de symboles γ de l'alphabet de sortie Γ_ε , soit le symbole d'échec \mathfrak{X} :

$$\varphi(\Delta) = \gamma \in \Gamma^* \cup \{\mathfrak{X}\}$$

En outre, dans la sous-section 2.2.2 nous avons défini comme non-déterministe tout automate fini qui à partir d'un symbole donné et d'un état de départ a comme

destination plusieurs états d'arrivée, ou dans lequel il est admis de changer d'état sans lire de symbole.

La faculté des grammaires locales de choisir quel sera le prochain état, ou de passer dans un autre d'état spontanément, implique aussi que dans une grammaire locale étendue, les fonctions qui sont attachées aux transitions doivent être évaluées selon un principe qui empêche la modification des registres associés à la fonction lorsque elle n'est pas satisfaite, soit quand $\varphi(\Delta) = \mathfrak{X}$.

Le principe qui assure que l'évaluation d'une fonction se réalise de façon « *tout ou rien* » peut s'assimiler à la propriété d'*atomicité* dans un système de gestion de base de données¹. En effet, dans ceux-ci, une opération, appelée transaction, est considérée comme une unité indivisible, de telle sorte que si elle ne s'achève pas, il est requis de laisser les données dans l'état précédent, celui antérieur à sa réalisation.

Ce principe est élargi à l'évaluation d'une série de fonctions étendues $\varphi_1, \varphi_2, \dots, \varphi_n$ qui sont attachés à une suite de transitions menant d'un état p à un état r . Ainsi, si une fonction φ_i n'est pas satisfaite, les modifications effectuées par φ_i , ainsi que par toute autre fonction précédente et satisfaite $\varphi_1, \varphi_2, \dots, \varphi_{i-1}$, ne doivent être :

1. ni prises en compte lors de une nouvelle évaluation de la fonction,
2. ni visibles lors de l'évaluation d'autres fonctions.

Comme conséquence, si φ_i n'est pas satisfaite, la série de changements $\varphi_{1,i}$ n'est pas prise en compte et les registres globaux sont restaurés tels qu'ils étaient à l'état p . Enfin, il est utile de rappeler que si $p \xrightarrow{\alpha_{i,j}} r$ n'est pas un chemin réussi, la série de fonctions satisfaites $\varphi_1, \varphi_2, \dots, \varphi_n$ est considérée comme incomplète et leur effets ne sont pas pris en compte. En d'autres termes, le principe d'*atomicité* implique que l'évaluation d'une série de fonctions étendues attachées à une suite de transitions menant d'un état p à un état r est complète si :

1. $\forall i \in n, \varphi_i(\Delta) \neq \mathfrak{X}$ (toutes les fonctions étendues sont satisfaites)
2. $p \xrightarrow{\forall i \in n, \varphi_i(\Delta) \neq \mathfrak{X}} r, p = q_0$ et $r \in F$ (la série de fonctions satisfaites se trouve dans un chemin réussi)

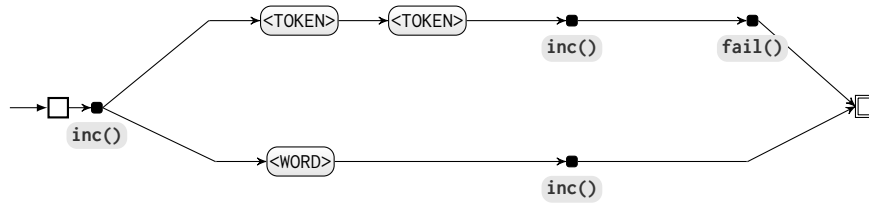
Afin d'illustrer l'*atomicité* de l'évaluation des fonctions étendues, considérons d'abord l'exemple 4.1 où ce principe n'est pas respecté.

1. En poursuivant le parallèle comparatif, nous pouvons dire, par exemple, qu'une propriété semblable à celle de l'*isolation* des bases de données, mais appliquée aux registres accessibles à travers les fonctions étendues, est garantie dès lors que toute évaluation simultanée est interdite, cette contrainte est respectée comme conséquence de la nature séquentielle de l'algorithme d'analyse de la grammaire. Par ailleurs, nous considérons qu'une propriété comme celle de la *durabilité* des transactions n'est proprement requise.

Exemple 4.1. Analysons la phrase d'exemple 12 :

(12) Roméo aperçoit Juliette

à l'aide de la ELG définie par le graphe 4.3 :



Graph 4.3 – Évaluation des fonctions étendues

du registre global :

- $i = 0$

et des fonctions étendues *inc* et *fail* définies comme :

- $\varepsilon \xrightarrow[i=i+1]{inc()} \varepsilon$ (ne reçois pas d'arguments, incrémente i de 1, retourne le symbole vide)
- $\varepsilon \xrightarrow[i=i+1]{fail()} \mathcal{X}$ (ne reçois pas d'arguments, incrémente i de 1, retourne le symbole d'échec¹)

Pour faciliter l'illustration, nous présentons également le graphe 4.3 sous la forme d'un automate fini :

$$\mathcal{L} = (Q, \Sigma, \Phi, \Gamma, \delta, \psi, \theta, \mathcal{X}, q_0, F),$$

1. $Q : \{q_0, q_1, q_2, q_3, q_4, q_5\}$
2. $\Sigma : \{aperçoit, Juliette, Roméo\}$
3. $\Phi : \{inc(), fail()\}$
4. $\Gamma : \{(), a, b, c, \dots, z\}$
5. $\delta :$

$$\frac{\delta(q_0, \varepsilon) = q_1, \quad \delta(q_1, \langle \text{WORD} \rangle) = q_2, \quad \delta(q_1, \langle \text{TOKEN} \rangle) = q_3, \quad \delta(q_2, \varepsilon) = q_5,}{\delta(q_3, \langle \text{TOKEN} \rangle) = q_4, \quad \delta(q_4, \varepsilon) = q_6, \quad \delta(q_6, \varepsilon) = q_5,}$$

6. $\psi :$

$$\psi(q_0, \varepsilon, q_1) = inc(), \quad \psi(q_2, \varepsilon, q_5) = inc(), \quad \psi(q_4, \varepsilon, q_6) = inc(), \quad \psi(q_6, \varepsilon, q_5) = fail(),$$

7. $\theta : \varepsilon$

1. Autrement dit, la fonction n'est pas satisfaite

8. \mathfrak{X}
9. q_0
10. $F : \{q_5\}$

dont la représentation sous forme de diagramme d'états-transitions est montrée à la figure 4.8 :

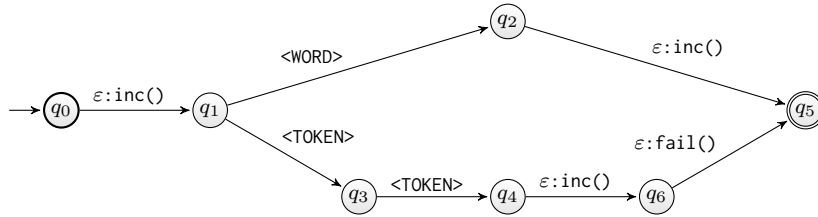


FIGURE 4.8 – Automate fini équivalent au graphe 4.3

Sachant que :

- $\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$. La fonction de transition δ reçoit un état Q et un symbole d'entrée Σ_ε et retourne un ensemble de parties $\mathcal{P}(Q)$,
- $\psi : Q \times \Sigma_\varepsilon \times Q \longrightarrow \Omega^*$. La fonction de sortie étendue ψ reçoit un état courant Q , un symbole d'entrée Σ_ε et un état suivant Q et retourne une séquence finie de symboles, appelée sortie étendue, de l'alphabet étendu $\Omega = \Phi_\varepsilon \cup \Gamma_\varepsilon$,
- $\mathcal{M} : \Omega^* \longrightarrow \Gamma^* \cup \{\mathfrak{X}\}$. \mathcal{M} est une machine abstraite qui prend en entrée une sortie étendue $\Omega^* = \{\Phi_\varepsilon \cup \Gamma_\varepsilon\}^*$ et qui retourne une suite finie de symboles de l'alphabet de sortie Γ_ε ou le symbole d'échec \mathfrak{X} ,
- $\theta : Q \times \Sigma_\varepsilon \times \mathcal{M}(\Omega^*) \times Q \longrightarrow \{\Sigma_\varepsilon \cup \Gamma_\varepsilon\}^*$. La fonction de sortie θ reçoit un état courant Q , un symbole d'entrée Σ_ε , la sortie de $\mathcal{M}(\Omega^*)$, et un état suivant Q . Elle retourne l'étiquette de sortie de la transition constituée d'une suite finie de symboles de l'alphabet résultant de l'union de l'alphabet de sortie et d'entrée $\{\Sigma_\varepsilon \cup \Gamma_\varepsilon\}^*$.

Nous débutons l'analyse de la séquence d'entrée $T = \{Roméo, aperçoit, Juliette\}$ en associant à chaque symbole de T , des éléments de l'ensemble $E = \{\langle \text{TOKEN} \rangle, \langle \text{WORD} \rangle\}$ selon la relation d'équivalence $\Sigma \longrightarrow \{\langle \text{TOKEN} \rangle, \langle \text{WORD} \rangle\}$. Nous obtenons alors que $E(T) = (\{\langle \text{TOKEN} \rangle, \langle \text{WORD} \rangle\}, \{\langle \text{TOKEN} \rangle, \langle \text{WORD} \rangle\}, \{\langle \text{TOKEN} \rangle, \langle \text{WORD} \rangle\})$.

À t_0 le registre global i est égal à 0 et la ELG se trouve à l'état initial q_0 . Nous lisons le premier symbole (*Roméo*) et consultons la fonction de transition δ . Le seul état de destination est définie par $\delta(q_0, \varepsilon) = q_1$, afin de vérifier s'il est possible de réaliser la transition vers q_1 , trois actions sont effectuées :

1. Consulter la fonction de sortie étendue ψ . Pour $\delta(q_0, \varepsilon) = q_1$ elle est définie comme $\psi(q_0, \varepsilon, q_1) = \text{inc}()$,

2. Évaluer la sortie étendue $\text{inc}()$. Puisque $\text{inc}() \in \Phi$ est une fonction étendue définie comme $\varepsilon \xrightarrow[i=i+1]{\text{inc}()} \varepsilon$, $\mathcal{M}(\text{inc}())$ incrémente le registre global i de 1 et retourne ε ,
3. Vérifier si la valeur retournée est différente du symbole d'échec \mathfrak{X} . Étant donné que $\varepsilon \neq \mathfrak{X}$, la fonction étendue est satisfaite et par conséquent la sortie étendue l'est aussi.

Ensuite, avant de réaliser la transition vers q_1 , sans consommer de symbole, nous consultons la fonction de sortie $\theta(q_0, \varepsilon, \varepsilon, q_1)$ qui renvoie ε , valeur utilisée comme étiquette de sortie de la transition.

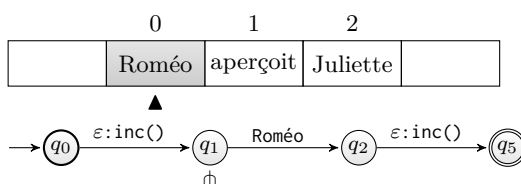
À présent nous sommes dans l'état q_1 , nous consultons à nouveau la fonction de transition δ . Deux transitions sont possibles $\delta(q_1, \text{Roméo}) = q_2$ et $\delta(q_1, \text{Roméo}) = q_3$. Nous analysons d'abord $\delta(q_1, \text{Roméo}) = q_2$ en appliquant les actions 1, 2 et 3 citées auparavant :

1. Consulter la fonction de sortie étendue ψ . Pour $\psi(q_1, \text{Roméo}, q_2)$, il n'existe pas de sortie étendue,
2. Évaluer la sortie étendue ε . La fonction étendue ε est définie comme $\varepsilon \xrightarrow{\varepsilon} \varepsilon$, $\mathcal{M}(\varepsilon)$ retourne alors ε ,
3. Vérifier si la valeur retournée est différente du symbole d'échec \mathfrak{X} . Étant donné que $\varepsilon \neq \mathfrak{X}$, la fonction étendue est satisfaite et par conséquent la sortie étendue l'est aussi.

Comme pour la transition $q_0 \xrightarrow{\varepsilon:\text{inc}()} q_1$, avant de réaliser la transition vers q_2 , en consommant le symbole *Roméo*, nous consultons la fonction de sortie $\theta(q_1, \text{Roméo}, \varepsilon, q_2)$ qui renvoie ε , valeur utilisée comme étiquette de sortie de la transition.

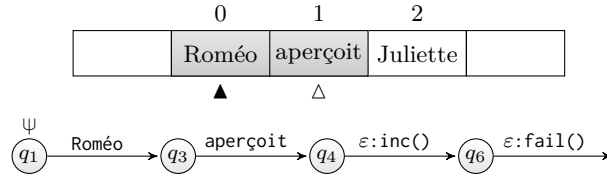
Un processus semblable se répète pour $q_2 \xrightarrow{\varepsilon:\text{inc}()} q_5$, qui incrémente le registre global i de 1 avant de passer en q_5 . Enfin, étant donné que $q_5 \in F$, la séquence *Roméo* est acceptée et la sortie ε est produite. Le parcours réalisé jusqu'à présent est résumé dans le schéma t_0 suivant :

$$t_0 : \begin{array}{l} q_0 \\ q_1 \\ q_2 \\ q_5 \in F \end{array} \left| \begin{array}{ll} \delta(q_0, \varepsilon) = q_1 & \psi(q_0, \varepsilon, q_1) = \text{inc}() \\ \delta(q_1, \text{Roméo}) = q_2 & \psi(q_1, \text{Roméo}, q_2) = \varepsilon \\ \delta(q_2, \varepsilon) = q_5 & \psi(q_2, \varepsilon, q_5) = \text{inc}() \end{array} \right. \begin{array}{ll} \mathcal{M}(\text{inc}()) = \varepsilon & \theta(q_0, \varepsilon, \varepsilon, q_1) = \varepsilon \\ \mathcal{M}(\varepsilon) = \varepsilon & \theta(q_1, \text{Roméo}, \varepsilon, q_2) = \varepsilon \\ \mathcal{M}(\text{inc}()) = \varepsilon & \theta(q_2, \varepsilon, \varepsilon, q_5) = \varepsilon \end{array} \left| \begin{array}{l} i = 1 \\ i = 2 \end{array} \right.$$



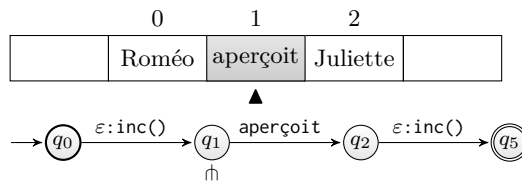
Une fois le parcours de t_0 achevé, en t_1 , il est nécessaire de continuer l'analyse de la fonction de transition $\delta(q_1, \text{Roméo}) = q_3$ que nous n'avons pas encore traitée. Nous procédons de la même manière pour les états q_3, q_4, q_6 de telle sorte que lorsque nous arrivons à $q_6 \xrightarrow{\varepsilon:\text{fail}()} q_5$, $\mathcal{M}(\text{fail}())^1$ incrémente le registre global i de 1 et retourne \mathfrak{X} . Étant donné que la valeur retournée est \mathfrak{X} , la fonction étendue n'est pas satisfaite, par conséquent la sortie étendue ne l'est pas non plus et la transition vers q_5 ne se réalise pas. La séquence *Roméo aperçoit* est alors rejetée. Le parcours réalisé est résumé dans le schéma t_1 suivant :

$$t_1 : \begin{array}{l} q_1 \\ q_3 \\ q_4 \\ q_6 \end{array} \left| \begin{array}{ll} \delta(q_1, \text{Roméo}) = q_3 & \psi(q_1, \text{Roméo}, q_3) = \varepsilon \\ \delta(q_3, \text{aperçoit}) = q_4 & \psi(q_3, \text{aperçoit}, q_4) = \varepsilon \\ \delta(q_4, \varepsilon) = q_6 & \psi(q_4, \varepsilon, q_6) = \text{inc}() \\ \delta(q_6, \varepsilon) = q_5 & \psi(q_6, \varepsilon, q_5) = \text{fail}() \end{array} \right. \begin{array}{ll} \mathcal{M}(\varepsilon) = \varepsilon & \theta(q_1, \text{Roméo}, \varepsilon, q_3) = \varepsilon \\ \mathcal{M}(\varepsilon) = \varepsilon & \theta(q_3, \text{aperçoit}, \varepsilon, q_4) = \varepsilon \\ \mathcal{M}(\text{inc}()) = \varepsilon & \theta(q_4, \varepsilon, \varepsilon, q_6) = \varepsilon \\ \mathcal{M}(\text{fail}()) = \mathfrak{X} & \theta(q_6, \varepsilon, \varepsilon, q_5) = \varepsilon \end{array} \left. \begin{array}{l} i = 3 \\ i = 4 \end{array} \right.$$



À la fin des parcours t_0 et t_1 , nous avons une séquence reconnue (*Roméo*), le registre global i est égal à 4 et il ne reste plus de transitions à examiner. Vu que l'analyse que nous effectuons se réalise avec le principe de fenêtre glissante, l'analyse reprend à partir du symbole *aperçoit*. Le parcours réalisé est résumé dans le schéma t_2 suivant :

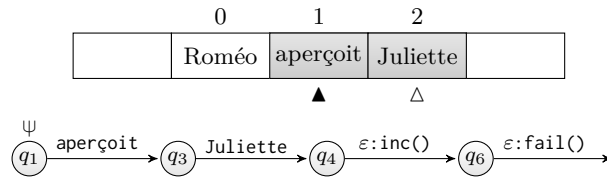
$$t_2 : \begin{array}{l} q_0 \\ q_1 \\ q_2 \\ q_5 \in F \end{array} \left| \begin{array}{ll} \delta(q_0, \varepsilon) = q_1 & \psi(q_0, \varepsilon, q_1) = \text{inc}() \\ \delta(q_1, \text{aperçoit}) = q_2 & \psi(q_1, \text{aperçoit}, q_2) = \varepsilon \\ \delta(q_2, \varepsilon) = q_5 & \psi(q_2, \varepsilon, q_5) = \text{inc}() \end{array} \right. \begin{array}{ll} \mathcal{M}(\text{inc}()) = \varepsilon & \theta(q_0, \varepsilon, \varepsilon, q_1) = \varepsilon \\ \mathcal{M}(\varepsilon) = \varepsilon & \theta(q_1, \text{aperçoit}, \varepsilon, q_2) = \varepsilon \\ \mathcal{M}(\text{inc}()) = \varepsilon & \theta(q_2, \varepsilon, \varepsilon, q_5) = \varepsilon \end{array} \left. \begin{array}{l} i = 5 \\ i = 6 \end{array} \right.$$



Une fois le parcours t_2 achevé, il est nécessaire de continuer une analyse qui aboutit au rejet de la séquence *aperçoit Juliette*. Ce parcours de rejet est résumé dans le schéma t_3 suivant :

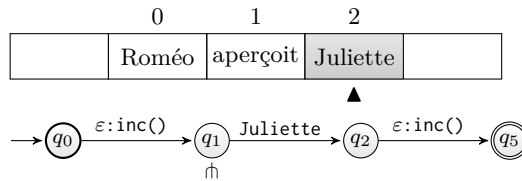
$$t_3 : \begin{array}{l} q_1 \\ q_3 \\ q_4 \\ q_6 \end{array} \left| \begin{array}{ll} \delta(q_1, \text{aperçoit}) = q_3 & \psi(q_1, \text{aperçoit}, q_3) = \varepsilon \\ \delta(q_3, \text{Juliette}) = q_4 & \psi(q_3, \text{Juliette}, q_4) = \varepsilon \\ \delta(q_4, \varepsilon) = q_6 & \psi(q_4, \varepsilon, q_6) = \text{inc}() \\ \delta(q_6, \varepsilon) = q_5 & \psi(q_6, \varepsilon, q_5) = \text{fail}() \end{array} \right. \begin{array}{ll} \mathcal{M}(\varepsilon) = \varepsilon & \theta(q_1, \text{aperçoit}, \varepsilon, q_3) = \varepsilon \\ \mathcal{M}(\varepsilon) = \varepsilon & \theta(q_3, \text{Juliette}, \varepsilon, q_4) = \varepsilon \\ \mathcal{M}(\text{inc}()) = \varepsilon & \theta(q_4, \varepsilon, \varepsilon, q_6) = \varepsilon \\ \mathcal{M}(\text{fail}()) = \mathfrak{X} & \theta(q_6, \varepsilon, \varepsilon, q_5) = \varepsilon \end{array} \left. \begin{array}{l} i = 7 \\ i = 8 \end{array} \right.$$

1. Rappelons que nous avons défini la fonction étendue $\text{fail}() \in \Phi$ comme $\varepsilon \xrightarrow[i=i+1]{\text{fail}()} \mathfrak{X}$.



À la fin des parcours t_2 et t_3 , nous avons une séquence supplémentaire reconnue (*aperçoit*), le registre global i est égal à 8 et il ne reste plus de transitions à examiner. Avec une fenêtre glissante, le processus reprend à partir du symbole *Juliette*. Le parcours réalisé est résumé dans le schéma t_4 suivant :

$$t_4 : \begin{array}{l} q_0 \\ q_1 \\ q_2 \\ q_5 \in F \end{array} \left| \begin{array}{ll} \delta(q_0, \varepsilon) = q_1 & \psi(q_0, \varepsilon, q_1) = inc() \\ \delta(q_1, Juliette) = q_2 & \psi(q_1, Juliette, q_2) = \varepsilon \\ \delta(q_2, \varepsilon) = q_5 & \psi(q_2, \varepsilon, q_5) = inc() \end{array} \right. \begin{array}{ll} \mathcal{M}(inc()) = \varepsilon & \theta(q_0, \varepsilon, \varepsilon, q_1) = \varepsilon \\ \mathcal{M}(\varepsilon) = \varepsilon & \theta(q_1, Juliette, \varepsilon, q_2) = \varepsilon \\ \mathcal{M}(inc()) = \varepsilon & \theta(q_2, \varepsilon, \varepsilon, q_5) = \varepsilon \end{array} \left| \begin{array}{l} i = 9 \\ i = 10 \end{array} \right.$$



À la fin du parcours t_4 , nous avons une séquence supplémentaire reconnue (*Juliette*), le registre global i est égal à 10 et il ne reste plus de transitions à examiner. Étant donnée qu'il n'existe plus de symboles pour faire glisser la fenêtre d'analyse, le processus est terminé.

Comme résultat de l'analyse, nous avons les séquences reconnues $\{Roméo\}$, $\{aperçoit\}$ et $\{Juliette\}$ chacune associée à une sortie vide ε , ainsi que le registre global i affecté à 10. Il est simple de constater que cette analyse reconnaît les bonnes séquences, cependant, l'évaluation des fonctions étendues ne respecte pas le principe d'atomicité d'évaluation.

Premièrement, nous avons défini que l'évaluation d'une fonction $\varphi \in \Phi$ doit se réaliser de façon « *tout ou rien* », lors que la fonction n'est pas satisfaite, soit quand $\mathcal{M}(\varphi) = \mathfrak{X}$, un registre comme i ne devrait pas être modifié. Cependant dans t_1 et t_3 , $q_6 \xrightarrow{\varepsilon:fail()} q_5$, la fonction étendue $\varepsilon \xrightarrow[i=i+1]{fail()} \mathfrak{X}$ incrémente i et retourne \mathfrak{X} .

Deuxièmement, nous avons dit que ce principe s'étend à l'évaluation d'un série de fonctions étendues $\varphi_1, \varphi_2, \dots, \varphi_n$ qui sont attachées aux transitions d'un chemin menant d'un état p à un état r . Comme conséquence, dans t_1 et t_3 , $q_4 \xrightarrow{\varepsilon:inc()} q_6$, les changements de la fonction étendue $\varepsilon \xrightarrow[i=i+1]{inc()} \mathfrak{X}$ qu'incrémente i et retourne ε , ne doivent pas non plus être prises en compte.

En résumé, le principe d'atomicité n'est pas respecté, dans t_0 et t_3 , l'évaluation des fonctions étendues n'est pas complète :

1. $\exists i \in n, \mathcal{M}(\varphi_i) = \mathfrak{X}$ (il existe des fonctions étendues qui ne sont pas satisfaites)
2. $p \xrightarrow{\exists i \in n, \mathcal{M}(\varphi_i) \neq \mathfrak{X}} r, p = q_0$ et $r = q_6 \notin F$ (il existe des fonctions satisfaites qui se trouvent dans un chemin non réussi)

4.8 Machines abstraites et évaluation des fonction étendues

Nous avons défini une machine abstraite \mathcal{M} comme un 5-uplet comprenant 1) des registres, 2) de constantes 3) de variables et 4) un état global, et 5) des opérations. Pour être interprétés, les fonctions étendues doivent être écrites dans un langage utilisant les caractéristiques de \mathcal{M} , être stockées et évaluées à la volée. Lorsqu'un chemin de G se trouve sur une transition $t_i = (p[t_i], li[t_i], f[t_i], lo[t_i], n[t_i]) \in E$ associée à une fonction étendue, la fonction $f[t_i]$, étiquetée comme « @f(i) », est analysée (cf. figure 4.9) afin d'extraire son nom « n » et ses entrées « i », si la fonction « n » existe stockée sur un support d'enregistrement, une pile S (stack) de données est utilisée pour empiler l'état de l'automate (p) ainsi que les entrées (i) et le nom (n) de la fonction ($S = p, i, n$).

Ensuite, \mathcal{M} fait remonter (i.e. dépile) le nom de la fonction ($S = p, i$), charge les sources et les compile dynamiquement pour générer les instructions (opérations de \mathcal{M}) qui peuvent être exécutées et charger dans les registres les données qui contiennent les valeurs constantes de la fonction. Les paramètres d'entrée de la fonction 'i' et l'état de l'automate sont ensuite dépilés ($S =$) et la fonction est exécutée. A la fin de l'exécution la valeur de sortie de la fonction est empilée dans S ($S = o$) et l'analyse de la transition se poursuit. Si la valeur de « o » est une sous-étiquette ($l[ft]$) différente du symbole d'échec (\mathfrak{X}) et égale à une valeur booléenne « vrai », la transition est satisfaite et l'état suivant sera alors t_{i+1} , si la valeur de sortie est le symbole d'échec ou égale à une valeur booléenne « faux », la transition n'est pas satisfaite et par conséquent il n'existe pas de chemin accepteur vers t_{i+1} , et l'exploration recommence (*backtracking*). Dans le cas où la transition est satisfaite, l'étiquette de la transition t (stockée sur une pile T) sera égale à la concaténation de $l[ft]$ et $lo[t]$.

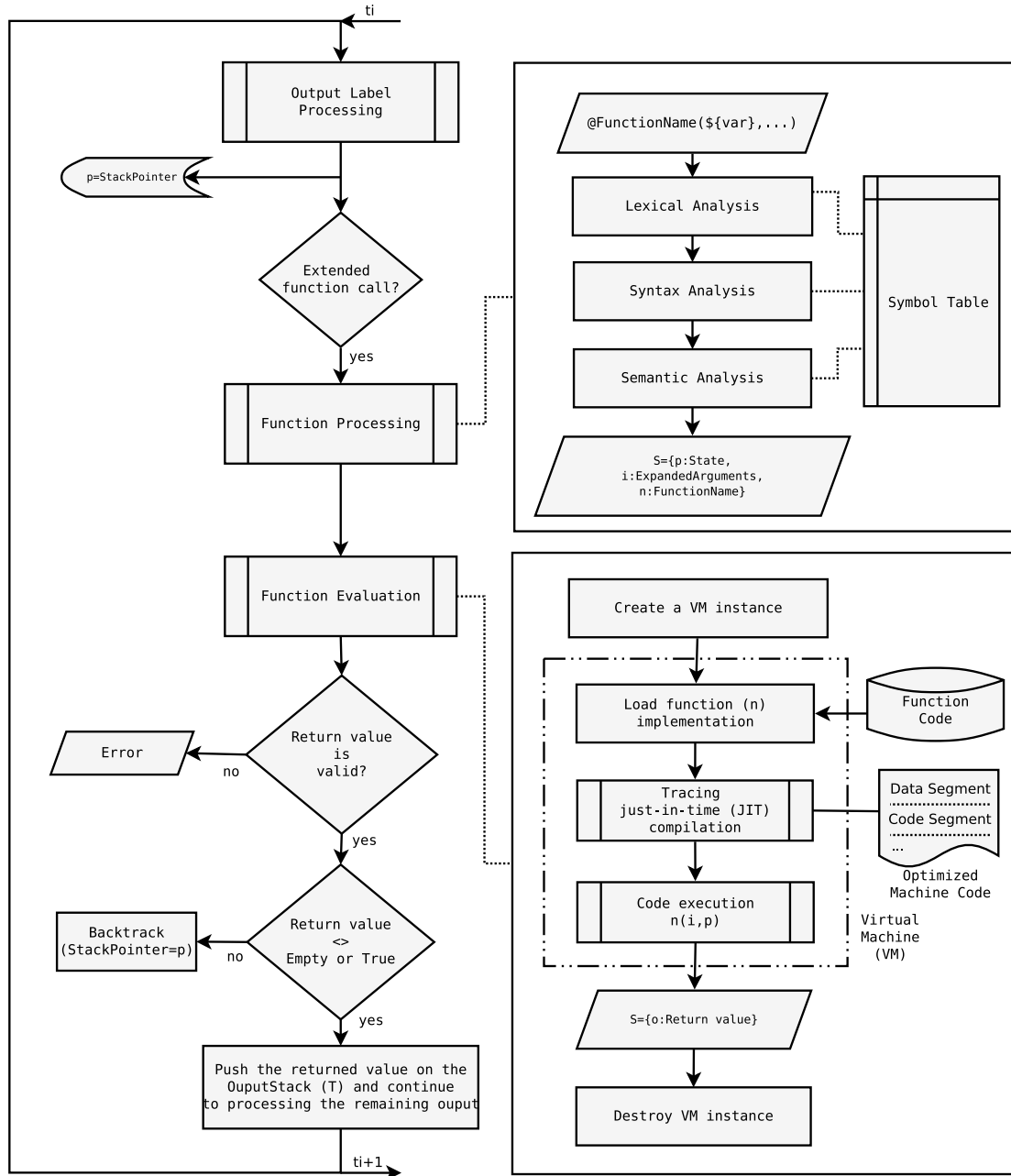


FIGURE 4.9 – Analyse d’une transition étiquetée par une fonction étendue par une machine abstraite

5

Grammaires locales étendues : mise en œuvre

Dans ce chapitre nous présentons la mise en œuvre du formalisme des grammaires locales étendues qui a été introduit au chapitre 4. Leur développement s'inscrit dans le cadre du moteur linguistique UNITEX, qui comme a été évoqué au chapitre 3 est un outil open source pour le traitement de corpus textuels à l'aide de grammaires locales. Dorénavant, nous utilisons le terme [manuel](#) pour faire référence au « Manuel d'utilisation d'Unitex » ([Paumier, 2016](#)).

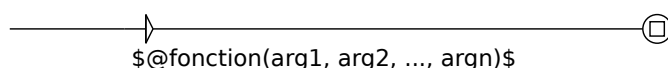
Rappelons qu'une des particularités qui différencie une grammaire locale étendue d'une grammaire locale classique est la capacité d'associer des fonctions arbitraires aux transitions. Considérer des fonctions arbitraires suppose d'une part qu'il n'existe pas de contrainte spécifique au moment où on associe une fonction à une transition, d'autre part, cela implique que les fonctions sont liées à la grammaire mais ne sont pas implémentées dans celle-ci. Par exemple, si *reverse* est le nom d'une fonction placée sur une des transitions d'une grammaire, il n'en existe aucune définition dans la grammaire relative à cette fonction, mise à part la syntaxe d'appel. Une telle caractéristique fournie, du point de vue des traitements, une grande souplesse, mais impose à la fois une réflexion sur la façon de mettre en œuvre les fonctions d'une grammaire locale étendue.

5.1 Anatomie de l'appel à une fonction étendue

Dans notre implémentation une fonction étendue est appelée en suivant différents types de conventions, dont la plus simple est :

$$\text{\$@fonction}(arg_1, arg_2, \dots, arg_n)\text{\$}$$

Sous forme de graphe :



Graphe 5.1 – Convention pour appeler une fonction étendue

Le symbole dollar (\$) comme préfixe et suffixe sert à délimiter la partie de la *sortie étendue* qui doit être évaluée, autrement dit, les séquences en dehors de l'appel à la fonction étendue sont interprétées séparément soit comme des symboles terminaux (des chaînes de caractères littérales), soit comme des appels à d'autres fonctions. S'il s'agit des symboles de l'alphabet de sortie et si la *sortie est satisfaite*, alors ces symboles seront concaténés à la valeur retournée par la fonction. Dans le cas où une sortie étendue comporte plus d'une fonction, l'évaluation est effectuée de la fonction la plus à gauche vers celle la plus à droite. Dans tous les cas, l'évaluation est effectuée seulement si la fonction qui la précède est aussi satisfaite.

Le nom d'une fonction est précédé par le symbole arobase (@). Nous avons choisi cette convention puisque l'arobase est communément associé à la préposition « à » (« at », en anglais). En effet, cela sert à évoquer le fait que la définition de la fonction, c'est-à-dire, le code source contenant les instructions à exécuter se trouve dans un fichier externe (que nous appelons extension) du même nom et se terminant par .upp (dans l'exemple, fonction.upp).

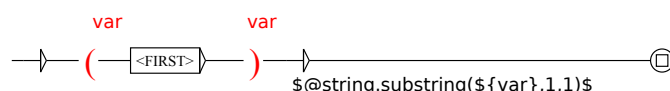
Le nom d'une fonction est valide s'il est composé uniquement de caractères appartenant à la classe [a-zA-Z0-9_], soit des lettres latines minuscules ou majuscule, des chiffres et des tirets bas. Cette restriction suit celle déjà utilisée dans UNITEX pour créer des noms de variables (d'entrée ou de sortie, manuel, p. 105) et permet ainsi de garantir une cohérence vis-à-vis de la création des identifiants associés aux grammaires.

Dans certains cas il est plus propice de définir plus d'une fonction étendue dans une seule extension .upp, par exemple, afin de créer des bibliothèques de fonctions qui partagent un but ou une sémantique commune. Il est alors admis de faire appel à une fonction en utilisant la convention suivante :

$$\text{\$@extension.fonction}(arg_1, arg_2, \dots, arg_n)\text{\$}$$

Dans le graphe 5.2, *string* est le nom du fichier .upp contenant la fonction étendue du nom *substring*. Le nom d'une extension doit respecter les mêmes caractéristiques que celles établies pour les noms de variables et de fonctions étendues, en d'autres termes,

être composé uniquement de lettres latines minuscules ou majuscules, de chiffres et de tirets bas. En outre, aussi bien les noms de variables, de fonctions étendues et d'extensions doivent avoir une longueur inférieure ou égale à 127 caractères et sont sensibles à la casse, c'est-à-dire, les noms avec des lettres minuscules sont considérés comme étant différents de ceux contenant des lettres majuscules.



Graph 5.2 – Convention pour appeler une fonction étendue avec un nom différent du fichier où elle est définie

Immédiatement après le nom de la fonction étendue, il est requis d'écrire une parenthèse ouvrante qui délimite avec une autre parenthèse fermante les arguments qui sont passés à la fonction étendue. Le nombre des arguments situées entre parenthèses doit normalement correspondre au nombre des paramètres attendus par la définition de la fonction. Cependant, il n'existe pas de restriction de cardinalité, si un appel est réalisé avec un nombre d'arguments différents que celui de paramètres, alors les arguments en excès sont ignorés et les paramètres manquants sont assignés à la valeur nulle.

Comme évoqué précédemment, une fonction étendue peut recevoir un ensemble d'arguments, séparés par des virgules, en tant que paramètres d'entrée. Tandis que les paramètres sont les variables utilisées pour définir l'implémentation de la fonction étendue (dans un fichier `.upp` externe à la grammaire), les arguments sont les valeurs qui sont passées à la fonction lorsqu'elle est appelée dans un graphe.

Une liste d'arguments peut être vide. En d'autres termes, la fonction étendue peut avoir zéro paramètres d'entrée ou se voir affectée la valeur nulle à chacun de ses paramètres manquants. Par contre, si la fonction étendue reçoit un ou plusieurs paramètres, chaque argument transmis peut correspondre à une valeur nulle, à une valeur booléenne (vraie ou fausse), à une chaîne de caractères, au contenu d'une variable d'entrée ([manuel](#), p. 105) ou de sortie ([manuel](#), p. 147), ou à une référence des variables précitées (entrée ou sortie).

5.1.1 Nuls et booléens

Pour passer un argument de valeur nulle il est nécessaire d'utiliser le mot-clé `nil`. Un argument de valeur nulle peut être compris comme un argument dépourvu de valeur. Pour la valeur booléenne vraie, le mot-clé est `true` et pour la valeur fausse le mot-clé est `false`. Nous avons adopté cette convention en accord avec les mot-clés `nil | false | true` qui peuvent être utilisés avec une connotation similaire dans la définition des fonctions. Par ailleurs, il est utile de remarquer que ces trois mot-clés doivent toujours être écrits en respectant la casse.

5.1.2 Chaînes de caractères littérales

Pour construire un argument contenant une chaîne littérale de caractères, telle que « chaîne » ou « autre chaîne », il est juste requis d'écrire la suite de caractères, sans aucun autre symbole particulier l'entourant. Autrement dit, une chaîne littérale est uniquement délimitée par les virgules ou par les parenthèses qui séparent les arguments. Afin d'utiliser les caractères qui ont un sens spécial dans la syntaxe d'appel dans une fonction étendue, tels que le symbole dollar (\$) ou « et » commercial (&), l'implémentation courante permet de les déspecialiser (mode d'« échappement »). Ainsi, pour utiliser ces symboles dans une chaîne littérale, il suffit de les redoubler : \$\$ → \$, && → &. En outre, dans une liste d'arguments non-vide, une chaîne littérale est vide si elle ne contient pas de caractères et dans ce cas elle prend la valeur nulle. Finalement, les chaînes littérales peuvent se concaténer aux variables que nous décrivons ensuite. Il est aussi utile de remarquer que la longueur maximale d'un argument du type chaîne littérale est de 4095 caractères.

5.1.3 Variables d'entrée et de sortie

Il est possible de passer un argument à une fonction étendue contenant la valeur d'une variable d'entrée ou de sortie. Contrairement à la convention utilisée dans UNITEX pour faire référence aux variables en encadrant leur nom avec le caractère \$, par exemple \$variable\$, dans les arguments d'une fonction étendue les valeurs des variables sont indiquées en utilisant la convention :

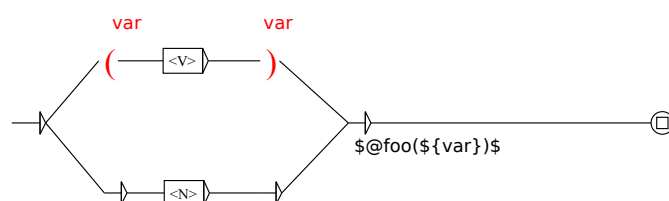
$$\${variable}$$

Le nom de la variable est entouré par des accolades ({ et }) et préfixé par le symbole dollar \$. Lorsqu'une variable est retrouvée, sa valeur (une chaîne de caractères) est assignée à l'argument passé à la fonction. Si la variable fait partie d'un argument du type chaîne littérale, alors sa valeur est concaténée à la chaîne dans laquelle se trouve.

En outre, les variables non assignées, qui ne font pas partie d'un argument du type chaîne, sont assignées par défaut à la valeur nulle. Dans le traitement des arguments des fonctions étendues, ce comportement est plus pertinent que celui d'utiliser une politique unique ([manuel](#), p. 154) lorsqu'une variable n'est pas définie. En effet, cela permet d'autoriser des chemins qui définissent une variable ainsi que d'autres chemins qui ne la définissent pas sans considérer ces derniers comme des erreurs (voir le graphe [5.3](#)). Il est alors de la responsabilité de la fonction étendue (l'implémentation) de prendre en charge les deux cas. Nous avons trouvé cette approche utile dans la réalisation de certaines tâches qui sont prises en charge par des fonctions étendues :

- D'une part, pour générer des sorties de graphes dictionnaires ([manuel](#), p. 69) avec une fonction étendue qui met en forme des ensembles de variables éventuellement non définies.

- D'autre part, pour passer des variables, éventuellement non définies, à des fonctions étendues qui ont la charge de faire des requêtes à une ressource externe telle qu'une base de données.



Graphe 5.3 – Variable défini par un seul chemin

5.1.4 Chaînes de caractères non-littérales

Il est possible de construire des arguments qui combinent des chaînes littérales et de variables dans un seul argument, par exemple : littéral $\${variable}$. L'argument qui est passé à la fonction étendue est alors composé de parties littérales et de variables concaténées.

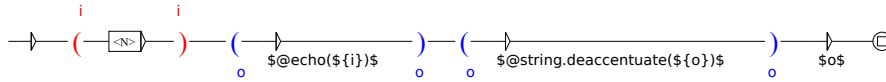
Les règles utilisées pour obtenir les valeurs des variables sont quasiment les mêmes que celles citées dans la partie *variables d'entrée et sortie* (cf. sous-section 5.1.3). Elles ne diffèrent que dans le cas du traitement des variables non assignées. Lorsqu'une variable isolée n'est pas définie, elle se voit affecter la valeur nulle. Par contre, lorsqu'elle fait partie d'un argument combinant chaînes littérales et variables, elle est remplacée par une chaîne vide (contenant zéro caractères) avant d'être concaténée. Par exemple, si $\${variable}$ est non défini, alors l'argument littéral $\${variable}$ sera égal à littéral.

5.1.5 Références aux variables d'entrée et de sortie

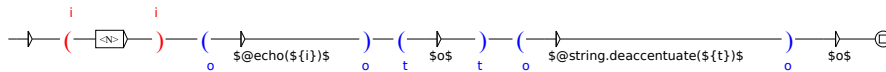
Nous avons vu qu'en utilisant la convention $\${variable}$, il est possible de passer un argument à une fonction étendue contenant la valeur d'une variable d'entrée ou de sortie. Une des tâches pour appeler une fonction étendue est donc de résoudre d'abord, pour chaque argument, les variables utilisées avant de construire l'argument final qui sera passé pour évaluer la fonction. Par résoudre nous entendons la transformation d'un nom de variable en une chaîne de caractères. Considérons à présent deux types de scénario où la notion de référence à une variable est utile :

- Il est nécessaire de réaliser une opération sur une variable de sortie et de stocker le résultat dans la même variable, par exemple, mettre la variable de sortie en majuscules, la convertir en une sous-chaîne, lui supprimer les accents, la translittérer, etc. En utilisant le mode de passage par valeur, la démarche pour y parvenir consisterait à appeler la fonction étendue qui réalise le processus : $\${fonction(\${variable})}$ et à stocker le résultat de l'appel dans la même variable. Cependant, cette stratégie ne s'avère pas satisfaisante. Il n'est pas possible de résoudre l'ancienne valeur d'une

variable lorsqu'elle est redéfinie (voir le graphe 5.4). Comme alternative, il est possible de créer une variable temporaire afin d'éviter de perdre la valeur affectée avant la réassignation (voir le graphe 5.5). Une approche plus simple, et moins coûteuse, consisterait à établir une notion de passage par référence des variables.

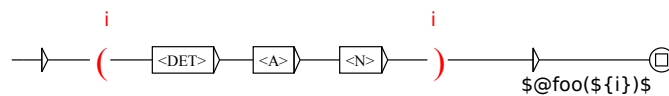


Graphe 5.4 – Variable redéfinie et passage par valeur



Graphe 5.5 – Variable temporaire et passage par valeur

- Il est requis de connaître ou de traiter directement les *tokens* qui composent une variable d'entrée. Rappelons qu'en utilisant les variables d'entrée il est possible de stocker des parties du texte reconnu (manuel, p. 105). Lorsqu'un argument contient une variable d'entrée, c'est la valeur de la variable, c'est-à-dire, la chaîne de caractères composée de la concaténation de chacun des *tokens*, séparés par des espaces, qui est utilisée. Étant donnée l'entrée « a deep scar on his brow », le graphe graphe ?? affecte la variable *i* à la séquence « a deep scar ». Il faut noter alors que pour la construction des arguments d'une fonction étendue, une fois la variable convertie en un chaîne de caractères, elle perd le lien avec les *tokens* qui la composent. Si dans une grammaire locale ce comportement peut être suffisant, dans une grammaire locale étendue il pourrait être souhaitable de réaliser une opération dépendant ou utilisant une partie des *tokens* d'une variable d'entrée. Comme pour le scénario précédent, une approche pour y parvenir consisterait à établir une notion de passage par référence des variables.



Graphe 5.6 – Variable composée par plusieurs *tokens*

Le passage par référence des variables implique que l'argument d'une fonction ne prenne pas la valeur de la variable mais une référence. La convention que nous avons adoptée pour passer une variable par référence est la suivante :

&{variable}

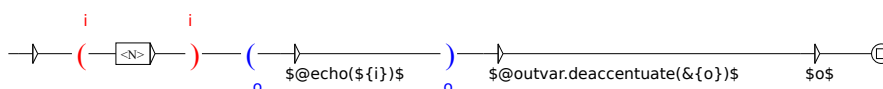
Ainsi, le fait de remplacer le symbole dollar par « et » commercial (&), indique que nous souhaitons obtenir non pas la valeur mais la référence de la variable. Les références aux variables d'entrée ou de sortie sont de nature différente.

Références à des variables d'entrée : Une référence à une variable d'entrée est passée comme une liste de constantes numériques $r_i = \{n_1, n_2, n_3, n_4\}$ indiquant le rang absolu $[n_1, n_2[$ des *tokens* utilisés pour construire la valeur référencée. Chacune de valeurs se rapporte à la version du texte tokenisé ([manuel](#), p. 38) et correspond à :

- n_1 = Position du premier *token*
- n_2 = Position après le dernier *token*
- n_3 = Position de début (en caractères) dans le premier *token*
- n_4 = Position de fin (en caractères) dans le dernier *token*

Comme exemple, dans le graphe (inputvarref1), tandis que $\${i}$ est transformé en « a deep scar », $\&\{i\}$ est converti en la liste $\{0, 3, 0, -1\}$. Notons en effet que $a=0$; $depp=1$; $scar=2$. Autrement dit, n_1 prend la position du premier *token* (0), n_2 la position après le dernier *token* ($2 + 1$), n_3 la position de début (en caractères) dans le premier *token* (0), n_4 la position de fin (en caractères) dans le dernier *token*, où -1 signifie prendre en compte tout le *token*. Nous avons adopté cette convention des 4 entiers car il correspond à la modélisation interne de variables dans UNITEX.

Références à des variables de sortie : Une référence à une variable d'entrée est passée comme un argument du type référence `elg.ustring`, `ustring` (*Unitex String*) est une bibliothèque de traitement de chaînes de caractères spécialement conçue pour être utilisée avec les grammaires locales étendues. Ainsi, lorsqu'un paramètre d'une fonction prend une référence du type `elg.ustring`, les changements réalisés sur le paramètre sont affectés directement à la variable de sortie. Le graphe 5.7 constitue ainsi une alternative au graphe 5.5.



Graph 5.7 – Variable passée par référence

Enfin, il est pertinent de préciser que contrairement aux variables passées par valeur, une variable passée par référence ne peut pas être combinée à une chaîne de caractères. Ceci est dû naturellement à l'incompatibilité entre les deux types.

5.1.6 Constantes numériques

Dans l'implémentation actuelle, les arguments du type constantes numériques, telles que 9, 98.4 ou 0.2e17 sont passées en tant que chaînes de caractères. Cette méthode de passage ne répond à aucune contrainte particulière et pourrait évoluer dans des versions à venir. Entre temps, il est possible de convertir un paramètre p en un nombre en utilisant l'expression $n = \text{tonumber}(p)$ dans la définition de la fonction étendue.

5.2 Un langage interprété comme infrastructure des fonctions étendues

Lorsque nous avons envisagé l'option d'utiliser un langage interprété comme infrastructure des fonctions étendues, nous avons passé en revue plusieurs langages de programmation, entre autres : C++, Java, Python, Haxe, Javascript et Lua.

Nous avons finalement porté notre choix par Lua ([Ierusalimschy et al., 1996, 2007](#)), un langage de programmation de scripts extensible et portable. Cette décision a été motivée par plusieurs aspects :

- Lua fournit une syntaxe concise et simple pour décrire des données ce qui le rend utile comme un langage de configuration et d'échange d'information. Cette simplicité est aussi présente lors de la manipulation des tables dynamiques, la seule structure de données disponible en Lua et qui sert à représenter tout autre type abstrait tel que les chaînes de caractères.
- Le langage offre un bon niveau d'abstraction en rapport à l'architecture matérielle où le langage est interprété.
- Il permet le prototypage et les tests rapides.
- L'interprète du langage dispose d'un sous-système de gestion automatique de la mémoire. Ce sous-système, non-intrusif, permet d'écrire des programmes plus simples du fait qu'il libère le développeur de la tâche de gestion manuelle de la mémoire.

Dont une particulièrement attractive afin de mettre en œuvre le formalisme des grammaires locales étendues sur Unitex. Il s'agit de la capacité de Lua d'être appelé et de communiquer avec d'autres logiciels écrits en C et C++, ce qui font de lui un langage propice pour le développement de briques logiciels modulaires, chacune regroupant des fonctionnalités bien définies.

En outre, Lua peut être aussi utilisé comme langage de configuration, par exemple dans [Roman et al. \(2002\)](#), il est employé pour paramétrer et coordonner des dispositifs informatiques dans un environnement informatique ubiquitaire. Il peut aussi prendre le rôle de langage d'extension, tel que dans [Ernst et al. \(2009\)](#), où il sert à étendre les fonctionnalités d'un [framework](#) destiné à la détection des gestes et l'analyse des visages. Finalement, Une autre caractéristique importante est que l'interprète de Lua est rapide et a une faible empreinte mémoire¹.

Dans le cadre des grammaires locales étendues (ELGs), Lua est utilisé à deux fins : comme extension du moteur d'analyse des grammaires locales et comme langage pour implémenter des fonctions étendues.

1. Pour d'autres détails techniques, consultez le livre de référence de Lua ([Ierusalimschy, 2016](#))

5.2.1 Procédure d'exécution d'une fonction étendue

- Création d'un λ -state vide
- Définition des fonctions autorisées dans le λ -state
- Chargement du fichier `.upp` contenant la fonction étendue
- Compilation de la fonction étendue
- Empilement (`push`) des paramètres de la fonction étendue dans la λ -pile
- Exécution du code compilé de la fonction étendue
- Désempilement (`pop`) des valeurs de retour qui se trouvent dans la λ -state

5.3 Opérations et événements

- Opérateurs : il s'agit des opérateurs disponibles sur les chaînes, sur les *tokens*, sur les états et sur les ressources.
- Événements : il s'agit des événements `init_event`, `load_event`, `unload_event`, `token_event`, `slide_event`, `onenter_event`, `onexit_event`, `onbacktrack_event`, `onmatch_event`, `onfail_event`.

5.4 Chaîne de traitement

5.5 Application d'une grammaire locale étendue

Les étapes d'application d'une grammaire locale étendue sont issues de la stratégie de représentation des *tokens* proposée par [Paumier \(2003b\)](#) et mise en œuvre dans UNITEX. La démarche se distingue notamment par la façon de traiter et construire les sorties, par conséquent dans la façon d'analyser et évaluer les fonctions étendues, lors de l'application de la grammaire.

Comme illustré sur la figure 5.1, le processus global est divisé en trois étapes : [Découpage en *tokens*](#), [Optimisation de la grammaire](#) et Application finale de la grammaire.

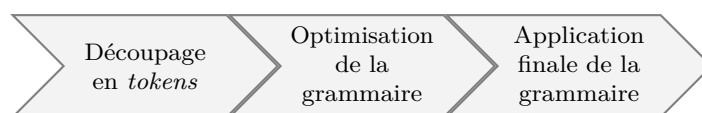


FIGURE 5.1 – Étapes d'application d'une grammaire locale étendue à un texte

5.5.1 Découpage en *tokens*

Pour les langues comportant une séparation précise de mots, un *token*, est soit une séquence de lettres, soit un caractère non-alphabétique. Étant donné un texte d'exemple :

« *The incident took place on Tuesday¹ the 2nd of May of 1969* », telle qu'illustrée à la figure 5.2, nous découpons le texte en *tokens* en leur associant un entier unique². Ce processus est réalisé en un seul passage et en un temps proportionnel à la taille du texte, soit $\mathcal{O}(n)$. Il est également indépendant des types ou du nombre de grammaires qui seront appliquées par la suite. Une fois le texte balayé, une liste indexée de *tokens* comme celle de la figure 5.3 est finalement constituée. Le texte peut alors être représenté comme une succession d'entiers chacun correspondant à l'indice du *token* qui lui est associé. Nous présentons un exemple dans la figure 5.4.

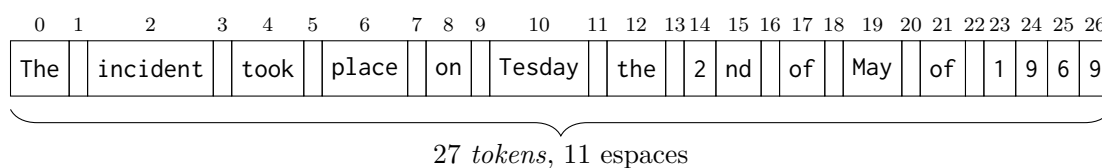


FIGURE 5.2 – Découpage : *tokens*

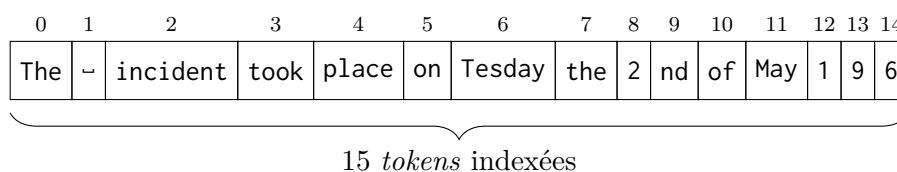


FIGURE 5.3 – Découpage : *tokens* indexées

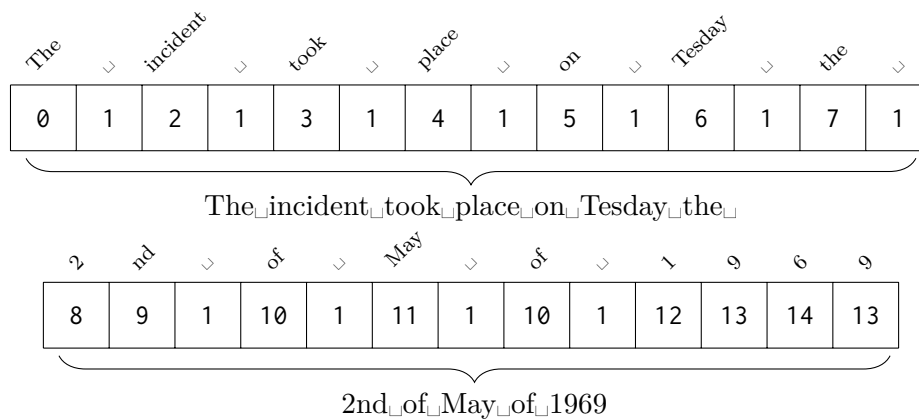


FIGURE 5.4 – Découpage : représentation sous forme d'indices

1. *Tuesday* : La faute d'orthographe est fournie à titre d'illustration
 2. Ceci est l'approche utilisée par UNITEX où les *tokens* sont représentées par des entiers, ainsi que les grammaires lors de leur phase d'application

5.5.2 Optimisation de la grammaire

Étant donné une grammaire dont les étiquettes de transition sont des *tokens* conformes à la règle de découpage utilisée au cours de la première étape, les trois techniques d'optimisation proposées par Paumier (2003b) sont mis en place :

Remplacer les *tokens* par des entiers

Chaque transition marquée par un *token* est remplacé par la liste d'entiers à laquelle il peut correspondre. Dans le cas exemplifié, le token *the* sera remplacé par 0_{the} et par 12_{The} . La complexité de cette étape est de $\mathcal{O}(N)$, où N est la taille de l'alphabet d'entrée.

Créer des références à des classes de mots :

Les classes de mots, qui établissent un lien avec les ressources linguistiques, sont référencées dans les transitions de la grammaire au moyen des masques lexicaux (cf. tableau II). Lors de l'application de la grammaire au texte, vérifier si une unité lexicale correspond à un masque lexical exige la consultation répétitive des ressources linguistiques ce qui rend la grammaire non-déterministe du fait que plusieurs masques pourraient reconnaître la même unité lexicale.

Unités lexicales simples : Pour pallier ce problème, en ce qui concerne les unités lexicales simples, la stratégie adoptée est basée tout d'abord sur le comptage du nombre d'unités lexicales qui correspondent à un masque lexical. Ensuite, le masque est remplacé par la liste complète des entiers représentant les unités lexicales auxquels il peut correspondre. Dans le cas contraire, le masque est indexé par un entier et pour enregistrer le fait que l'unité lexicale correspond au masque lexical une structure auxiliaire de données est utilisée.

Cette structure est un tableau de bits associé à chaque *token*. Le masque d'indice n vérifie une unité lexicale lorsque le $n^{\text{ième}}$ bit de leur tableau est mis à 1. Ainsi, prouver si une unité lexicale quelconque correspond à un masque lexical, se réalise en deux temps : le premier par le numéro de l'unité lexicale, le deuxième par le numéro du masque lexical.

Unités lexicales composées : S'agissant des unités lexicales composées, la stratégie suivie marche comme une extension de la précédente. Dans un premier temps, tous les motifs qui peuvent correspondre à une unité lexicale composée sont vérifiés, ensuite un arbre est construit à partir de la liste de leur composants qui vérifient au moins un masque lexical. Chaque nœud de l'arbre correspond à l'entier qui est associé à l'unité lexicale. Chaque nœud terminal mémorise également les numéros des masques qui correspondent à l'unité lexicale composée.

Savoir si un masque lexical quelconque correspond à une ou plusieurs unités lexicales composées peut se réaliser en explorant simultanément l'arbre et la représentation

indexée du texte. L'hypothèse est alors vérifiée, si lors de cette exploration l'algorithme atteint un nœud final et si parmi les numéros des masques stockés se trouve celui recherché.

Trier les listes des entiers :

Finalement, à l'issue des optimisations précédentes, les transitions sont de deux types, soit des listes d'entiers représentant les tokens (transitions-*token*), soit des masques représentant les classes de mots (transitions de classe).

Étant donné un état de la grammaire, vérifier si la transition peut correspondre à un *token* du texte équivaut à comparer leur numéro avec ceux de la transition. Pour améliorer le temps de ce traitement, les transitions de lexique sont fusionnées en une liste unique ordonnée de n entiers, qui par recherche dichotomique, pourra être consultée en $\mathcal{O}(\log(n))$.

5.5.3 Construction des sorties

Lorsque une fonction étendue fait partie d'une sortie, des politiques supplémentaires à celles d'ignorer les sorties, les remplacer ou les insérer à gauche des séquences [manuel](#), p. 146 peuvent être mises en place, il s'agit d'un mode *mettre à jour* avec priorité, leur construction est décrit au tableau 5.1.

Cas	Σ_{input}	Γ_{output}	ELGs					
			LGs			$\mathcal{U}_{pdate_\sigma}$	$\mathcal{U}_{pdate_\gamma}$	$\mathcal{U}_{pdate_\varepsilon}$
			\mathcal{I}_{gnore}	\mathcal{R}_{eplace}	\mathcal{M}_{erge}			
$\dots \rightarrow \bullet \rightarrow \dots$	ε	ε	ε	ε	ε	ε	ε	ε
$\dots \rightarrow \underset{\gamma}{\bullet} \rightarrow \dots$	ε	γ	ε	γ	γ	γ	γ	γ
$\dots \rightarrow \circ \rightarrow \dots$	σ	ε	σ	ε	σ	σ	σ	σ
$\dots \rightarrow \underset{\gamma}{\circ} \rightarrow \dots$	σ	γ	σ	γ	$\gamma \cdot \sigma$	γ	σ	ε

Cas	Σ_{input}	Γ_{output}	ELGs					
			LGs					
			\mathcal{I}_{ignore}	$\mathcal{R}_{replace}$	\mathcal{M}_{merge}	$\mathcal{U}_{pdate_\sigma}$	$\mathcal{U}_{pdate_\gamma}$	$\mathcal{U}_{pdate_\varepsilon}$

TABLEAU 5.1 – Politiques pour la construction des sorties des ELGs

Afin de comprendre leur utilité prenons comme exemple la phrase (13) :

(13) *The crazy things*

Ainsi que la grammaire locale étendue du graphe 5.8 ci-après :



Graph 5.8 – Exemple de la construction des sorties avec une ELG

L'appel à une fonction étendue `superlatif` renvoie le superlatif de l'étiquette d'entrée courante qui est directement accessible à partir de la fonction étendue. Les possibles sorties sont énumérées dans le tableau 5.2.

NOM	DESCRIPTION	SORTIE
\mathcal{I}_{ignore}	Le résultat de la concaténation des étiquettes d'entrée appartenant à chacun des chemins réussis stockés (c'est-à-dire ignorant les étiquettes de sortie)	<i>The crazy things</i>
\mathcal{M}_{merge}	Le résultat de la concaténation consistant à fusionner les étiquettes de sortie (à gauche) avec les étiquettes de sortie appartenant à chacun des chemins réussis stockés (c'est-à-dire les sorties sont insérées à gauche des séquences reconnues)	<i>The craziest crazy things</i>
$\mathcal{R}_{replace}$	Le résultat de la concaténation consistant à remplacer les étiquettes d'entrée par les étiquettes de sortie appartenant à chacun des chemins réussis stockés (c'est-à-dire en remplaçant les étiquettes d'entrée par des étiquettes de sortie)	<i>craziest</i>
$\mathcal{U}_{pdate_\sigma}$	Le résultat de la concaténation des étiquettes d'entrée avec les étiquettes de sortie étendues appartenant à chacun des chemins réussis stockés (c'est-à-dire le remplacement des étiquettes d'entrée par des étiquettes de sortie étendues)	<i>The craziest things</i>

TABLEAU 5.2 – Construction de sorties pour le graphe 5.8

6

Extraction d'information : la reconnaissance d'entités nommées dans des textes formels et bruités

La reconnaissance d'entités nommées dans des données textuelles compte déjà un grand nombre d'applications qui s'appuient sur des approches orientées connaissances linguistiques, données brutes ou hybrides. Ce chapitre débute avec la présentation de travaux autour de la reconnaissance des entités nommées et se poursuit par l'identification des entités nommées contenant du bruit. Par bruit nous sous-entendons les fautes typographiques, telles que celles rencontrées souvent dans des textes provenant de transcriptions orales, produites par OCR, SMS ou tweets, etc. En effet, le traitement automatique de documents dégradés s'avère sensiblement plus compliqué et limité, nécessitant davantage de traitements. Nous présentons les entités nommées de façon générale tout en essayant de nous focaliser sur les recherches et applications existantes autour de la reconnaissance d'entités nommées dans des textes dégradés.

6.1 Les entités nommées

Les entités nommées sont des fragments du langage qui se réfèrent à une entité unique du domaine du discours tout en apportant une valeur informationnelle. Plus

particulièrement, ils peuvent se définir comme des unités lexicales qui décrivent entre autres des noms et prénoms de personnes, des noms d'organisations, des lieux, des dates, des quantités, des distances, des valeurs, etc. (Chinchor et al., 1998, Tjong Kim Sang et De Meulder, 2003, Meur et al., 2004).

L'identification des entités nommées s'avère incontournable dans de nombreuses tâches liées à l'extraction automatique de l'information. En effet, les entités identifiées sont utilisées dans plusieurs types de systèmes : fouille de données textuelles, moteurs de recherche, traduction automatique, question-réponse, interaction homme-machine, etc. En outre, les entités nommées peuvent concerner plusieurs domaines scientifiques et être potentiellement appliquées à différents styles de texte.

En <date>1528</date>, <pers>François Ier</pers> fixe officiellement sa résidence à <lieu>Paris</lieu>.
--

TABLEAU 6.1 – Exemple d'annotation embarquée d'entités nommées

date	4	8	1528
pers	10	22	François Ier
lieu	58	63	Paris

TABLEAU 6.2 – Exemple d'annotation déportée d'entités nommées (*stand-off annotation*)

L'exemple du tableau 6.1 et 6.2, illustre les entités nommées identifiées dans la phrase « En 1528, François Ier fixe officiellement sa résidence à Paris » : 1528 DATE, François Ier PERS et Paris LIEU. Les entités identifiées sont annotées à l'aide de balises textuelles portant l'étiquette correspondant au type d'entité qu'elles délimitent, dans l'exemple : date, personne et lieu. Dans une annotation textuelle, les balises se retrouvent fusionnées dans le texte (tableau 6.1). En revanche, dans une annotation déportée, les étiquettes sont munies de la position de l'entité et sont répertoriées séparément du texte (tableau 6.2).

6.1.1 Aperçu général

La définition d'entités nommées fut le fruit des campagnes réalisées à la fin des années 90 consacrées au développement des méthodologies autour de l'extraction d'information. Plus précisément, à l'occasion de la septième conférence MUC (Message Understanding Conferences), Chinchor (1998) apporte une première définition des entités nommées d'un point de vue utile à l'extraction automatique d'information. Ils les décrivent alors comme des noms propres particuliers désignant *les noms de personne, les organisations, les lieux, aussi bien que les dates, les heures, les pourcentages et les montants monétaires*. Daille et al. (2000) ont étendu cette définition en indiquant que la notion d'entité nommée est plus vaste et que l'on peut y inclure aussi bien les gentilés, les personnages

de légendes, les maladies ou les drogues qui ne sont pas toujours admis comme des noms propres.

Quelques années plus tard, [Friburger \(2002\)](#) dans ses travaux de thèse rapproche également le terme d'entité nommée de celui de nom propre et apporte des éléments qualitatifs supplémentaires. Notamment, en remarquant le fait que les chercheurs travaillant autour de l'extraction d'information emploient le terme d'entité nommée pour cibler non seulement tous les noms propres au sens classique et élargi, mais aussi les expressions temporelles et les expressions de quantités. Par ailleurs, ces travaux soulignent la difficulté de *délimiter les noms propres des autres noms ; puisque il y a une continuité entre l'ensemble des noms propres et l'ensemble des noms communs*. Dans la même optique, [Vicente \(2005\)](#) oriente le terme vers le traitement automatique du langage naturel en attribuant des éléments discursifs monoréférenciels proches des noms propres en accord avec des patrons syntaxiques déterminés.

Même si la définition d'entité nommée semble échapper à toute définition standard, c'est à l'occasion de la campagne ESTER (Evaluation des Systèmes de Transcription Enrichie d'Émissions Radiophoniques) que [Meur et al. \(2004\)](#) donnent une définition plus pragmatique des EN comme des types d'*unités lexicales particulières qui font référence à une entité du monde concret dans certains domaines spécifiques notamment humains, sociaux, politiques, économiques ou géographiques et qui ont un nom (typiquement un nom propre ou un acronyme)*.

6.1.2 Typologie des entités nommées

L'annotation d'entités nommées implique la définition d'une typologie dédiée pour leur classification. Ainsi, il existe un grand nombre de typologies, dont une des plus répandues et générales est celle de la conférence MUC 7 ([Chinchor, 1998](#)), divisée en trois catégories (tableau 6.3). En accord avec la définition d'entité nommée, la catégorie ENAMEX regroupe les anthroponymes, les toponymes et les entreprises. La catégorie TIMEX comprend les entités temporelles et la catégorie NUMEX les valeurs physiques et monétaires.

TYPE	DESCRIPTION
ENAMEX	Noms de personne, lieu, organisations, entreprises
TIMEX	Dates et heures
NUMEX	Expressions monétaires et pourcentages

TABLEAU 6.3 – Catégories d'entités nommées dans MUC(1995)

Néanmoins, cette typologie exclut une grande partie d'entités nommées qui pourraient s'avérer utile dans le cadre de l'extraction d'information. Même si dans la pratique, un besoin particulier imposerait d'adapter ou de concevoir une nouvelle typologie, [Paik et al. \(1996\)](#) ont introduit une classification élargie comportant trente

catégories divisées en 9 classes ¹ :

CLASSE	TYPES
Géographique	villes, ports, aéroports, pays, etc
Affiliation	religions, nationalités
Organisation	entreprises, organisations, etc
Humain	personnes, fonctions
Document	documents
Équipement	machines, matériels, etc
Scientifique	maladies, médicaments
Temporel	dates et heures
Divers	autres entités

TABLEAU 6.4 – Typologie d’entités nommées de Paik et al.

Parmi d’autres catégorisations plus exhaustives, citons également la hiérarchie étendue de Sekine et al. (2002) conçue pour couvrir le plus d’entités possibles, ceci en introduisant 150 types d’EN, en essayant de ne pas couvrir uniquement un domaine spécifique. Cet effort de complétude, ainsi que d’autres propositions ultérieures (CITATION) doivent être considérées en tenant compte de la remarque d’Ehrmann (2008), qui indique qu’il n’existe aucune catégorisation idéale.

Afin de mettre en rapport les typologies des entités nommées avec les principales campagnes d’évaluation qui ont été réalisées durant les 20 dernières années (1996-2016), nous présentons dans le tableau 6.5 un récapitulatif chronologique adapté de Nouvel (2012). Mises à part les typologies, le tableau indique les langues visées et la modalité des écrits.

6.1.3 Difficultés autour de la catégorisation d’entités nommées

L’ambiguïté intrinsèque aux langues naturelles est beaucoup plus vaste que celles concernant les langages artificiels. De ce fait, elle complexifie la tâche d’identification des entités, empêchant leur catégorisation et dégradant les performances des outils qui visent à les extraire. En effet, lors de la phase d’annotation d’entités nommées, certains phénomènes linguistiques récurrents génèrent de l’ambiguïté dans la reconnaissance. Ces phénomènes sont aussi connus comme facettes sémantiques selon Cruse (1986, 1995, 2004) (Martineau et al., 2007) et comprennent, par exemple, la polysémie, la métonymie ou les référents multiples.

1. Comme remarquée par Daille et al. (2000), cette classification en 9 classes, est similaire à la typologie des noms propres de Bauer (1985). En effet, cette dernière comporte des classes semblables : anthroponymes, toponymes, ergonymes, praxonymes, phénonymes et zonymes.

ANNÉES	CAMPAGNE	LANGUES	MODALITÉ	TYPLOGIE
1996 1997	MUC-6 MUC-7	en	Rapports écrits et journalistiques	(pers), (org), (loc), (date), (heure), (montant), (pourcentage)
1997 1998	MET-1 MET-2	es, jp, zh	Écrit journalistique	(pers), (org), (loc), (date), (heure), (montant), (pourcentage)
1999	IREX	jp	Écrit journalistique	(pers), (org), (loc), (artefact), (date), (heure), (montant), (pourcentage)
2002 2003	CoNLL-2002 CoNLL-2003	es, en, de, nl	Écrit journalistique	(pers), (org), (loc), (misc)
2006	HAREM	pt	Écrit journalistique	(pers), (org), (loc), (temps), (œuvre), (événement), (abstraction), (chose), (valeur), (autre)
2006	SIGHAN	zh	Écrit	(pers), (org), (loc), (entité géo-politique)
2007 2008	ACE07 ACE08	en, ar, zh	Écrit journalistique et conversationnel	(pers), (org), (loc), (bâtiments), (entité géo-politique), (armes), (véhicules)
2007-2011	EVALITA	it	Écrit journalistique	(pers), (org), (loc), (entité géo-politique)
2009	ESTER2	fr	Oral journalistique	(pers), (org), (loc), (temps), (montant), (fonction), (produit)
2011-2012	ETAPE	fr	Oral journalistique et conversationnel	(pers), (org), (loc), (temps), (montant), (fonction), (produit)
2015	TAC-EDL	en, zh, es	Écrit	(pers), (org), (loc), (entité géo-politique), (établissement)

TABLEAU 6.5 – Campagnes menées entre 1996 et 2015 en relation avec la reconnaissance d'entités nommées

Polysémie

Par définition, une unité lexicale est polysémique lorsqu'elle correspond à plusieurs significations. C'est ainsi que lorsqu'une entité nommée appartient à plus d'une classe sémantique, elle devient ambiguë. Prenons l'exemple du nom « calcul »¹ :

Dans le domaine des mathématiques et de leurs applications, il peut signifier :

1. *l'opération ou l'ensemble d'opérations portant sur des nombres ou des symboles,*
2. *la méthode particulière à certaines branches des mathématiques,*

Par contre, en dehors des mathématiques et de leurs applications, il peut se référer à :

3. *le processus d'évaluation des probabilités,*
4. *la prévision et combinaison de moyens d'action en vue de son intérêt personnel.*

Un autre exemple assez évocateur pour décrire la polysémie est celui du mot Orange :

1. Définitions tirées du site de Centre National de Ressources Textuelles et Lexicales (CNRTL) : <http://www.cnrtl.fr>

- (14) Comment `<org>Orange</org>` recycle les téléphones portables ?
- (15) Située au Sud de la France, au cœur de la Provence, `<lieu>Orange</lieu>` est une ville romaine qui possède deux mille ans d'histoire.
- (16) Une `<fruit>orange</fruit>` a deux fois plus de potassium qu'une banane.
- (17) Le ciel prend des nuances `<couleur>orange</couleur>` pendant le lever et le coucher du soleil

Nous remarquons que dans la phrase (14) l'entité *Orange* désigne la société française de télécommunications, dans (15), *Orange* est la ville qui se trouve au sud de la France, dans (16), *orange* se réfère au fruit comestible de l'oranger ; et quant à la phrase (17), *orange* désigne la couleur formée du mélange de jaune et de rouge.

Les noms de personnes n'échappent guère à ce phénomène puisque dans de nombreux cas des patronymes, tels que ceux de personnages historiques, sont utilisés pour désigner une entreprise, un lieu public ou un phénomène. Un exemple récurrent est celui du nom *Washington*, qui peut indiquer l'état des États-Unis ou faire partie d'un nom comme celui de *George Washington*. Un autre exemple est le mot *Irma* lorsqu'il ne décrit pas une personne mais l'ouragan *Irma*, un événement météorologique : *Après Irma, la population est dans un état psychologique médiocre.*

Métonymie

La métonymie en tant que phénomène linguistique se définit comme une figure de style. Elle désigne le fait de se référer à un objet ou à un concept en empruntant un terme qui correspond normalement à un autre objet ou concept en lien direct ou indirect avec lui. Par exemple, la métonymie est utilisée pour désigner :

- une date à la place d'un événement (phrase (18), (19))
- l'auteur à la place d'une œuvre (phrase (20))
- une organisation ou le nom d'un lieu à la place de l'ensemble d'individus (phrase (21), (22))
- un lieu d'origine à la place d'un produit (phrase (23))

Dans l'exemple (18) et (19), les dates du *14 juillet 2016* et du *11 septembre 2001*, servent à désigner les événements qui se sont produits ces jour-là :

- (18) *La douleur du 14 juillet 2016 « encore vive », Nice renonce à son feu d'artifice prévu le 15 août.*
- (19) *Nous n'oublierons jamais les horreurs du 11 Septembre 2001.*

Dans l'exemple (20), l'entité nommée *Picasso* désigne l'œuvre d'art produit par son auteur *Pablo Picasso*.

- (20) *Un Picasso a été vendu pour 67,45 millions de dollars à New York.*

Dans l'exemple (21), le sigle OMS est utilisé pour désigner l'ensemble d'individus qui représentent l'Organisation Mondiale de la Santé :

- (21) *"L'épidémie du choléra en République Démocratique du Congo atteint des proportions inquiétantes..."*, écrit l'OMS dans un communiqué.

Dans l'exemple (22), l'entité toponymique États-Unis prend la place de l'ensemble d'une communauté d'individus, exactement comme pour les noms d'organisation dans l'exemple précédent :

- (22) *Les Etats-Unis ont diffusé dimanche soir à leurs 14 partenaires du Conseil de sécurité de l'ONU un texte remanié....*

Dans l'exemple (23), l'entité nommée *Coulommiers* se réfère au fromage à base de lait de vache car il provient de la commune française, *Coulommiers*, située dans le département de Seine-et-Marne.

- (23) *Le Coulommiers est sans doute aussi ancien que le Brie.*

Référents multiples

Un autre phénomène vient s'ajouter aux problèmes de catégorisation, celui de référents multiples qui est en lien avec la notion d'homonymie. En effet, nous parlons de référents multiples lorsqu'une entité, par exemple, du type personne a plusieurs propriétés. Cette notion-là rencontre ainsi celle de « facettes sémantiques » de [Cruse](#) et est en lien avec la signification en contexte lorsque le sens d'une unité lexicale est activé selon son contexte d'apparition ([Jacquet, 2006](#)). Ainsi dans la phrase (24) :

- (24) *Dans « Killing Gunther », Arnold Schwarzenegger interprète le rôle d'un tueur à gages à la retraite.*

L'entité nommée *Arnold Schwarzenegger* correspond à celle du type personne aussi bien qu'à celle de l'acteur mais en aucun cas aux autres propriétés du personnage médiatique en tant que gouverneur républicain de Californie, bodybildeur ou célébrité. Cependant, comme [Poibeau \(2005\)](#) le rapporte pour Cadiot et Visetti, *il ne s'agit que de phénomènes de synecdoque et de métonymie, qui ruinent toute tentative directement référentielle mais qui appellent plutôt une analyse dynamique par « profilage » de sens en fonction du contexte.*

6.1.4 Conférences liées à la reconnaissance d'entités nommées

L'apparition du terme entité est fait à l'occasion de MUC-7 (Message Understanding Conference) en 1997 aux États-Unis. En 1987, MUC est la première conférence ayant

comme but de méthodes nouvelles et améliorées pour l'extraction d'information. L'objectif de cette série de conférences est l'évaluation des systèmes d'extraction d'informations (Le Pevedic et Maurel, 2016, Hatmi, 2014). Depuis, plusieurs autres campagnes d'évaluation ont vu le jour (tableau 6.5).

L'objectif central des conférences est que des équipes de travail du TAL puissent comparer leurs résultats dans des conditions égales, après avoir défini un protocole d'annotation, de typologie et d'évaluation pour chaque campagne. Dans la suite de cette section nous allons exposer quatre campagnes d'évaluation menées en lien avec la reconnaissance d'entités nommées.

Message Understanding Conference - MUC

La série de conférences MUC a été conçue pour promouvoir la recherche dans le domaine de l'extraction d'information. Grâce à Grishman et Sundheim (1996), nous disposons des informations autour de l'histoire de ces conférences. Tout d'abord, malgré le fait que nous les nommons conférences, il ne s'agit que des évaluations soumises par les participants lesquelles leur donnaient le droit d'assister à la conférence. La mission de participants, pour chaque MUC était le développement d'un système après avoir reçu les directives (échantillon de messages et instructions d'annotation) pour traiter ces messages. Ainsi, le système de chaque équipe de participants est testé sur des messages spécifiques et est comparé et évalué avec les mêmes messages traités manuellement.

Pour le MUC-1 (1987) et MUC-2 (1989) l'extraction d'information s'est faite à partir de messages militaires et se centrait sur dix formulaires (templates) à extraire (temps, agents, lieu, effet etc.). Le MUC-3 en 1991 se concentre sur l'extraction d'informations contenues dans des textes qui traitent des éléments de terrorisme en Amérique Centrale et du Sud (18 formulaires à extraire). En 1993 à l'occasion de MUC-5, une structure hiérarchique qui permet l'enregistrement plus facile d'informations a été introduite. Pour MUC-6 (1995) la tâche était divisée en sous-tâches afin d'identifier des technologies de composants en cours de développement pour l'extraction de l'information mais aussi, les fonctions qui seraient largement indépendantes du domaine, et réalisées automatiquement avec une grande précision. Pour cette raison, le terme entité nommée a été proposé afin d'identifier dans un texte les noms de personnes, les organisations, les lieux, etc. Avec MUC-7 (1998) nous avons acquis progressivement une extraction d'information de haut niveau avec les différentes tâches, telles qu'elles ont été formulées au sein de MUC-7 (Abramowicz, 2013, Declerck et Andre, 2002) :

- **Les entités nommées** (Named Entities, NE) : identification d'entités nommées (personne, lieux, temps, organisation etc.)
- **Les éléments des formulaires** (Template Element task, TE) : tâche qui peut être mise en relation avec les organisations, les personnes et autres entités concrètes reconnues
- **Les relations au sein de formulaires** (Template Relation task, TR) : extraction des informations relationnelles du type "employé de", "produit de", "lieu où" etc.

- **Les scénarios au sein de formulaires** (Scenario Template task , ST) : extraction d'informations autour des événements pré-spécifiés selon le domaine d'application et l'indication de leur rôle par les différentes entités nommées reconnues auparavant.
- **La coréférence** (Coreference task, CO) : identification d'expressions se référant à une même entité (pour plus d'information *cf.* la partie "Le traitement de la coréférence").

Le tableau 6.6 résume les caractéristiques des conférences MUC 3-7 selon les différentes tâches, telles qu'elles ont été formulées sur [Chinchor \(1998\)](#).

Tache d'évaluation	NE	CO	TE	TR	ST	Multiling.
MUC-3					✓	
MUC-4					✓	
MUC-5					✓	✓
MUC-6	✓	✓	✓		✓	
MUC-7	✓	✓	✓	✓	✓	

TABLEAU 6.6 – Taches d'évaluation de MUC-3 à MUC-7 d'après [Chinchor \(1998\)](#)

Conference on Natural Language Learning - CONLL

La conférence CONLL-2002 a été organisée pour réaliser la tâche de reconnaissance d'entités nommées indépendamment de la langue. Sur [Tjong Kim Sang \(2002\)](#), nous disposons des informations sur l'ensemble de données et la méthode d'évaluation établie, présentant un aperçu général des systèmes qui ont participé à la tâche et leurs performances. Au total quatre types d'entité nommées ont été concernées : les noms de personnes, les lieux, les organisations et diverses entités qui n'appartiennent pas aux trois autres types. La tâche consistait à reconnaître des entités nommées de deux langues de l'europe de l'ouest : l'espagnol et le néerlandais.

Pour CONLL-2003 les participants ont travaillé sur d'autres langues européennes : l'anglais et l'allemand. Parmi les seize systèmes de participants, le système de [Florian et al. \(2003\)](#) obtient les meilleures performances combinant quatre algorithmes à base d'apprentissage automatique (classificateur linéaire robuste, entropie maximale, apprentissage basé sur la transformation et modèle de Markov caché). La démarche pour CONLL-2002 et CONLL-2003 étaient assez similaire à celle des MUC ([Hatmi, 2014](#)).

Évaluation des Systèmes de Transcription d'Émissions Radiophoniques - ESTER

L'ESTER-2004 est une campagne d'évaluation dédiée à la reconnaissance d'entités nommées issues de documents d'émissions radiophoniques pour le français. [Meur et al. \(2004\)](#) nous fournissent des informations sur les conventions d'annotation manuelle d'enregistrements audio de journaux télévisés et radiophoniques en français pour le

développement des systèmes de reconnaissance automatique d'entités nommées. Ils décrivent l'entité nommée en tant qu'un terme générique définissant d'autres unités lexicales. Ces mentions peuvent faire référence à une seule et même entité : l'entité nommée, l'ellipse, la dénomination complétée par ajout de précision (*le président Jacques Chirac*), le surnom, l'expression faisant référence à l'EN sans la nommer (par exemple, *le président de la République*) et l'anaphore (par exemple, *Marie est à la maison, elle vient d'arriver*). Ils utilisent huit classes d'entités nommées dont chacune dispose de sous-catégories (*cf.* tableau 6.5). Ils ajoutent également l'étiquette *unk* (qui signifie incertain), ne faisant pas partie de ces classes, afin de désigner les entités nommées qui n'appartiennent à aucune des classes définies et pallier les incertitudes.

Évaluations en Traitement Automatique de la Parole - ETAPE

La campagne ETAPE-2011 et 2012 ont été consacrées à l'évaluation des technologies vocales. ETAPE se positionne dans la continuité des campagnes ESTER. Elle a été conçue pour faire progresser la production de ressources mais aussi la mise en relation des acteurs du domaine (Gravier et al., 2012). La campagne a mis en valeur trois axes technologiques : la segmentation, la transcription et l'extraction d'entités nommées.

L'évaluation ETAPE 2011, selon les informations fournies par ELRA¹ est dédiée au matériel télé-diffusé traitant les différents niveaux de parole spontanée et des locuteurs multiples en français. Comme il a été mentionné, l'originalité de la campagne ETAPE est le fait qu'elle ne concerne pas un type particulier d'émissions, comme par exemple, les actualités, mais plus généralement le matériel multimédia de qualité professionnelle.

Selon Nouvel (2012), l'annotation d'ETAPE était plus fine en comparaison avec celle utilisée pour la campagne ESTER 2. Les entités rencontrées dans le corpus ETAPE étaient : les noms de personnes, les fonctions, les organisations, les lieux, les productions humaines, les dates et les heures, les montants et les événements. Ces classes principales ont été divisées, par la suite, en trente-quatre sous-types, les composants de ces entités nommées. Dans le tableau 6.7 nous trouvons la répartition des classes avec leur pourcentage selon le type du corpus (ETAPE-train pour l'entraînement, ETAPE-dev pour les évaluations, ETAPE-test pour la phase de test).

1. European Language Association : <http://www.elra.info/en/>

Type d'entité	ETAPE-train	ETAPE-dev	ETAPE-test
production	7%	7%	8%
montant	11%	11%	12%
fonction	13%	10%	12%
lieu	11%	13%	14%
organisation	15%	15%	13%
personne	30%	30%	29%
temps	13%	14%	12%

TABLEAU 6.7 – Répartition en pourcentage des types d'EN pour chaque partie d'ETAPE dans [Nouvel \(2012\)](#)

6.1.5 Métriques d'évaluation

Les mesures les plus couramment utilisées lors de la phase de l'évaluation dans la majorité des campagnes d'évaluations sont le rappel, la précision et la F-mesure (Rijsbergen 1979) ou bien le SER (Slot Error Rate) ([Makhoul et al., 1999](#)). D'autres variantes existent également pour mesurer les performances des systèmes.

Rappel et Précision

Le Rappel et la Précision sont deux métriques qui sont définies lorsque l'ensemble de documents pertinents et non pertinents sont reconnus dans un corpus, de telle sorte, nous pouvons évaluer les quantités définies.

Le **Rappel** mesure le rapport entre le nombre des documents correctement identifiés dans un corpus annoté automatiquement et le nombre des documents contenus dans le corpus annoté manuellement. Autrement dit, la mesure de rappel calcule la quantité de documents trouvés.

$$\text{Rappel} = \frac{\text{Vrais Positifs}}{\text{Vrais Positifs} + \text{Faux Négatifs}} \quad (6.1)$$

La **Précision** mesure le rapport entre les documents correctement identifiés et le nombre des documents repérés automatiquement par le système. Autrement dit, la mesure de la précision calcule la qualité de documents trouvés

$$\text{Précision} = \frac{\text{Vrais Positifs}}{\text{Vrais Positifs} + \text{Faux Positifs}} \quad (6.2)$$

La **F-mesure** est la moyenne harmonique pondérée du rappel et de la précision. Le coefficient b équilibre le Rappel et la Précision ($b = 1$), mais aussi accorde plus d'importance à l'une des deux mesures : le Rappel ($b > 1$) ou la Précision ($b < 1$).

$$F - mesure = \frac{(1 + b^2) * Précision * Rappel}{b^2 * Précision + Rappel} \quad (6.3)$$

Slot Error Rate - SER

Le SER est une mesure d'erreur qui combine les différents types d'erreur directement, sans avoir à recourir à la précision et au rappel comme mesures préliminaires. Il est analogue à un taux d'erreur qui est utilisé pour mesurer les performances de reconnaissance vocale, il est destiné à être une mesure de coût pour l'utilisateur pour le système qui fait les différents types d'erreurs (Makhoul et al., 1999). Nous adoptons et adaptons cette mesure d'évaluation pour nous en servir lors de l'évaluation entre un corpus manuellement annoté et un autre annoté automatiquement. Avec cette métrique nous sommes capables de prendre en compte les types et les frontières de l'annotation.

Grâce à SER nous pouvons compléter les mesures précédentes vu qu'il prend en compte :

- le nombre de entités/attributs qui figurent dans le corpus annoté manuellement mais qui ne sont pas alignées dans le corpus annoté automatiquement (D = « suppression »);
- le nombre de entités/attributs qui figurent dans le corpus annoté automatiquement mais qui ne sont pas alignées dans le corpus annoté manuellement (I = « insertion »);
- le nombre de entités/attributs qui figurent dans le corpus annoté manuellement qui ont été ramenées par le corpus annoté automatiquement avec type et frontières incorrects (TF= « type + frontière »);
- le nombre de entités/attributs qui figurent dans le corpus annoté manuellement et qui ont été ramenées par le corpus annoté automatiquement avec typage correct mais des frontières incorrectes (T = « type »);
- le nombre de entités/attributs qui figurent dans le corpus annoté manuellement et qui ont été ramenées par le corpus annoté automatiquement avec des frontières correctes mais un typage incorrect (F = « frontière »);
- le nombre des entités/attributs attendus (R = « référence »).

$$SER = \frac{D + I + TF + 0,5 * (T + F)}{R} \quad (6.4)$$

6.1.6 Applications

Dans la suite de cette section, nous allons exposer trois applications de l'extraction d'informations impliquant la reconnaissance des entités nommées : la fouille de données, les agents conversationnels et l'anonymisation automatique des documents .

La fouille de données

La fouille de données est le processus de recherche, d'extraction et d'analyse de l'information utile contenue dans de gros volumes de données. Ces données peuvent être textuelles et représentées sous diverses formes, ayant une bonne, ou non-conforme, structure lexicale, syntaxique ou grammaticale. Par exemple, l'analyse de texte provenant du web comportant des fautes d'orthographe et diverses formes abrégées. Les principaux objectifs de la fouille de données textuelles est l'identification et le classement de catégories textuelles, ainsi que l'accès à l'information pertinente.

L'extraction d'entités nommées n'est qu'une étape parmi d'autres dépendant de l'objectif et du contexte de la fouille puisqu'elle apporte des informations sur le contenu du texte. Par exemple, l'identification de tendances et d'influences sur les réseaux sociaux en lien avec les entités nommées présentes dans les messages permet d'offrir des indices sur l'appréciation globale et spécifique des produits, des marques, des événements, etc. Voir par exemple la figure 6.1 qui concerne un message posté sur Twitter exprimant l'avis sur un produit.

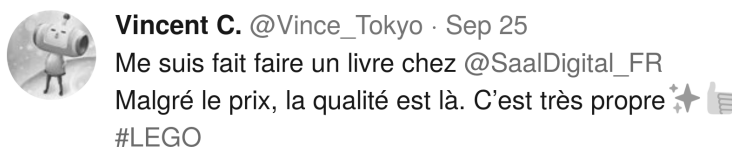


FIGURE 6.1 – Message posté sur Twitter exprimant l'avis sur un produit

Agents conversationnels

Un agent conversationnel, en anglais *chatbot*, est un logiciel capable de dialoguer avec un utilisateur par échange vocal ou textuel. Au travers d'un service de conversation automatisée effectuée en langage naturel, un « chatbot » est un support utilisé en ligne par les services clients ou des commerçants via une messagerie instantanée. Historiquement, précurseur du chatbot est ELIZA, le premier agent conversationnel créé en 1966 par Joseph Weizenbaum. Le programme simulait la conversation d'un psychologue en reformulant des affirmations de son interlocuteur en forme question.

La mission de l'agent conversationnel est de converser, répondre à des requêtes et fournir des services. Actuellement, intégrés aux sites webs et aux interfaces des réseaux sociaux, les agents donnent la possibilité aux utilisateurs d'une application de contacter

un service client, pour réserver un voyage, passer une commande, ou même faire des rencontres, etc.

Le développement de ces agents, afin de rendre les messages des utilisateurs compréhensibles, se fonde sur l'utilisation des techniques : 1) de l'apprentissage automatique et 2) des règles de *hard-code*¹. L'utilité de l'apprentissage automatique réside, notamment dans l'identification du contexte du message mais aussi pour la reconnaissance des entités nommées. En effet, parmi les différents traitements autour du TAL, tels que la segmentation de phrases, la tokenisation, l'analyse morphosyntaxique et la lemmatisation, un agent conversationnel nécessite de passer par l'identification des entités nommées. Le processus de génération de réponses automatiques doit, d'une part, utiliser le contexte de la conversation et d'autre part les entités extraites du précédent message saisi par l'utilisateur.

Perez-Marin (2011) définit la reconnaissance d'entités nommées comme une sous-tâche pour l'extraction d'information pour les systèmes de dialogue. Il souligne l'important rôle que la reconnaissance détient vis à vis à l'extraction de contenu intelligent, puisque le sens d'une phrase peut dépendre de l'interprétation que nous donnons à l'entité nommée. Avec la phrase (25) qui suit, Perez-Marin met en évidence le fait qu'une entité nommée peut correspondre à deux catégories différentes et selon ces catégories nous interprétons les deux phrases (26) et (27) de différentes façons :

- (25) I search the world according to Garp.
- (26) I search the world according to Garp PERS.
- (27) I search the world according to Garp FILM.

La figure 6.2 illustre l'agent conversationnel du comparateur de vols, Skyscanner, intégré dans la messagerie de Facebook, dont nous pouvons observer que la place d'entités nommées toponymiques joue un rôle central pour la recherche de vols.

1. Le hard-coding se réfère au développement de logiciels qui intègrent des données d'entrée ou de configuration directement dans le code source d'un programme ou d'un autre objet exécutable.

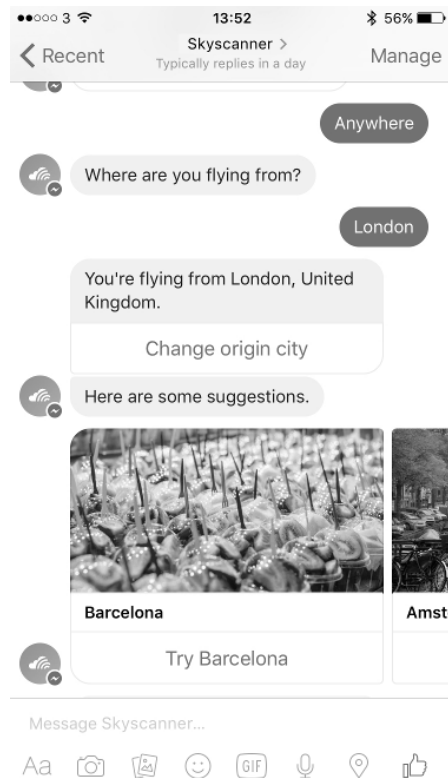


FIGURE 6.2 – Agent conversationnel Skyscanner pour la recherche de vols

Anonymisation automatique des documents

L'anonymisation de données consiste à supprimer toutes les données permettant l'identification d'une personne (physique ou morale). Les données à anonymiser peuvent être nominatives (nom, prénom, etc.) et indirectement nominatives, comme par exemple, une adresse, un numéro de téléphone, une adresse IP internet, etc.

Il s'agit d'un processus nécessaire pour la distribution de données, telles que les données médicales ou juridiques contenant des informations personnelles sensibles. Ce processus est étroitement lié avec la reconnaissance d'entités nommées. En effet, il impose préalablement la reconnaissance automatique des entités à anonymiser. Dans [Plamondon et al. \(2004\)](#), la première étape de la description du système d'anonymisation automatique de décisions de justice est la reconnaissance automatiquement des noms, dates, lieux et d'autres entités nommées pouvant constituer des renseignements permettant d'identifier les personnes visées par les restrictions à la publication.

Dans le cas de textes bruités, [Patel et al. \(2013\)](#) décrivent un système pour l'anonymisation d'un corpus de SMS qui est fondé sur deux méthodes : 1) une à base des règles heuristiques et l'utilisation des dictionnaires des mots et 2) une autre fondée sur l'apprentissage supervisé à l'aide d'arbres de décision. Ils tiennent à préciser le fait que ce processus ne peut pas être exclusivement automatique, puisqu'il est obligatoire

de passer par une phase de validation manuelle afin de garantir l'absence de données personnelles.

6.2 Reconnaissance des entités nommées dans des textes formels et bruités

Il existe trois types d'approches pour la reconnaissance d'entités nommées (NER) : les systèmes d'ingénierie des grammaires, souvent appelés à base de règles, symboliques ou à base de connaissances ; les systèmes basés sur l'apprentissage automatique, et les systèmes hybrides combinant les deux approches précédentes. De nombreux travaux fournissent un large panorama des approches évoquées, notamment [Fourour](#) fait un compte rendu des travaux réalisés en français et en anglais selon la méthode de traitement ([2002](#)) ainsi que selon les systèmes utilisés ([2004](#)). De même, plusieurs autres systèmes sont décrits dans [Poibeau](#) ([2005](#)).

6.2.1 Textes formels

Le point commun des travaux précédents sur la reconnaissance d'entités nommées est d'être centrés autour de l'identification des entités nommées dans des textes issus de dépêches de presse, ou de publications scientifiques ([Ehrmann, 2008](#), [Fourour, 2002](#), [Friburger, 2002](#), [Martineau et al., 2007](#), [McCallum et Li, 2003](#)). Ainsi, le caractère bien structuré des données, et le choix d'une typologie des entités nommées assez standard et générale (organisations, personnes, lieu), a rendu possible, tout au long des années, le développement de systèmes robustes caractérisés par leurs hautes performances.

Dans ce sens, [Martineau et al. \(2007\)](#) mentionnent le fait que toutes les approches offrent des taux de reconnaissance supérieurs à 90%, même si l'attribution des catégories reste un processus complexe. Cependant, le vrai défi concerne, comme [Palmer et Day \(1997\)](#) le stipulent, l'application des systèmes de reconnaissance d'entités nommées à des textes moins conformes, tels que les textes particuliers, non journalistiques ou ceux obtenus par la reconnaissance optique de caractères (OCR).

Nous ajouterons qu'au-delà du travail autour de la reconnaissance des entités nommées dans des textes moins conformes, il y a également des défis à relever concernant la structuration, normalisation, validation, et enrichissement des entités nommées ainsi que leur liaison et regroupement. Ceci sans oublier d'autres aspects comme le multilinguisme ou le traitement des entités avec des structures syntaxiques plus complexes que celles appartenant aux classes personne, organisation, localisation ou date.

En résumé, si le problème de l'identification des entités nommées semble alors résolu grâce aux hautes performances des méthodes et des outils conçus jusqu'à présent, la réalité est toute autre. En effet, hors de données textuelles bien formées et selon la langue visée et l'approche utilisée, plusieurs problèmes sont à soulever :

-
- Segmentation des phrases
 - Segmentation des unités lexicales
 - Traitement des mots inconnus (OOV- Out Of Vocabulary)
 - Annotation morphosyntaxique
 - Désambiguïsation lexicale

6.2.2 Textes bruités

Une première question à se poser concerne alors la conception de méthodes performantes sur des textes dégradés. Autour de cette interrogation nous trouvons des travaux qui reposent sur des textes bruités ou informels publiés dans les réseaux sociaux, les blogs, les courriers électroniques, la communication numérique plus généralement avec les SMS et les chats, la transcription automatique de la parole et la production de documents à travers la reconnaissance OCR. Ainsi, nous continuons en évoquant quelques travaux réalisés autour de l'identification des entités nommées dans des textes bruités.

Documents OCRisés

Un des premiers travaux concernant l'extraction d'entités nommées dans des textes produits par l'OCR est celui de [Miller et al. \(2000\)](#). Le but de leur démarche est l'analyse de transcriptions humaines de textes journalistiques sans casse ou ponctuation, ainsi que celles de textes issus de transcription automatique de la parole erronée et de textes OCR.

[Grover et al. \(2008\)](#), quelques années plus tard, proposent un prototype de méthodes basées sur des règles pour l'identification d'entités de type lieu et personne. Les textes utilisés pour évaluer le système sont des documents océrisés contenant des erreurs de transcription (plusieurs caractères non-alphabétiques, manque de majuscule pour certaines entités, etc), issus de textes historiques contenus dans des archives du parlement britannique datant de la fin du 17e jusqu'au début du 19e siècle. Les auteurs affirment que le système se montre plus performant pour reconnaître des entités du type personne que pour celles du type lieu, en supposant qu'une telle conclusion peut être due au fait que les noms de personnes dépendent plus de la recherche de motifs dans le texte (preuves externes), alors que les lieux dépendent plus de ressources linguistiques.

[Packer et al. \(2010\)](#) utilisent, également, des documents historiques océrisés afin de reconnaître les noms de personne grâce à la mise en place de quatre approches différentes. Ces approches font appel respectivement à l'utilisation des dictionnaires, des expressions régulières, des modèles de Markov caché et des champs aléatoires conditionnels (CRF). Selon les auteurs, la combinaison de méthodes permet d'obtenir de meilleurs scores de performance pour la REN.

De leur côté, [Dinarelli et Rosset \(2012\)](#) proposent de passer par une étape de normalisation à trois niveau : resegmentation de phrases, retokenisation de mots et correction d'erreurs produites lors de la phase d'océrisation. Après le prétraitement, les données sont utilisées pour entraîner un système de REN basé sur l'utilisation d'un

modèle CRF et d'une grammaire non contextuelle.

Plus récemment, [Sagot et Gábor \(2014\)](#), proposent une méthode fondée sur des règles grammaticales définies pour chaque type d'entité nommée afin de corriger des formules chimiques, des dates ou des adresses. Mais comme [Kooli \(2016\)](#) l'indique, le plus grand inconvénient de cette méthode est le besoin de prédéfinir toutes les règles de correction.

Communication virtuelle

Plusieurs travaux sont réalisés sur la reconnaissance d'entités nommées contenues dans des messages publiés sur les réseaux sociaux. [Liu et al. \(2011\)](#) décrivent un modèle d'apprentissage semi-supervisé qui, dans une première phase, effectue un pré-étiquetage des preuves globales (internes et externes) sur des messages du type tweet, tandis que dans un deuxième temps, un modèle à base de CRFs procède à un étiquetage séquentiel pour récupérer des informations fines codées dans les tweets. Le système combine les CRFs et un classificateur fondé sur l'algorithme des k voisins les plus proches (k-nearest neighbor ou kNN). Les performances sont exposées à travers une série d'expériences qui prouvent l'efficacité de la méthode et, en particulier, l'aspect positif de l'apprentissage KNN et de l'apprentissage semi-supervisé.

Plus tard, [Liu et al. \(2012\)](#), reviennent avec un nouveau modèle d'apprentissage qui utilise une caractéristique unique pour indiquer si une paire de mots se réfère à une même entité. Les résultats issus d'expérimentations menées par les auteurs montrent que la méthode produit une meilleure F-mesure pour la REN et une meilleure précision pour la normalisation des entités nommées en comparaison avec une approche de base.

Dans [Ritter et al. \(2011\)](#), le système proposé est fondé sur des CRFs et une approche supervisée, basée sur un modèle dénommé LabeledLDA ([Ramage et al., 2009](#)), pour classer les entités nommées. Les résultats d'évaluation du système montre une F-mesure de 66% sur un corpus composé de 2 400 tweets. Par ailleurs, les auteurs mentionnent le fait que le système obtient une amélioration de 25% de F-mesure comparé à l'approche de classification d'entité nommées non-supervisée proposée par [Collins et Singer \(1999\)](#) lorsqu'elle s'applique à des messages Twitter.

Nous trouvons, également, un certain nombre de travaux sur l'identification d'entités nommées contenues dans les SMS. Les SMS moins structurés que les tweets, où nous trouvons parfois des marqueurs pour les entités nommées (#Paris, #NY, #Jan25, @Jean-Pierre), constituent aussi des documents bruités.

Dans [Ek et al. \(2011\)](#), est décrit un système dédié à la reconnaissance des entités nommées dans les SMS écrits en suédois. Ce système fonctionne sur des téléphones Android afin de reconnaître des entités du type personne, lieu, date, heure et numéro de téléphone. L'approche utilisée est fondée sur une combinaison d'expressions régulières et des classificateurs utilisant la régression logistique. Les performances du système atteignent une F-mesure de 86% pour des reconnaissances strictes et 88% pour des reconnaissances partielles. Les auteurs signalent l'importance du manque de corpus de

SMS qui limite la possibilité d'entraîner un bon modèle capable de traiter la plupart des cas.

En outre, les propriétés des SMS rendent la reconnaissance complexe car l'étiqueteur morphosyntaxique utilisé a été conçu pour traiter des textes journalistiques, dont le style et la langue sont très différents de ceux des messages SMS. La brièveté et le manque de contexte dans ce type de messages diminue la capacité du système à extraire des informations.

En ce qui concerne, les SMS en anglais, [Polifroni et al. \(2010\)](#), proposent un système pour les entités nommées issus de messages vocaux et écrits en utilisant un modèle de régression logistique. La performance du système atteint les 88% de F-mesure pour les noms de personnes et de lieux.

6.2.3 Quelles sont les difficultés autour de la reconnaissance d'entités nommées ?

Lorsque nous sommes confrontés à la tâche de la reconnaissance d'entités nommées dans des textes formels, nous devons dépasser des problèmes récurrents du TAL. Comme nous avons vu dans la partie sous-section [6.1.3](#) « Difficultés autour de la catégorisation », la désambiguïsation d'entités nommées liée à l'homonymie, la métonymie et les multiples référents joue un rôle non trivial lors de ce processus. Mais que se passe-t-il lorsque nous traitons des textes contenant du bruit, par exemple, issus d'OCR ? Est-ce qu'il y a d'autres paramètres à prendre en compte ? Par la suite, nous énumérons certains problèmes de reconnaissance dans des textes formels qui peuvent s'accroître lorsque il s'agit de traiter des données bruités :

Les mots inconnus

Un phénomène assez fréquent empêchant le bon fonctionnement de systèmes de REN est la présence de mots inconnus, ou hors vocabulaire, contenus dans le texte. En effet, la plupart de systèmes pour détecter les entités nommées se basent sur les preuves internes et externes. Autrement dit, lorsque des mots inconnus se trouvent dans le contexte qui entoure l'entité à identifier, ou même, quand le mot inconnu est l'entité elle-même (preuve interne) les systèmes ne parviennent pas à accomplir la tâche de la REN.

L'exemple qui suit (figure [6.3](#)) illustre la capture d'écran de l'application de comparateur de vols Skyscanner disponible sur Facebook. Nous remarquons que l'agent conversationnel ne parvient pas de reconnaître que l'entité nommée erronément écrite "Cleremont Ferrant" correspond à l'entité toponymique "Clermont-Ferrand".

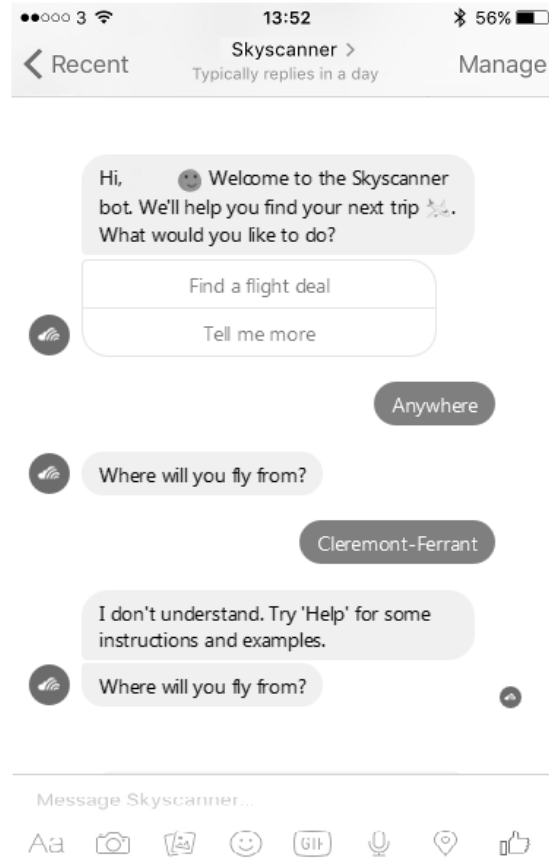


FIGURE 6.3 – Agent conversationnel avec un message contenant de mots inconnus

La coordination et ellipse

La coordination existe lorsque deux entités nommées sont juxtaposées l'une à côté de l'autre en effaçant l'un des constituants communs. Par exemple, quand il faut segmenter une ou plusieurs entités puisqu'elles partagent une partie commune dans la phrase (28) :

- (28) *La malédiction présidentielle qui a touché Nicolas Sarkozy et Cécilia, serait-elle en train de s'abattre sur Donald Trump et Melania ?*

Dans l'exemple les entités *Nicolas Sarkozy* et *Cécilia* sont coordonnées, ainsi que *Donald Trump* et *Melania*, la seconde (*Cécilia*) et quatrième (*Melania*) sont des ellipses :

- $\langle \text{PERS} \rangle \text{Nicolas Sarkozy} \langle \text{PERS} \rangle$ et $\langle \text{PERS} \rangle \text{Cécilia} \langle \text{PERS} \rangle$, qui correspond respectivement à *Nicolas Sarkozy* et *Cécilia (Sarkozy)*.
- $\langle \text{PERS} \rangle \text{Donald Trump} \langle \text{PERS} \rangle$ et $\langle \text{PERS} \rangle \text{Melania} \langle \text{PERS} \rangle$, qui correspond respectivement à *Donald Trump* et *Melania (Trump)*

L'émergence de nouvelles entités nommées

L'émergence d'entités nommées est un phénomène d'un essor continu avec les médias et le web qui envahissent jour après jour notre vie quotidienne : personnages politiques, acteurs, chanteurs, maladies, phénomènes météorologiques, etc. Elles ne sont jamais citées auparavant mais elles existent en raison d'une actualité, par exemple, l'*Ebola*, la maladie virale apparue en 2013. Contrairement, aux noms communs, il est impossible d'indexer la totalité des entités nommées dans un dictionnaire pour alimenter un système, alors il est nécessaire de prévoir d'autres méthodes qui à la fois reconnaissent les nouvelles entités mais aussi sont capables de les catégoriser.

La casse

Pour les systèmes à base de règles, la casse et la ponctuation sont deux éléments qui jouent un rôle assez important lors de la phase de reconnaissance. En effet, lorsqu'une entité nommée issue d'un texte bruité n'a pas la lettre initiale écrite en majuscule, elle peut être considérée comme un nom commun, produisant ainsi de l'ambiguïté. Dans les exemples qui suivent, un système doit reconnaître les entités $\langle \text{LOC} \rangle \textit{paris} \langle / \text{LOC} \rangle$ et $\langle \text{LOC} \rangle \textit{angleterre} \langle / \text{LOC} \rangle$ dans la phrase (29) en tant que toponymes et dans la phrase (30) *paris* en tant qu'un nom commun :

(29) *Tu paries sur paris - angleterre ?*

(30) *Les paris sont ouverts.*

La variation graphique, syntaxique et lexicale

La variation graphique caractérise les entités nommées tant à l'utilisation de la casse ((31), (32), (33)) que par la présence ou non de points à l'intérieur des sigles (34) mais aussi par l'existence de formes abrégées (35) :

(31) Président de l'Assemblée Nationale de la République

(32) Président de l'Assemblée nationale de la République

(33) président de l'Assemblée nationale de la République

(34) T.G.V, TGV, SNCF, S.N.C.F., U.S.A, USA

(35) Assedic (Association pour l'emploi dans l'industrie et le commerce), DSK (Dominique Strauss Khan)

La variation syntaxique concerne l'ordre et la disposition des entités dans la phase mais aussi le remplacement d'un constituant de l'entité nommée par un autre appartenant à la même classe morphosyntaxique.

(36) République française ; République de France

- (37) Mélenchon Jean-Luc ; Jean-Luc Mélenchon

La variation lexicale concerne les différentes formes qu'une même entité nommée peut avoir :

- (38) Charles de Gaulle, Général Charles de Gaulle, De Gaulle
 (39) Jacques Chirac, J.Chirac, Chirac, le Président Chirac, le Président de la République française
 (40) la Mont. Maudite, le mont Blanc, le massif du Mont-Blanc

Le traitement de la coréférence

La coréférence correspond à la relation qui marque au moins deux identifiants qui se réfèrent à une même entité [Chein et al. \(2015\)](#). [Mitkov \(1999\)](#), définit la référence en tant qu'anaphore et l'entité à laquelle elle se réfère comme antécédent. L'antécédent et l'anaphore sont utilisés comme expressions de référence et ont le même référent et nous les appelons coréférentiels. Par exemple :

- (41) L'Europe est un continent ou une partie des supercontinents de l'Eurasie et de l'Afro-Eurasie. Elle est parfois appelée le « Vieux Continent », par opposition au « Nouveau Monde » (l'Amérique).

Dans la phrase (41), l'entité toponymique *Europe* est l'antécédent du pronom personnel *elle* avec lequel elle établit une relation de coréférence. Traiter la coréférence signifie prendre en compte l'entité présente ailleurs dans le texte, à laquelle se réfère un groupe de mots d'une phrase. Une autre forme de coréférence est l'utilisation d'un groupe nominal pour désigner une même entité, par exemple, pour mentionner *Paris* nous pouvons utiliser également les appellations : *la capitale de la France* ou *la ville lumière*. Selon la nature de la coréférence, les approches pour sa résolution s'appuient soit sur les données linguistiques soit sur des approches à base d'apprentissage statistique.

6.3 Au-delà de l'identification des entités nommées

6.3.1 Normalisation

Comme nous l'avons indiqué en sous-section [6.2.3](#) les entités nommées sont sujettes à diverses variations : variation de casse et autres variations graphiques, lexicales et syntaxiques. La normalisation des entités nommées consiste à associer une forme de référence commune à toutes les variantes d'une entité. L'absence de cette étape de normalisation aboutirait à considérer comme différentes plusieurs entités extraites alors qu'elle ne sont, en réalité, que les variantes d'une entité unique. Considérons divers types d'entités pour lesquelles la normalisation s'avère essentielle.

Noms de personnes : Pour les noms de personnes la normalisation concerne, entre autre, la casse et l'ordre prénom, patronyme. Nous pouvons, par exemple, choisir comme forme normalisée celle qui comprend d'abord le prénom, puis le patronyme avec pour chacun d'eux la première lettre en majuscule et les autres en minuscules. Si le nom comporte une particule, celle-ci est écrite en minuscule dans la forme normalisée. Par exemple, si dans un titre, très souvent écrit en lettres capitales (majuscules), nous rencontrons l'entité « DOMINIQUE DE VILLEPIN », il faut produire la forme normalisée « Dominique de Villepin ».

Dans certains cas l'ordre des composants de l'entité concerne également le patronyme quand celui-ci est composé. Pour l'entité « Geneviève de Gaulle-Anthonio » nous trouvons fréquemment la forme « Geneviève Anthonioz de Gaulle ». Nous noterons qu'il faut, également, traiter la présence ou l'absence de trait d'union. Dans certains cas la normalisation peut faire appel à la forme canonique du patronyme ou du prénom. Nous pouvons trouver dans des titres ou dans le cours d'un texte des prénoms dont un accent est omis sur une lettre majuscule. Il est nécessaire de rétablir l'accent manquant : RENE → RENÉ, Eric → Éric. Enfin, pour les langues à cas dans lesquelles les prénoms et/ou patronymes se déclinent comme le serbe ou le grec, il faut clairement utiliser la forme canonique (Kyriacopoulou et al., 2011, Krstev et al., 2005).

Entités numériques : Toutes les entités comportant des données numériques (dates, données monétaires, adresses, unités de mesures, etc) font l'objet de normalisations complexes. La première consiste à transformer en chiffres les nombres écrits en lettres. Cette transformation rend l'information plus compacte et permet la comparaison de diverses occurrences d'entités de même type. Nous analysons deux types d'entités : les dates et les données monétaires.

- **Les dates :**

Si nous considérons la date « 18 octobre 2017 », elle peut prendre de nombreuses formes : « dix-huit octobre deux mille dix-sept », « 18 octobre 2017 », « 18 oct 2017 », « 18 oct. 2017 », « 18/10/2017 », « 18-10-2017 », « 18.10.2017 », « 18/10/17 », « 18-10-17 », « 18.10.17 », etc. La forme normalisée, souvent choisie, qui doit être produite à partir de n'importe quelle des formes précédentes est « 2017-10-18 » (ou même 20171018). Le format étendu avec tiret est plus aisément lisible par un humain, en revanche celui sans tiret (format de base) est directement utilisable par une machine. En effet, cette dernière permet de comparer facilement deux dates. Le « 18 octobre 2017 » est supérieur (postérieur) au « 3 janvier 2017 » (20171018 > 20170103).

Cette forme normalisée est issue de la norme internationale ISO 8601. Cette norme permet d'étendre la date stricte à la prise en compte de l'heure. Ainsi le « 18 octobre à trois heures de l'après-midi » est représenté par « 2017-10-18T15 :00 :00 » (T pour TIME). Remarquons qu'ici il faut faire attention à la transformation « trois heures de l'après-midi en 15 heures » (représentation de l'heure sur 12 ou 24 heures). La norme 8601 concerne également la normalisation des intervalles de dates ainsi que celle d'autres représentations de la date courante. Dans le domaine de la comptabilité

des entreprises, par exemple, nous faisons appel à une représentation qui utilise le numéro de la semaine dans l'année.

- **Les données monétaires :**

Les entités monétaires comportent, comme les dates présentées auparavant, des données numériques exprimées en chiffres ou en lettres. Cependant, les nombres ont une amplitude de valeurs bien plus étendue que celle qui concerne les dates. En effet, nous devons pouvoir traiter des sommes allant de plusieurs milliards à quelques centimes. Ceci induit des normalisations sur la forme même du nombre avant de prendre compte l'unité monétaire concernée.

Par exemple, la forme un « demi milliard » peut être exprimée également sous les formes : 0,5 milliard ; 500 millions ; cinq cents millions ; 500 000 000 ; 500.000.000 ; 500000000. Les deux dernières formes constituent des formes normalisées souvent employées. Le choix entre ces deux formes dépend de l'utilisation envisagée. La première permet une lisibilité aisée de la valeur numérique, la seconde est directement utilisable pour des calculs.

Une fois la partie numérique proprement dite traitée, il faut normaliser le nom de la devise utilisée. Les monnaies *livre*, *dollar*, *euro*, *yen* peuvent être respectivement représentées par les symboles £, \$, €, ¥. La forme normalisée utilisée pour les données monétaires est issue de la norme internationale ISO 4217. Par exemple, « 10 000 yen » ou « 10000 ¥ » est normalisé « 10.000 JPY ». Dans cette norme chaque monnaie est représentée par un code de trois lettres en majuscules.

Dans le cas des dates, comme celui des monnaies, l'utilisation de normes internationales va au delà de l'obtention d'une forme normalisée. Les formes, ainsi produites, peuvent être considérées comme des formes pivot facilitant la traduction d'une représentation à une autre au sein d'une même langue ou dans une autre langue (y compris la génération des nombres en lettres dans une langue cible).

6.3.2 Validation

La validation consiste à vérifier si l'entité extraite existe effectivement ou si elle est susceptible d'exister. Par exemple, dans le cas d'une adresse postale nous pouvons interroger une ressource interne ou externe pour vérifier l'exactitude de l'adresse. A tout le moins, il est possible de vérifier si le numéro de la voie (rue, boulevard, etc) est inférieur à une certaine valeur (un numéro de rue 401 semble peu plausible). Pour une date complète, (numéro de jour mois année) mentionnant un nom de jour, il est possible de vérifier grâce à une formule mathématique si le nom du jour correspond à la date exprimée.

6.3.3 Enrichissement

L'enrichissement d'une entité nommée est l'ajout à l'entité de données issues de ressources externes ou internes ou même de données issues du texte mais éloignées de

l'entité analysée. Par exemple, si un toponyme de type pays ou ville a été identifié, nous pouvons lui ajouter ses coordonnées terrestres (latitude, longitude), voire la taille de sa population. En ce qui concerne des entités de type personne, si nous considérons, par exemple, la phrase : « Un homme de cinquante-trois ans a été retrouvé inconscient. La consultation de ses papiers d'identité a révélé qu'il se nomme Xavier François. ». Il serait intéressant de pouvoir enrichir l'entité *Xavier François* de l'attribut/valeur `age=53`. De manière réciproque, au cas de la validation présentée ci-avant la donnée d'une date complète (numéro de jour mois année) peut être enrichie par le nom du jour calculable automatiquement.

6.3.4 Résolution

La résolution est l'utilisation de différentes méthodes pour rapprocher une entité d'une référence pertinente. Dans le cas de toponymes, nous pouvons utiliser la présence d'autres toponymes pour différencier, par exemple *Paris France* de *Paris Texas*. En ce qui concerne des entités de type personne, il s'agit par exemple de mettre en relation une forme complète de nom de personne (prénom, patronyme) avec des formes incomplètes : prénom absent, prénom réduit à son initiale. Dans un texte, la première occurrence d'un nom de personne est souvent exprimée sous sa forme complète *prénom patronyme*. Cette première occurrence peut être accompagnée d'une civilité ou d'une fonction (profession) voire des deux. Les occurrences ultérieures ne reprennent le plus souvent que le patronyme. Il s'agit alors de savoir jusqu'à quel point (jusqu'à quelle distance) il est correct de relier les formes ultérieures elliptiques à la première occurrence.

6.4 Conclusion

Dans ce chapitre, nous avons présenté une étude autour de travaux liés à la reconnaissance des entités nommées et ses aspects en lien avec l'identification des entités nommées contenues dans des textes formels et bruités.

En effet, l'objet de ce chapitre est l'entité nommée d'un point de vue théorique aussi bien que pratique. Nous nous intéressons, d'une part, à la définition, à la classification et ses problèmes, aux campagnes d'évaluation, aux applications impliquant la tâche de reconnaissance et aux mesures les plus couramment utilisées lors des évaluations. D'autre part, nous avons distingué la reconnaissance des entités nommées dans des textes formels, de celles de textes bruités. En présentant un état de l'art sur les travaux réalisés et en énumérant un certain nombre de problèmes de reconnaissance dans des textes formels, nous réalisons que ces problèmes peuvent s'accroître lors du traitement des données bruités.

Malgré le fait que des systèmes pour la reconnaissance des entités nommées pour des textes formels atteignent de hautes performances, nous avons pu constater que le traitement des entités nommées issues des textes bruités est plus compliqué par rapport aux textes formels. Au-delà de la reconnaissance des entités nommées qui ne se limite pas à la simple identification des entités nommées, nous avons décrit les défis

d'extraction d'information à relever : la normalisation, la validation, l'enrichissement ainsi que la résolution des entités nommées.

Dans le chapitre qui suit, nous allons présenter des approches pour l'extraction d'information à l'aide de Grammaires Locales Étendues ELG, capables de traiter les défis que nous avons cités ci-dessus.

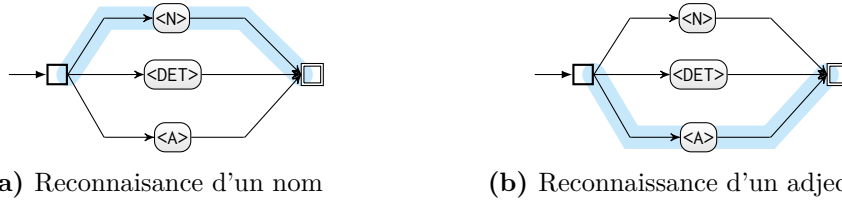
7

Grammaires locales étendues : approches pour l'extraction de l'information

Dans ce chapitre nous étudions quelques approches pour l'extraction d'information à l'aide des grammaires locales étendues. Nous abordons plusieurs problématiques : la désambiguïsation des catégories grammaticales à l'aide de méthodes fondées sur l'apprentissage automatique, l'analyse sémantique prédicat-argument à l'aide d'un moteur d'inférence logique, la recherche adaptative de motifs dans un dictionnaire électronique et la reconnaissance des entités nommées bruitées. Pour chacun des sujets, nous allons d'abord contextualiser la problématique et ensuite étudier comment utiliser le formalisme des grammaires locales étendues pour apporter des alternatives de traitement.

7.1 Désambiguïsation des catégories grammaticales

Rappelons qu'une grammaire est ambiguë (cf. définition 2.13) s'il existe plus d'un arbre de dérivation (cf. définition 2.12) pour une phrase du langage. Dans le même sens, l'automate qui modélise une grammaire locale est ambigu lorsque deux chemins réussis ont la même étiquette d'entrée.



Graphe 7.1 – Grammaire locale ambiguë pour le cas d'analyse nom <N> ou adjectif <A>

Dans certains cas, pour lever l'ambiguïté de l'automate, il est possible d'effectuer à l'avance sa détermination¹ afin d'obtenir un automate qui par définition n'est pas ambigu. Cependant, dans la pratique, comme dans les cas du graphe 7.1, pouvant avoir des ambiguïtés issues des catégories grammaticales², réaliser une telle opération n'est pas possible. En effet, à l'inverse des langages artificiels, les langages naturels sont très expressifs et bien connus pour avoir beaucoup d'ambiguïtés et il n'est pas alors toujours possible de les supprimer. Comme alternative, plusieurs approches peuvent être prises en compte. Nous considérons brièvement celles fondées sur la construction d'un automate du texte pour lever l'ambiguïté grammaticale, pour ensuite nous consacrer à certaines techniques qui peuvent être mise en place pour traiter l'ambiguïté (deux chemins réussis qui ont la même étiquette d'entrée) dans la conception d'une grammaire locale. Finalement, nous proposons une approche fondée sur la construction d'une grammaire locale étendue pour réduire l'ambiguïté des catégories grammaticales.

7.1.1 Construction d'un automate du texte

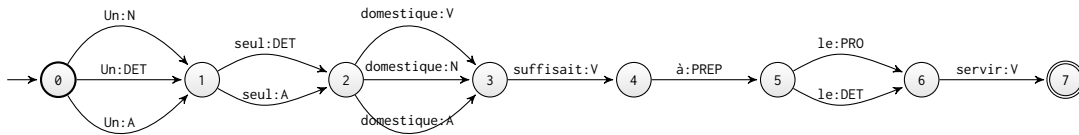


FIGURE 7.1 – Ambiguïté dans l'analyse de la phrase *Un seul domestique suffisait à le servir*

Prenons comme exemple la figure 7.1, elle modélise toutes les interprétations grammaticales de la phrase « *Un seul domestique suffisait à le servir* » au moyen d'un automate, aussi appelé *automate du texte*. Les codes grammaticaux possibles d'une unité lexicale, par exemple $Un \leftarrow \{A, DET, N\}$, étiquettent chacun une transition partant de la même origine : $0 \xrightarrow{Un:A} 1$, $0 \xrightarrow{Un:DET} 1$, $0 \xrightarrow{Un:N} 1$ pour indiquer que le mot « Un » peut être un adjectif (A), un déterminant (DET) ou un nom (N). La question est alors comment, à partir de l'automate du texte, lever l'ambiguïté, plusieurs approches peuvent être utilisées :

1. Pour une discussion sur la détermination des grammaires locales, voir [Sastre \(2011, pp. 199–204\)](#).
 2. Prenons comme exemple le mot *domestique*, avec les entrée lexicales **domestique**, .A+z1:ms:fs; **domestique**, .N+z1:ms:fs et **domestique**, domestiquer.V+P1s:P3s:S1s:S3s:Y2s:ms:fs, le graphe 7.1 reconnaît *domestique* à la fois comme un nom (N) et comme un adjectif (A)

- Modifier manuellement l'automate du texte à l'aide d'un éditeur graphique comme celui fourni par NOOJ (Silberztein, 2003, p. 156) ou par UNITEX (Paumier, 2016, p. 191),
- Appliquer des méthodes fondées sur la construction de règles symboliques comme celles des grammaires de contraintes (Karlsson et al., 1995) ou de grammaires de levée d'ambiguïté (ELAG, Laporte et Monceaux, 1999).
- Utiliser une stratégie fondée sur l'apprentissage statistique afin de linéariser l'automate et d'obtenir un seul chemin, c'est-à-dire, une seule interprétation du texte, comme celles qui s'appuient sur des modèles de Markov cachés (HMM, Kupiec, 1992), des champs aléatoires conditionnels (CRF, Lafferty et al., 2001), des machines à vecteurs de support (SVM, Giménez et Marquez, 2004) ou des réseaux de neurones (Andor et al., 2016),
- Faire appel à des méthodes capables de combiner les approches de désambiguïsation symboliques et statistiques dans un même processus, tel qu'est le cas dans l'étiqueteur hybride pour le français HYBRIDTAGGER (Sigogne, 2010).

Définition 7.1 (Entrée ambiguë). Une séquence en entrée d'une grammaire locale est ambiguë si elle est étiquetée par deux chemins réussis, en d'autres termes, lorsque la séquence est reconnue par plus d'un chemin.

Définition 7.2 (Sortie ambiguë). Étant donné une entrée ambiguë, l'ensemble des sorties produit par une grammaire locale ambiguë, est dénommée **sortie ambiguë** de la grammaire.

Considérons l'exemple 7.3 ci-après.

Exemple 7.3. Analysons les séquences (42) et (43) suivantes :

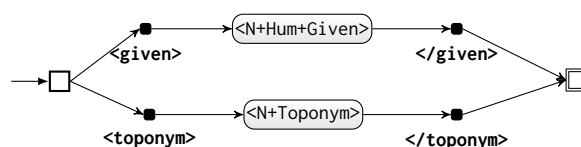
(42) *The role of Virginia Woolf was played by Nicole Kidman*

(43) *The state of Virginia is divided into 95 counties*

en tenant compte des entrées lexicales :

Virginia, .N+Hum+Given:fs
Virginia, .N+Toponym+Region

ainsi que de la grammaire locale définie par le graphe 7.2 :



Graphe 7.2 – Grammaire locale ambiguë

L'analyse de (42) produira deux sorties (séquences en noire) différentes :

(44) *The role of $\langle given \rangle$ Virginia $\langle /given \rangle$ Woolf was played by Nicole Kidman*

(45) *The role of $\langle toponym \rangle$ Virginia $\langle /toponym \rangle$ Woolf was played by Nicole Kidman*

il en est de même pour l'analyse de (43) :

(46) *The state of $\langle given \rangle$ Virginia $\langle /given \rangle$ is divided into 95 counties*

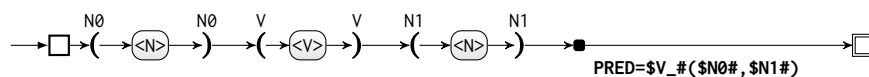
(47) *The state of $\langle toponym \rangle$ Virginia $\langle /toponym \rangle$ is divided into 95 counties*

Étant donné que la grammaire locale de l'exemple 7.3 produit deux analyses pour (42) et (43), elle est ambiguë. En outre, (44) et (45) ainsi que (46) et (47) sont des sorties ambiguës de la grammaire.

Remarquons qu'une sortie ambiguë d'une grammaire locale est produite lorsqu'une séquence en entrée est acceptée par plus d'un chemin de la grammaire. Observons également que pour qu'une grammaire locale soit ambiguë, il suffit qu'elle produise au moins une sortie ambiguë. Dans l'exemple 7.3, la sortie constituée par (44) et (45) suffit à rendre la grammaire ambiguë.

7.2 Analyse sémantique prédicat–argument

Considérons le graphe d'exemple 7.3 tiré de Silberztein (2003, p. 197), ce graphe est utilisé pour reconnaître des séquences $\langle N_0 \rangle \langle V \rangle \langle N_1 \rangle$ et produire en sortie une analyse sémantique exprimée dans un formalisme du type prédicat–argument telle que celui utilisé par Prolog (Clocksin et Mellish, 2003), un langage de programmation logique.



Graph 7.3 – Analyse prédicat–argument passif

Étudions la phrase :

(48) *John sees Mary*

Cette phrase décrit une action « see » impliquant deux personnes : *John* et *Mary*. Tandis que l'action est définie autour d'un noyau verbal $\langle V \rangle$, appelé **prédicat**, les deux noms, $\langle N_0 \rangle$ et $\langle N_1 \rangle$, définissent les participants de l'action et sont dénommés ces **arguments**. La représentation de cette relation prédicat–argument en Prolog est donnée par l'expression `see(John,Mary)`. Le graphe 7.3 est capable de produire des expressions de type prédicat–argument. En particulier, il est en mesure de reconnaître une phrase en entrée, telle que (48), et de produire une analyse sémantique en sortie, telle que (49) :

(49) PRED=see(John,Mary)

Notons que, quant à la capacité transformationnelle, l'analyse produite est puissante, en particulier, l'expression de sortie peut être adaptée facilement pour exprimer d'autres formalismes (Silberztein). Cependant, cette analyse est retrainte à produire une sortie passive, autrement dit, la grammaire locale est nullement capable de comprendre les expressions du formalisme qui est en train de produire, ainsi (49) n'a aucun effet ni sur (48) ni sur les autres phrases qui pourrait être reconnues. Ceci restreint énormément la capacité d'analyse sémantique. En effet, pour cet exemple, la seule stratégie pour créer une base de connaissances à partir des prédicats générés afin de faire des requêtes déductibles des faits est d'attendre la fin de l'analyse effectuée par la grammaire et ensuite d'utiliser les sorties résultantes comme entrée d'un interpréteur logique comme Prolog.

Naturellement, la question qu'on se pose est de savoir si en utilisant le formalisme des grammaires locales étendues il est possible de 1. Produire en sortie des expressions du type prédicat-argument, 2. Utiliser ces expressions pour ajouter, à la volé, des faits dans une base de connaissances et 3. Faire des requêtes à la base au cours des analyses. Nous appelons cette démarche une analyse sémantique active, ceci en opposition à l'analyse sémantique des grammaires locales classiques, comme celle du graphe 7.3, limitée à produire en sortie des expressions logiques.

L'analyse sémantique effectuée par une grammaire étendue permet d'utiliser deux niveaux de connaissances : d'abord un niveau *endogène* basée sur la reconnaissance des entrées et la construction de faits qui découle, ensuite un niveau *exogène* où le point de départ est une base de connaissances avec des faits préétablies, c'est-à-dire, avec des prédicats qui ne viennent pas de l'entrée mais de l'extérieur.

7.3 Recherche adaptative de motifs dans un dictionnaire électronique

Dans cette section nous étudions le problème de la recherche approximative de motifs dans un dictionnaire. Étant donné un dictionnaire W sur un alphabet fini Σ , un motif p sur le même alphabet, une distance dans l'espace métrique $\delta : \Sigma^* \times \Sigma^* \rightarrow \mathbb{R}^+$ et un seuil maximal autorisé k , le problème consiste à retrouver l'ensemble des entrées de W qui coïncide avec p tel que la distance δ entre chaque candidat et p est plus petite ou égale à k . Nous nous intéressons donc à toutes les occurrences de W qui peuvent être transformées en p avec k opérations au plus : $S = \{s \in W : \delta(s, p) \leq k\}$

Comme pour le problème de la recherche exacte de motifs¹, il existe deux grandes approches pour effectuer une recherche approximative. D'une part, celles fondées sur l'algorithmie, aussi appelées méthodes de recherche en ligne (*on-line*), qui permettent de résoudre une nouvelle instance du problème à chaque fois que l'algorithme est

1. Voir à ce sujet Faro et Lecroq (2011)

exécuté. D'autre part, les approches basées sur les structures de données, ou méthodes de recherche hors ligne (*off-line*), qui réalisent d'abord une phase de prétraitement de W pour ensuite répondre à toute requête impliquant p .

Le temps d'exécution des méthodes de recherche approximative en ligne ont une complexité linéaire en la taille du texte, ce qui les rend inefficaces pour exploiter de grands volumes de données textuelles (Boytsov, 2011). Dans le cas de la recherche approximative de motifs dans un dictionnaire, cette contrainte motive la réflexion sur la façon de concevoir des algorithmes hors-ligne plus efficaces.

Nous présenterons à la fin de ce chapitre une structure de données et des algorithmes qui permettent de chercher efficacement p en W avec une métrique δ et un seuil de distance k , ainsi que connaître combien de mots de W coïncident avec p et quelle est la position de ces occurrences. Nous verrons aussi comment l'approche proposée ne se limite pas à une seule métrique d'édition, mais peut aussi s'adapter à la nature du problème et des données.

7.3.1 Approche naïve

L'approche naïve pour rechercher approximativement un motif p dans un dictionnaire W consiste à parcourir toutes les entrées du dictionnaire en calculant chaque fois la distance entre p et l'entrée s , soit l'ensemble d'occurrences $S = \{s \in W : \delta(s, p) \leq k\}$.

Algorithme 1 : Recherche approximative d'un motif p dans un dictionnaire W

Entrées : p, k, W

Résultat : $S = \{s \in W : \delta(s, p) \leq k\}$

```

1  $n \leftarrow |p|$ ;
2  $s \leftarrow \{ \}$ ;
3 pour chaque  $s \in W$  faire
4    $d \leftarrow \delta(s, p)$ ;
5   si  $d \leq k$  alors
6     empiler  $s$  dans  $S$  ;
7   fin
8 fin
9 return  $S$ ;
```

Si la longueur moyenne d'une entrée est n , alors la complexité de cet algorithme est bornée par le coût du calcul de la distance d'édition effectuée à la ligne 4, soit $\mathcal{O}(n^2)$ en utilisant l'algorithme standard de programmation dynamique proposé par Wagner et Fischer (1974), en rapport à la taille du dictionnaire $|W|$, c'est-à-dire $\mathcal{O}(|W|n^2)$.

7.3.2 Motivation et objectifs

Les algorithmes de la littérature pour la recherche approximative de motifs se sont concentrés à traiter la nature du problème ainsi qu'à améliorer la complexité temporelle ou spatiale. Contrairement aux algorithmes pour la recherche exacte, peu de références existent faisant état d'approches qui soient efficaces à la fois en temps et en espace, par exemple qui mobilisent des méthodes hors-ligne avec une complexité sous-linéaire en la taille du texte en restant compétitives en rapport à l'utilisation de l'espace mémoire.

7.3.3 Distance d'édition

La distance d'édition entre deux mots est définie comme étant le nombre minimum d'insertions, de suppressions ou de substitutions de symboles nécessaires pour transformer un mot en un autre.

Pour le cas des mots de taille n , l'algorithme standard de programmation dynamique de [Wagner et Fischer \(1974\)](#) a une complexité en temps quadratique $\mathcal{O}(n^2)$. [Wong et Chandra \(1976\)](#) ont démontré que ceci est optimal dans le modèle restreint où l'on ne peut que comparer les caractères les uns aux autres. Les autres algorithmes connus, qui essentiellement calculent la distance d'édition en traitant des blocs de caractères comme un seul, ont une complexité en temps sous-quadratique. Notamment, pour un alphabet fini et un coût discret des opérations d'édition, la meilleure borne supérieure connue, due à [Masek et Paterson \(1980\)](#), est $\mathcal{O}(n^2/\log n)$ ¹. Concernant ce résultat, le problème d'avoir un coût d'opérations non limité aux rationnels est resté ouvert pendant deux décennies, jusqu'à ce que [Crochemore et al. \(2002, 2003\)](#) proposent un algorithme en temps $\mathcal{O}(n^2/\log n)$ ². La question de trouver une meilleure borne supérieure est, de nos jours, encore ouverte. Cependant, il pourrait ne pas en exister une, en particulier, [Backurs et Indyk \(2014\)](#) fournissent une preuve démontrant l'impossibilité d'existence d'un algorithme pour calculer la distance d'édition en temps fortement sous-quadratique, c'est-à-dire en temps $\mathcal{O}(n^{2-c})$ pour une constante $c > 0$, à moins que la conjecture SETH (*Strong Exponential Time Hypothesis*) sur la résolution du problème de satisfaisabilité booléenne (SAT) soit fausse³. Ce qui continuerait à restreindre le calcul à une complexité en temps quadratique avec un facteur logarithmique.

En raison du coût élevé du calcul de la distance d'édition, plusieurs approches ont vu le jour, parmi lesquelles celles fondées sur des schémas de filtrage-vérification comme les q-grammes.

1. Si la longueur du deuxième mot est égal à m et $n > m$, alors la borne supérieure est $\mathcal{O}(n \max(1, m/\log n))$

2. Entre autre, les auteurs montrent que pour des séquences compressibles, la borne supérieure est $\mathcal{O}(hn^2/\log n)$, où h est un nombre réel, $0 \leq h \leq 1$, qui mesure l'entropie de la séquence.

3. La preuve apportée par les auteurs consiste à démontrer que si le calcul de la distance d'édition peut être effectué en $\mathcal{O}(n^{2-c})$, pour une constante $c > 0$, alors une instance du problème SAT de N variables et M clauses sous forme normale conjonctive (CNF-SAT) pourrait être résolue en temps $M^{\mathcal{O}(1)} 2^{(1-\epsilon)N}$ pour une constante $\epsilon > 0$, ce qui contredirait la conjecture SETH formulée par [Impagliazzo et Paturi \(2001\)](#) qui revendique qu'un tel algorithme n'existe pas.

7.3.4 Recherche approximative dans un dictionnaire classique

T. Bocek (2007) présentent un algorithme hors ligne appelé FastSS (*Fast Similarity Search*) pour la recherche approximative de motifs. L'approche utilisée par FastSS met en œuvre une variante de la stratégie de génération de mots voisins (*word neighborhoods*) présentée auparavant par Myers (1994). L'algorithme se déroule en deux étapes : la première (indexation) : étant donnée un dictionnaire W de n entrées avec une longueur moyenne de m caractères et un nombre maximal d'erreurs autorisés égal à k , consiste à construire un dictionnaire T de taille $\mathcal{O}(nm^k)$ de toutes les k -suppressions de chaque entrée (*deletion neighbourhoods*). Ensuite, dans une deuxième étape (recherche), l'algorithme tente de créer un dictionnaire de taille $\mathcal{O}(m^k)$ de toutes les k -suppressions du motif p en les comparant à T en $\mathcal{O}(\log |T|)$ opérations. L'intuition sous-jacente est la suivante : précalculer et chercher les k -suppressions d'une entrée est plus rapide que traiter également les insertions et remplacements.

7.3.5 Dictionnaires électroniques du LADL

Comme nous l'avons évoqué au chapitre 3, les expressions décrites par les grammaires locales ne sont pas simplement des unités lexicales. En effet, elles peuvent aussi faire appel à des listes et des classes de mots contenus dans des lexiques qui sont stockés sous la forme de dictionnaires électroniques.

Ces derniers se distinguent des lexiques classiques par les informations associées à chaque unité lexicale qui constitue les entrées du dictionnaire. La capacité d'avoir recours aux informations morphosyntaxiques des dictionnaires représente une des caractéristiques les plus importantes des grammaires locales en leur conférant un grand pouvoir descriptif.

Les dictionnaires utilisés dans les grammaires locales sont ceux du LADL (Courtois et Silberztein, 1990, Courtois et al., 1997), le tableau 7.1 résume les types de dictionnaires disponibles.

NOM	FORMES	CATÉGORIE
DELAS	Canoniques	Mots simples
DELAF	Fléchies	Mots simples
DELAC	Canoniques	Mots composés
DELACF	Canoniques et fléchies	Mots composés
DELAP	Canoniques et fléchies	Mots phonétiques

TABLEAU 7.1 – Dictionnaires électroniques

Définition 7.4 (Trait). Un trait est une 2-tuple $T = (\text{attribut}, \text{valeur})$.

Définition 7.5 (Structure de traits). Une structure de traits $S = (t_1, t_2, \dots, t_n)$ est un ensemble ordonné de traits.

Définition 7.6 (Dictionnaire). Un *dictionnaire* $W = (s_1, s_2, \dots, s_n)$ est un ensemble ordonné de mots, aussi appelés entrées, où $n = |W|$ est le nombre d'entrées du dictionnaire.

Définition 7.7 (Dictionnaire morphosyntaxique). Un *dictionnaire morphosyntaxique* est un ensemble de tuples (d_1, d_2, \dots, d_n) .

7.3.6 k -suppressions

Nombre de k -suppressions. Soit un mot de longueur n . Alors le nombre de k -suppressions est égal à $f(n, k) = \sum_{m=1}^k \binom{n}{m}$.

Par exemple, le nombre de k -suppressions du mot *page*, listées au tableau 7.2, pour $k = 1..4$, est égal à :

1-suppressions. $f(4, 1) = \binom{4}{1} = 4$

2-suppressions. $f(4, 2) = \binom{4}{1} + \binom{4}{2} = 10$

3-suppressions. $f(4, 3) = \binom{4}{1} + \binom{4}{2} + \binom{4}{3} = 14$

4-suppressions. $f(4, 4) = \binom{4}{1} + \binom{4}{2} + \binom{4}{3} + \binom{4}{4} = 15$

Un extrait du nombre total de k -suppressions pour un mot de longueur $n \leq 10$ est donné dans le tableau 7.3.

$f(4,1)$	ε a g e	4
	p ε g e	
	p a ε e	
	p a g ε	
$f(4,2)$	ε a g e	10
	ε ε g e	
	ε a ε e	
	ε a g ε	
	p ε g e	
	p ε ε e	
	p ε g ε	
	p a ε e	
$f(4,3)$	p a ε ε	14
	ε a g e	
	ε ε g e	
	ε ε ε e	
	ε ε g ε	
	ε a ε e	
	ε a ε ε	
	ε a g ε	
	p ε g e	
	p ε ε e	
$f(4,4)$	p ε ε ε	15
	p ε g ε	
	p ε ε e	
	p ε ε ε	
	p ε g ε	
	p a ε e	
	p a ε ε	
	p a g ε	

TABLEAU 7.2 – k -suppressions du mot *page*

n										
1	1									
2	2	3								
3	3	6	7							
4	4	10	14	15						
5	5	15	25	30	31					
6	6	21	41	56	62	63				
7	7	28	63	98	119	126	127			
8	8	36	92	162	218	246	254	255		
9	9	45	129	255	381	465	501	510	511	
10	10	55	175	385	637	847	967	1012	1022	1023
	1	2	3	4	5	6	7	8	9	10
	k									

TABLEAU 7.3 – Nombre de k -suppressions pour un mot de longueur n

Déroulement. Pour un mot de longueur n et un entier k , le nombre de k -suppressions est égal à la somme partielle des $k + 1$ premiers nombres de la $n^{\text{ième}}$ ligne du triangle de Pascal moins 1 :

$$\sum_{m=0}^k \binom{n}{m} - 1$$

ou, en simplifiant,

$$\binom{n}{0} + \sum_{m=1}^k \binom{n}{m} - 1 = \sum_{m=1}^k \binom{n}{m}$$

nous avons alors les cas suivants :

$$f(n, k) = \begin{cases} 0 & \text{si } k = 0 \\ n & \text{si } k = 1 \\ 2^k - 1 & \text{si } k = n \\ 2^k - 2 & \text{si } k = n - 1 \\ \sum_{m=1}^k \binom{n}{m} & \text{si } k \geq 2 \end{cases}$$

i					
5	\$	p	a	g	e
2	a	g	e	\$	p
4	e	\$	p	a	g
3	g	e	\$	p	a
1	p	a	g	e	\$
	1	2	3	4	5
	j				

TABLEAU 7.4 – Possibles rotations du mot *page*

$f(4, 1)$	ϵ	a	g	e	\$ ₁ →	ϵ	\$ ₂ ←
	p	ϵ	g	e	\$ ₂ →	ϵ	\$ ₃ ←
	p	a	ϵ	e	\$ ₃ →	ϵ	\$ ₄ ←
	p	a	g	ϵ	\$ ₄ →	ϵ	\$ ₅ ←
$f(4, 2)$	ϵ	a	g	e	\$ ₁ →	ϵ	\$ ₂ ←
	ϵ	ϵ	g	e	\$ ₂ →	ϵ	\$ ₃ ←
	ϵ	a	ϵ	e	\$ ₃ →	ϵ	\$ ₄ ←
	ϵ	a	g	ϵ	\$ ₄ →	ϵ	\$ ₅ ←
	p	ϵ	g	e	\$ ₂ →	ϵ	\$ ₃ ←
	p	ϵ	ϵ	e	\$ ₃ →	ϵ	\$ ₄ ←
	p	ϵ	g	ϵ	\$ ₄ →	ϵ	\$ ₅ ←
	p	a	ϵ	e	\$ ₃ →	ϵ	\$ ₄ ←
p	a	ϵ	ϵ	\$ ₄ →	ϵ	\$ ₅ ←	
$f(4, 3)$	ϵ	a	g	e	\$ ₁ →	ϵ	\$ ₂ ←
	ϵ	ϵ	g	e	\$ ₂ →	ϵ	\$ ₃ ←
	ϵ	ϵ	ϵ	e	\$ ₃ →	ϵ	\$ ₄ ←
	ϵ	ϵ	g	ϵ	\$ ₄ →	ϵ	\$ ₅ ←
	ϵ	a	ϵ	e	\$ ₃ →	ϵ	\$ ₄ ←
	ϵ	a	ϵ	ϵ	\$ ₄ →	ϵ	\$ ₅ ←
	ϵ	a	g	ϵ	\$ ₄ →	ϵ	\$ ₅ ←
	p	ϵ	g	e	\$ ₂ →	ϵ	\$ ₃ ←
	p	ϵ	ϵ	e	\$ ₃ →	ϵ	\$ ₄ ←
	p	ϵ	ϵ	ϵ	\$ ₄ →	ϵ	\$ ₅ ←
	p	ϵ	g	ϵ	\$ ₄ →	ϵ	\$ ₅ ←
	p	a	ϵ	e	\$ ₃ →	ϵ	\$ ₄ ←
p	a	ϵ	ϵ	\$ ₄ →	ϵ	\$ ₅ ←	
p	a	g	ϵ	\$ ₄ →	ϵ	\$ ₅ ←	

TABLEAU 7.5 – Génération des k -suppressions du mot *page*

7.4 Tolérance au bruit

Nous avons exposé qu'une des principales limitations des approches fondées sur les grammaires locales découle de la difficulté de concevoir des grammaires complètes. En d'autres termes, il est difficile d'augmenter la couverture d'une grammaire locale pour capturer davantage de combinaisons de motifs ou d'accroître les possibilités de traitement de variations attendues et inattendues dans un texte. Au-delà de cela, les grammaires locales ne disposent pas de mécanismes leur permettant accéder à des connaissances exogènes, c'est-à-dire des informations extraites de ressources externes au cours de l'analyse et qui peuvent s'avérer utiles pour normaliser, transformer, enrichir ou valider les motifs reconnus.

Cette section concerne l'extraction d'information dans des textes bruités. L'approche exposée mobilise le développement de trois sortes de ressources :

- Des dictionnaires électroniques, tel que des dictionnaires d'anthroponymes et de toponymes, permettant des recherches approximatives (cf. section 7.3).
- Des fonctions étendues pour la manipulation des chaînes de caractères, de variables et, en général, pour la mise en place des stratégies de recherche approximative de motifs.
- Des grammaires locales étendues pour l'extraction d'informations dans des textes bruités.

Nous étudions chacune ces ressources en apportant une attention particulière à présenter la façon dont elles sont agencées afin de permettre l'extraction d'information (normalisation, validation et enrichissement d'entités) même dans des textes bruités.

Par rapport au pouvoir d'analyse d'une grammaire locale classique, le fait de construire des grammaires locales étendues dotées de fonctions étendues, nous permet de :

- Prendre en charge l'analyse de cas non traités, par exemple, la faculté de rapprocher un mot inconnu, ou des mots connus qui forment un mot composé inconnu, d'une entrée lexicale disponible dans un dictionnaire ou dans n'importe quelle autre type de ressource (fichier texte, base de données, etc.)
- Étendre les capacités pour convertir une phrase en une autre, par exemple, au-delà des opérations transformationnelles traitant le temps, l'aspect, le mode ou la voix, une phrase comme *Jean avale une pomme* peut être transformée en *Jean mange un fruit*, avec une grammaire qui décrit la structure syntaxique et qui fait à la fois appel à une ressource externe, capable de rapprocher le mot *avale* à *mange* et celui de *pomme* à *fruit*.

C'est dans le principe de l'ingénierie des grammaires que s'inscrit notre approche. D'une part, nous utilisons une approche classique pour reconnaître les séquences correctes et plus fréquentes :

- Premièrement, nous cherchons les unités élémentaires d'un sous-langage, par exemple, lorsque nous étudions le problème de géolocalisation d'une adresse, nous considérons les codes postaux, les noms des villes et des pays ; ensuite, nous les répertorions soit sous la forme de dictionnaires électroniques, soit sur des listes de mots ou encore en identifiant des masques lexicaux capables de les repérer.
- Deuxièmement, nous réalisons une description formelle des contraintes morphosyntaxiques d'une phrase ou segment de phrase sous la forme de grammaires locales. Dans l'exemple d'une adresse postale, nous décrivons les enchaînements les plus fréquents des unités élémentaires : $\langle \text{Ville} \rangle$, $\langle \text{Pays} \rangle$ ¹ ; $\langle \text{CodePostal} \rangle$ $\langle \text{Ville} \rangle$, $\langle \text{Pays} \rangle$ ² ; $\langle \text{CodePostal} \rangle$ $\langle \text{Ville} \rangle$ $\langle \text{NB} \rangle$ ³ ; $\langle \text{Ville} \rangle$ $\langle \text{CodePostal} \rangle$ ⁴, etc.
- Finalement, une fois la description linguistique réalisée, les dictionnaires et grammaires sont transformés en automates finis qui peuvent être appliqués de façon efficiente à un texte afin d'extraire des informations.

D'autre part, nous implémentons et faisons appel à partir des graphes à des fonctions étendues.

7.4.1 Grammaires locales et tolérance au bruit

Concernant la reconnaissance de motifs, tels que des dates, dans des documents bruités à l'aide des grammaires locales, la littérature disponible n'est pas exhaustive. Citons par exemple, le travail de [Sagot et Gábor \(2014\)](#) pour la détection et correction automatique d'entités nommées dans des corpus OCRisés.

Dans son étude, Sagot propose une architecture d'identification et de correction d'erreurs dans des entités nommées à l'aide de SXPIPE ([Sagot et Boullier, 2008](#)). Le cœur de la démarche consiste à construire des grammaires locales SXPIPE⁵. Ces grammaires, appliquées en cascade, reconnaissent des dates, des adresses et des formules chimiques sous une forme standard ainsi que sous une forme légèrement bruitée. Par la

1. *Vitry Sur Seine, France*

2. *75123 Uppsala, Sweden*

3. *13385 Marseille 05*

4. *London WC1E 6BT*

5. Les grammaires locales de SXPIPE ne sont pas construites à l'aide d'une interface graphique, en revanche, il est possible d'utiliser un module nommé UNITEX2SXPIPE pour convertir un graphe d'Unitex version 2.0 en une grammaire dag2dag ([Sagot et Boullier, 2008](#)). Les grammaires dag2dag sont exprimées dans un langage proche de la forme de Backus-Naur (BNF) et analysées à l'aide du système SYNTAX ([Boullier et Deschamp \(1991\)](#)).

suite, des règles de correction sont appliquées ¹ pour obtenir, si nécessaire, leur version normalisée et corrigée.

Les résultats d'évaluation sur des corpus appartenant aux domaines de la littérature, la jurisprudence et les brevets, montrent un bon rappel et une haute précision en correction, ce qui conduit les auteurs à déduire l'adéquation de l'architecture proposée pour une tâche de reconnaissance et de correction de corpus OCRisés. L'étude conclut en ouvrant deux perspectives, la première vers l'intégration des grammaires locales SXPIPE avec des modèles d'erreurs et des modèles statistiques de langage ², la seconde en permettant aux grammaires locales SXPIPE de proposer plus d'une correction et laisser le modèle d'erreurs choisir la plus pertinente.

Concernant ces deux perspectives de travaux futurs, nous verrons comment une grammaire locale étendue permet de les mettre en place sous une forme intuitive. Ceci sans avoir recours à un couplage en amont des modèles statistiques à partir des sorties de la grammaire comme envisagé par l'étude précitée. En effet, une approche basée sur des grammaires locales étendues peut alors faire appel à un éventail de techniques, dont celles fondées sur des méthodes statistiques, qui sont sollicitées au fur et à mesure que l'analyse se déroule et pas simplement utilisées à l'issue de la reconnaissance.

7.4.2 Reconnaissance tolérante au bruit à l'aide des grammaires étendues

Si la construction des grammaires locales peut s'appuyer sur la même technique utilisée pour reconnaître des langages contextuels (cf. section 3.5.8), quelle est la limite des grammaires locales classiques pour tolérer du bruit, par exemple une faute lexicale inattendue ? Si ces grammaires locales sont capables de définir un langage non contextuel, la première partie de l'approche est accomplie, il suffit donc de définir la façon d'exclure les séquences qui n'appartiennent pas au langage que nous souhaitons décrire pour y parvenir. Cependant, c'est dans la définition de l'opération à réaliser pour exclure ces séquences que les grammaires locales classiques trouvent leur limite.

Dans le cas pratique, pour le problème qui nous concerne, définir l'opération à réaliser pour exclure les séquences revient à définir premièrement la notion de bruit dans la reconnaissance, sujet lié à ce que nous cherchons à reconnaître. Deuxièmement, il faut spécifier les opérations qui permettent de distinguer les séquences bruitées proches de celles recherchées de séquences non désirées. Enfin, il est indispensable de définir comment incorporer ces opérations dans l'algorithme d'analyse syntaxique et comment les utiliser lors la conception des grammaires locales.

Malheureusement, cette stratégie devient rapidement problématique. D'une part il

1. L'identification de règles de correction est effectuée par exploration manuelle d'un corpus d'écarts. Par exemple pour la reconnaissance des nombres, les règles construites par inspection pour faire correspondre un caractère à un chiffre peuvent avoir la forme : $0 \leftarrow \{o, \circ\}$, $1 \leftarrow \{I, l, i\}$... $9 \leftarrow \{g\}$.

2. L'approche envisagée pour ce faire est d'utiliser l'architecture exposée comme un pré-traitement qui viendra en amont du composant reposant sur les modèles statistiques

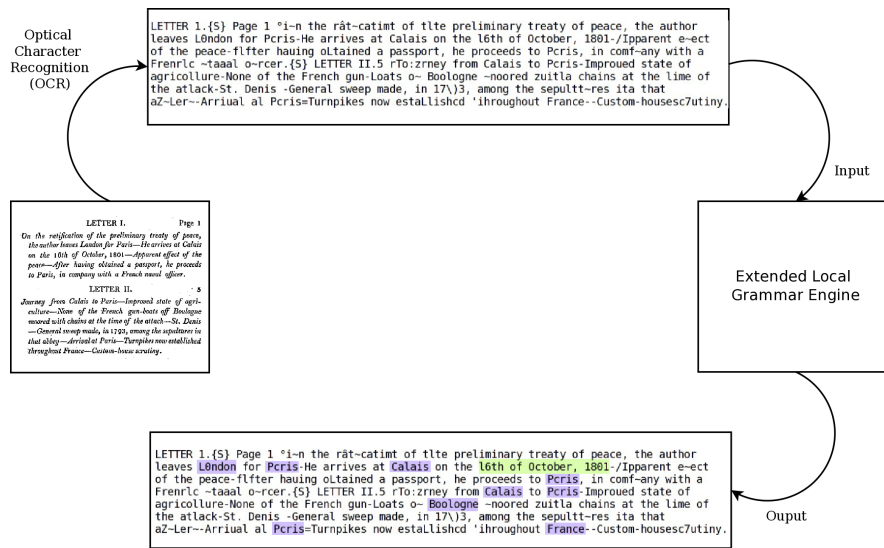


FIGURE 7.2 – Reconnaissance d’entités nommées à l’aide des grammaires locales étendues

n’est pas possible de définir à l’avance le type du bruit à traiter, ce qui permettrait d’implémenter l’opération de filtrage la plus adéquate. D’autre part, même avec quelques opérations standards, elles seront très restreintes en l’absence de mécanismes permettant de :

- paramétrer, modifier et par conséquent adapter, corriger ou améliorer l’implémentation de l’opération,
- accéder aux informations au-delà des données fournies par l’analyse syntaxique courante,

Cela veut dire qu’au niveau de l’implémentation, un couplage fort entre l’algorithme d’analyse syntaxique et les opérations sur les grammaires, implique également d’allourdir la taille de l’analyseur et sa maintenance, tâche toujours sujette aux erreurs. Même en négligeant ces points, l’analyseur résultant, sera soumis à un ensemble d’opérations prédéfinies à l’avance, ni modifiables ni extensibles.

La stratégie qui peut être déployée à l’aide des grammaires locales étendues est abordée différemment, les opérations sont des fonctions étendues définies hors de l’algorithme d’analyse syntaxique et ne sont pas connues à l’avance. Cette adaptabilité permet de :

- définir librement la notion de bruit dans la reconnaissance, pouvant prendre en compte, par exemple, des aspects dérivés de la morphologie, la syntaxe, la sémantique ou la pragmatique ;
- implémenter la fonction la plus adaptée pour exclure les séquences qui ne sont pas considérés comme bien formées.

En outre, comme vue dans le chapitre 4, une fonction peut recevoir des arguments pour adapter son comportement, être corrigée, améliorée et, en particulier, utiliser des connaissances issues à partir de 3 sources d'information différentes¹ :

1. **endogènes** : des informations locales issues de l'analyse de la séquence courante ainsi que des informations globales correspondant à l'analyse des autres séquences qui sont regroupées dans une même analyse,
2. **exogènes** : des informations issues de ressources externes qui ne sont pas directement disponibles à partir de l'analyse, soit celles qui ne sont pas directement ou indirectement intégrées dans l'algorithme d'analyse syntaxique, et
3. **anthropogènes** : des informations qui peuvent être fournies par l'utilisateur au cours de l'analyse.

Par la suite nous présentons des exemples de grammaires locales étendues capables d'extraire de l'information de données bruitées.

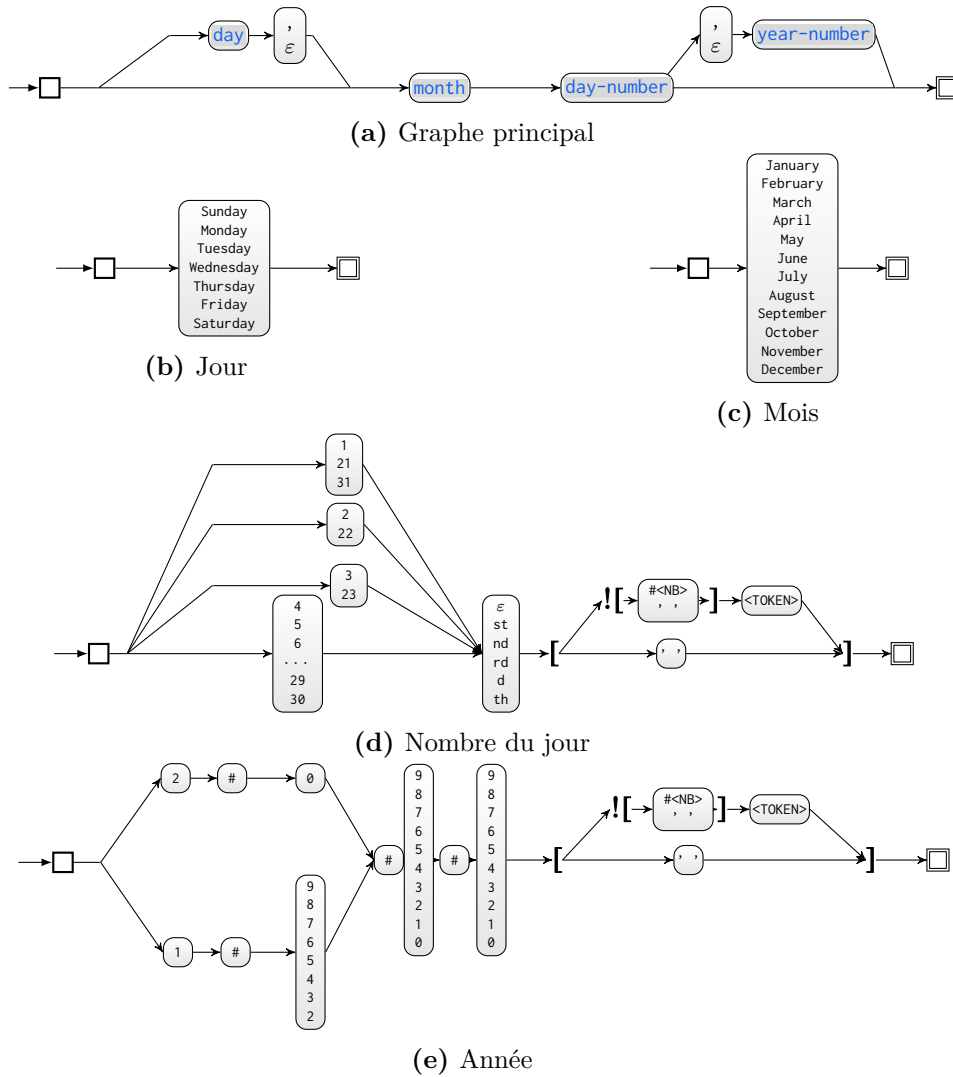
7.4.3 Reconnaissance de dates et tolérance au bruit

Il est bien connu que les expressions de temps, telles que les dates, peuvent être définies aisément à l'aide de grammaires locales. Cependant, la reconnaissance est limitée aux séquences bien formées et parfois peut tolérer certaines séquences mal formées connues à l'avance.

Considérons le graphe 7.4 capable de reconnaître certaines dates en anglais. Ce graphe est composé de 4 sous-graphes en charge d'identifier respectivement les jours de la semaine (7.4b), les mois (7.4c), les nombres du jour entre 1 et 31 (7.4d), ainsi que les années entre 1200 et 2099 (7.4e).

Aussi bien le nombre du jour comme l'année sont optionnels. De plus, une virgule peut être présente ou pas après le jour ou avant l'année. En général, les dates reconnues auront une des formes présentées dans le tableau 7.6.

1. Ces trois sources correspondent aux processus d'immersion documentaire développés largement dans Andreani (2011)



Graphe 7.4 – Reconnaissance des dates en anglais

TYPE DU MOTIF	EXEMPLE
$\langle \text{month} \rangle \langle \text{day-number} \rangle$	January 13th
$\langle \text{month} \rangle \langle \text{day-number} \rangle \langle \text{year-number} \rangle$	December 18 1987
$\langle \text{month} \rangle \langle \text{day-number} \rangle, \langle \text{year-number} \rangle$	April 12, 2016
$\langle \text{day} \rangle \langle \text{month} \rangle \langle \text{day-number} \rangle$	Sunday September 17
$\langle \text{day} \rangle \langle \text{month} \rangle \langle \text{day-number} \rangle \langle \text{year-number} \rangle$	Thursday March 15 2012
$\langle \text{day} \rangle \langle \text{month} \rangle \langle \text{day-number} \rangle, \langle \text{year-number} \rangle$	Wednesday August 29, 1792
$\langle \text{day} \rangle, \langle \text{month} \rangle \langle \text{day-number} \rangle$	Friday, August 3rd
$\langle \text{day} \rangle, \langle \text{month} \rangle \langle \text{day-number} \rangle \langle \text{year-number} \rangle$	Saturday, May 12th 2001
$\langle \text{day} \rangle, \langle \text{month} \rangle \langle \text{day-number} \rangle, \langle \text{year-number} \rangle$	Tuesday, October 11, 1492

TABLEAU 7.6 – Types de dates reconnues par le graphe 7.4

Il est évident que face à un texte avec des séquences bien formées, c'est-à-dire, correspondant aux entrées attendues, la précision de la reconnaissance du graphe 7.4 sera haute¹. Cependant, quelle sera la performance lorsque l'analyse se déroule sur des entrées bruitées ? Par exemple, quand l'analyse s'effectue sur des corpus qui ont été constitués en transformant automatiquement des images en texte grâce à un logiciel de reconnaissance optique de caractères (OCR). Considérons les dates présentées dans le tableau 7.7 provenant d'un corpus océrisé².

IMAGE	RÉSULTAT OCR
<i>December 25, 1801</i>	Decém1er 25, 1801
<i>July 1796,</i>	July] 7g6
<i>January 10, 1802.</i>	JanuarJ 10, 1802
<i>March 8, 1802.</i>	Marcia 8., 18.02
<i>January 18, 1802.</i>	.Ta iuar y 18, 180 2
<i>April 1771.</i>	April]771
<i>February 2, 1802.</i>	Fe,Lr:cary 2 '1802
<i>September 1771.</i>	September J 771
<i>February 28, 1802.</i>	Februar2j 28, 1802

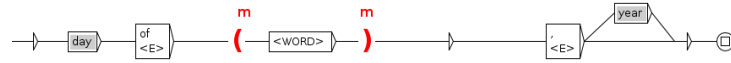
TABLEAU 7.7 – Exemples de dates provenant d'un corpus océrisé

Le résultat OCR des exemples des dates listées dans le tableau 7.7 n'est pas conforme aux sous-séquences décrites par le graphe 7.4. D'une part certains noms de mois contenant du bruit ne sont pas répertoriés, à l'instar de « *JanuarJ* » ou « *Fe,Lr:cary* ». D'autre part les années ne sont plus formées par des nombres mais contiennent d'autres caractères, comme dans « *] 7g6* » ou « *180 2* ». À partir de cet exemple de base, la question est de savoir s'il est possible de concevoir une grammaire locale étendue capable d'identifier des dates bien formées, ainsi que des dates comportant du bruit. Nous verrons par la suite comment l'utilisation des grammaires locales étendues peut aider à obtenir une réponse affirmative.

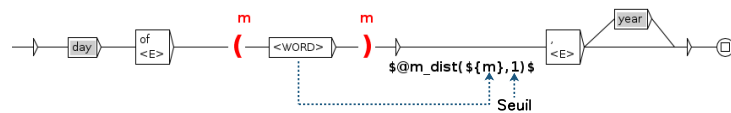
Approche utilisant une grammaire locale étendue Nous cherchons à vérifier, à l'aide de la grammaire locale étendue présentée à la figure figure 7.3, si les entrées suivantes sont de dates :

1. Notons que dans UNITEX les chiffres son traités comme des tokens indépendants. En conséquence, dans le graphe 7.4, le sous-graphe (e) *Année* peut reconnaître par exemple l'année 2017 mais aussi le chiffre 12017. Surmonter ce problème à l'aide d'une grammaire locale étendue devient facile, il suffit de vérifier, à l'aide d'une fonction étendue, que le chiffre est composé par 4 numéros.

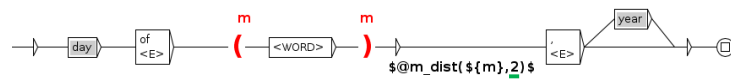
2. *Paris as it was and as it is, or a Sketch of the French capital illustrative of the effects of the Revolution*. Bibliothèque nationale de France. <http://gallica.bnf.fr/ark:/12148/bpt6k102153w>



(a) Dates : reconnaître un langage plus grand



(b) Dates : vérifier le mois à l'aide d'une fonction étendue, seuil 1



(c) Dates : vérifier le mois à l'aide d'une fonction étendue, seuil 2

FIGURE 7.3 – Dates : grammaire locale étendue

- 28th of January .
- 5 of Nlarch 1792
- 8th of Nivôse
- 3 OctUber
- 3 of Brumaire
- 23d of Sceptember
- 11 CQuI1trppen

Évaluation : Nous utilisons un corpus océrisé de la BnF¹, comportant ~ 128.000 mots et 100 dates. Nous comparons la performance de la grammaire étendue (avec les fonctions étendues activées et désactivées) par rapport à trois autres systèmes : une grammaire locale classique, l'extracteur des expressions temporelles de l'outil OPENNLP qui est fondée sur un modèle d'entropie maximale (Ratnaparkhi, 1997) et le module SUTime du Stanford CORENLP (Chang et Manning, 2012).

1. *Paris as it was and as it is, or a Sketch of the French capital illustrative of the effects of the Revolution.* Bibliothèque nationale de France. <http://gallica.bnf.fr/ark:/12148/bpt6k102153w>



FIGURE 7.4 – Exemple de dates reconnues à l'aide d'une grammaire locale étendue

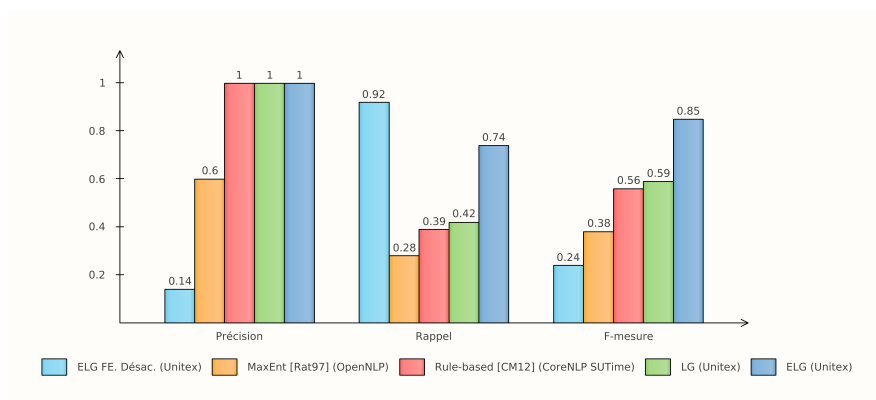


FIGURE 7.5 – Évaluation reconnaissance de dates

7.4.4 Reconnaissance d'anthroponymes et classification du genre des prénoms

Nous cherchons à reconnaître des noms de personnes provenant d'un corpus journalistique ainsi qu'à classifier le genre du prénom en deux catégories : masculin ou féminin :

- i. A World Health Organization official, Margaret Chan, told the meeting there is a great risk.
 - A World Health Organization official, Margaret Chan, told the meeting there is a great risk
 - A World Health Organization official, <START:person gender="f"> Margaret Chan <END>, told the meeting there is a great risk.

- ii. The Swiss star was upset Wednesday by German Tommy Haas in the opening match.
 - The Swiss star was upset Wednesday by German Tommy Haas in the opening match.
 - The Swiss star was upset Wednesday by German <START:person gender="m"> Tommy Haas <END> in the opening match.

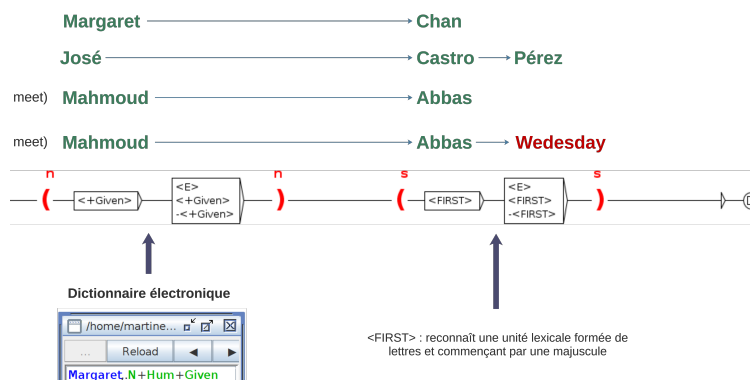


FIGURE 7.6 – Anthroponymes : reconnaître un langage plus grand

Évaluation : Nous utilisons un sous-corpus du GMB(Groningen Meaning Bank)¹, au format CoNLL 2002, comportant ~ 24.000 mots et 215 anthroponymes. Deux évaluations sont réalisées, la première en conservant la casse originale du texte, la deuxième en mettant en majuscule la première lettre de chaque mot. Nous comparons la performance de la grammaire étendue (avec les fonctions étendues activées et

1. Disponible sur <https://www.kaggle.com/abhinavwalia95/entity-annotated-corpus>

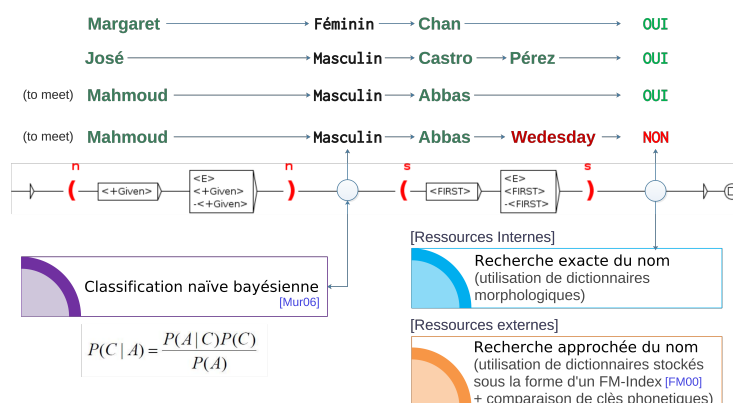


FIGURE 7.7 – Anthroponymes : vérifier les noms à l'aide d'une fonction étendue

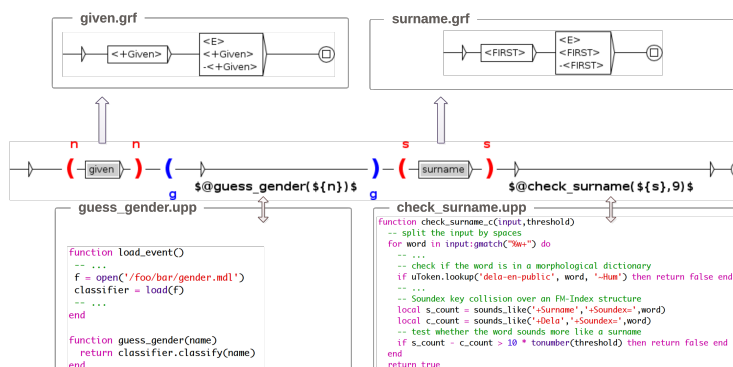


FIGURE 7.8 – Anthroponymes : implémentation de la fonction étendue

désactivées) par rapport à quatre autres systèmes : une grammaire locale classique, une cascade de transducteurs de l'outil CASEN (Maurel et al., 2011) dans sa version ISTEK anglaise, l'extracteur de noms de personnes de l'outil OPENNLP qui est fondé sur un modèle d'entropie maximale (Ratnaparkhi, 1997) et le modèle à 3 classes (personne, organisation lieu) du Stanford CORENLP fondé sur des champs aléatoires conditionnels (Manning et al., 2014).

153 matches

ation Treaty .(S) Iran 's New President <START:person gender="m"> Mahmoud_Ahmadijead <END> Said Tuesday That E
 With The German Firm Bilfinger Berger , <START:person gender="m"> Thomas_Horbach <END> , Said The Gunmen Stoppe
 Somalia 's Interim President Abdullahi <START:person gender="m"> Yusuf_Ahmad <END> .(S) It Was Not Immediately
) Bedfordshire Police Said Tuesday That <START:person gender="m"> Omar_Khayam <END> Was Arrested In Bedford For
 e Officials Say Prominent Tribal Leader <START:person gender="m"> Malik_Faridullah_Khan <END> Was Traveling In
 ay When His Vehicle Was Ambushed In The <START:person gender="f"> Kani_Wam <END> Area .(S) His Driver And A Tri
 t Ready To Make The Nuclear Scientist , <START:person gender="m"> Abdul_Qadeer_Khan <END> , Available For Direc
 Summit On September 8 And 9 .(S) Voa 's <START:person gender="f"> Nancy-Amelia_Collins <END> Reports From Sydne
 is To Do The Same .(S) Foreign Minister <START:person gender="m"> Mustafa_Osman_Ismail <END> Says Sudanese Troo
 With Pistols Attacked The Nun , Sister <START:person gender="f"> Leonella_Sgorbati <END> , After She Finished
 itdraw .(S) Top Palestinian Negotiator <START:person gender="m"> Ahmed_Qursia <END> Says Israeli And Palestini
 he Decision .(S) Israeli Prime Minister <START:person gender="m"> Ehud_Olmert <END> And Palestinian President M
 r Ehud Olmert And Palestinian President <START:person gender="m"> Mahmoud_Abbas <END> Resumed U.s.-Brokered Pea
 hing A Peace Deal Before U.s. President <START:person gender="m"> George_Bush <END> Leaves Office Early Next Ye
 akistan 's Foreign Ministry Spokeswoman <START:person gender="m"> Tasnim_Aslam <END> Declined To Say Whether A
 thier And More Normal Lives .(S) Voa 's <START:person gender="m"> June_Soh <END> Found Camps That Provide Child
 ination Attempt Against Cuban President <START:person gender="m"> Fidel_Castro <END> .(S) The Countries Agreed
 ing Friday Between Panamanian President <START:person gender="m"> Martin_Torrijos <END> And Cuban Vice Presiden
 artin Torrijos And Cuban Vice President <START:person gender="m"> Carlos_Lage <END> .(S) The Meeting Took Place
 gust , Hours After Panamanian President <START:person gender="f"> Mireya_Moscoco <END> , In Her Final Days In 0

FIGURE 7.9 – Exemple de noms reconnus à l'aide d'une grammaire locale étendue

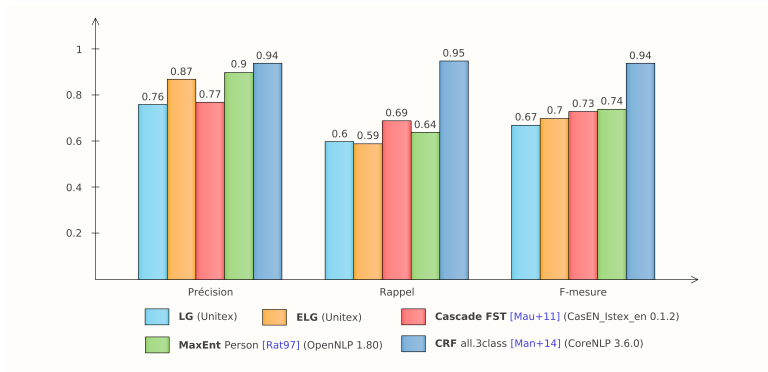


FIGURE 7.10 – Évaluation I reconnaissance d'anthroponymes, texte original

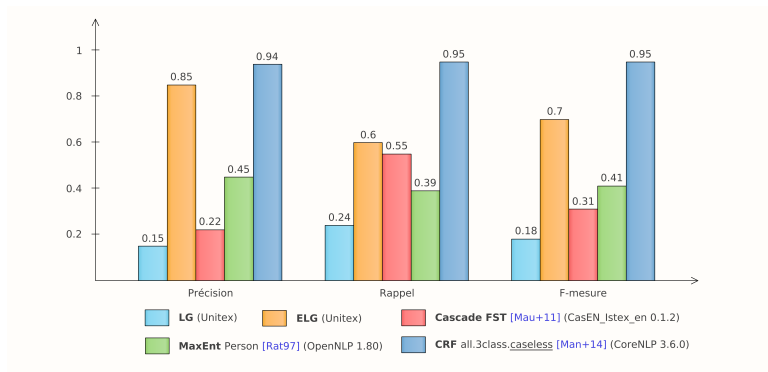


FIGURE 7.11 – Évaluation II reconnaissance d'anthroponymes, changement de casse

7.4.5 Reconnaissance de toponymes et géolocalisation des adresses d'organisations

Nous cherchons à géolocaliser des adresses d'organisations provenant de notices bibliographiques (*Web of Knowledge*) :

- i. GlaxoSmithKline Biol, 1330 Rixensart, Belgium
 - GlaxoSmithKline Biol, 1330 Rixensart, Belgium
 - GlaxoSmithKline Biol, 1330 Rixensart, Belgium
 - GlaxoSmithKline Biol, 1330 Rixensart, Belgium, 4.541692, 50.709794

- ii. Bayer Anim Hlth GmbH, Leverkusen, Germany
 - Bayer Anim Hlth GmbH, Leverkusen, Germany
 - Bayer Anim Hlth GmbH, Leverkusen, Germany
 - Bayer Anim Hlth GmbH, Leverkusen, Germany, 7.029976, 51.048254

- iii. Univ Havre, 76058 Le Havre, France
 - Univ Havre, 76058 Le Havre, France
 - Univ Havre, 76058 Le Havre, France
 - Univ Havre, 76058 Le Havre, France, 0.1077, 49.4938

micro, meso, macro
 meso, macro, longitude, latitude

La grammaire locale étendue utilisée est présentée à la figure 7.12.

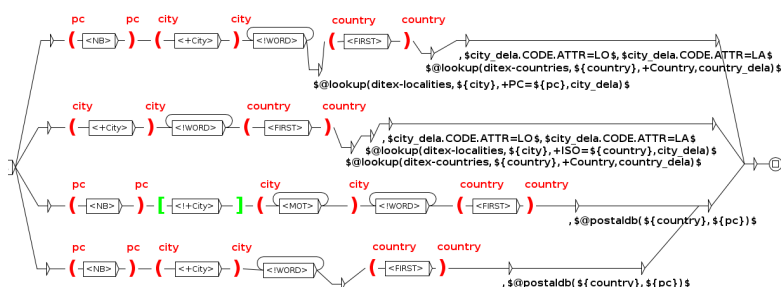


FIGURE 7.12 – Toponymes : grammaire locale étendue

Évaluation : Nous utilisons un corpus de 1000 adresses françaises extraites à partir de notices bibliographiques (chimie et pharmacologie). Nous comparons la performance

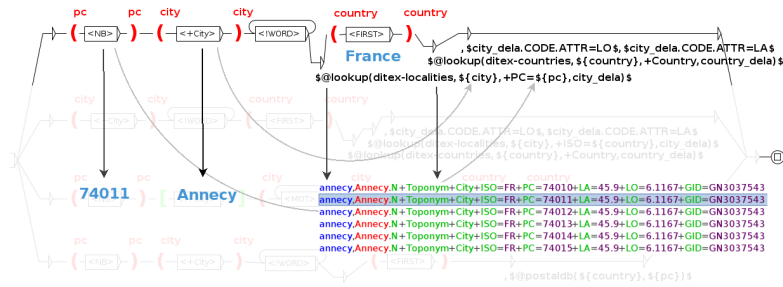


FIGURE 7.13 – Toponymes : analyse du première cas

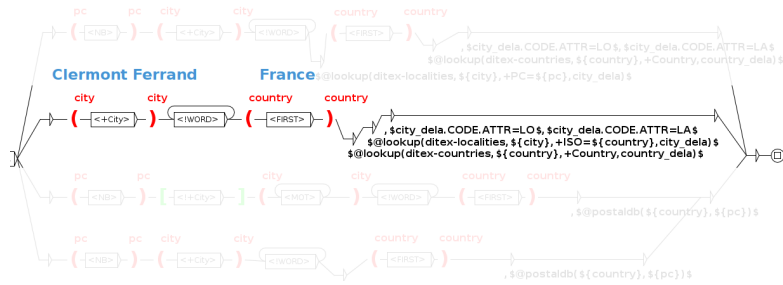


FIGURE 7.14 – Toponymes : analyse du deuxième cas

de la grammaire étendue par rapport à PELIAS¹, un outil de géolocalisation fondé sur l'analyse de l'adresse à l'aide de champs aléatoires conditionnels (LIBPOSTAL) et sa recherche dans une base de données d'environ 500 millions d'adresses.

1. <https://github.com/pelias/pelias>

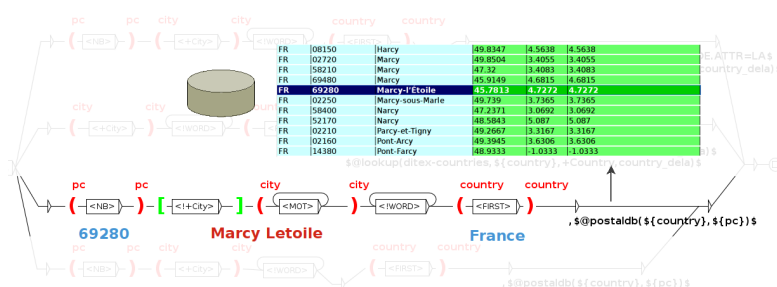


FIGURE 7.15 – Toponymes : analyse du troisième cas

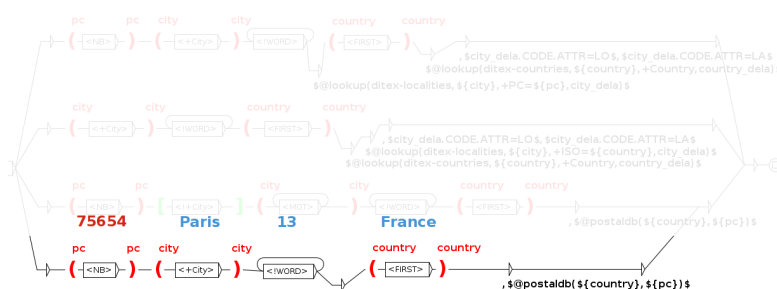


FIGURE 7.16 – Toponymes : analyse du quatrième cas

1014 matches

```

u, Angers, France(S) Novartis Pharma SAS, Creteil, France,2.4667,48.7833(S) CHU Henri Mondor, 9
SAS, Creteil, France(S) CHU Henri Mondor, 94010 Creteil, France,2.4667,48.7833(S) Hop Beaujon,
or, 94010 Creteil, France(S) Hop Beaujon, Clichy, France,2.3895,48.9002(S) Hop St Joseph, Marse
Beaujon, Clichy, France(S) Hop St Joseph, Marseille, France,5.3811,43.2969(S) Hop Tenon, 75970
t Joseph, Marseille, France(S) Hop Tenon, 75970 Paris, France,2.3984,48.8646(S) Hop Cochin, 756
Tenon, 75970 Paris, France(S) Hop Cochin, 75674 Paris, France,2.3264,48.8331(S) St Joseph Hosp,
n, 75674 Paris, France(S) St Joseph Hosp, Marseille, France,5.3811,43.2969(S) CEA, 91191 Gif Sur
St Joseph Hosp, Marseille, France(S) CEA, 91191 Gif Sur Yvette, France,2.1333,48.6833(S) INERIS
St Joseph Hosp, Marseille, France(S) CEA, 91191 Gif Sur Yvette, France,2.1333,48.6833(S) INERIS
, 91191 Gif Sur Yvette, France(S) INERIS, 60150 Verneuil En Halatte, France,2.8608,49.4553(S) A
, 91191 Gif Sur Yvette, France(S) INERIS, 60150 Verneuil En Halatte, France,2.8608,49.4553(S) A
rneuil En Halatte, France(S) Air Liquide, 78350 Jouy En Josas, France,2.1697,48.7591(S) GDF SUE
rneuil En Halatte, France(S) Air Liquide, 78350 Jouy En Josas, France,2.1697,48.7591(S) GDF SUE
e St Denis, France(S) Sanofi Aventis Grp, Paris, France,2.3488,48.85341(S) Univ Paris 11, 91405
ntis Grp, Paris, France(S) Univ Paris 11, 91405 Orsay, France,2.1873,48.6957(S) Sanofi Aventis,
1, 91405 Orsay, France(S) Sanofi Aventis, Paris, France,2.3488,48.85341(S) Boehringer Ingelheim
nce(S) Boehringer Ingelheim GmbH & Co KG, Reims, France,4.0333,49.25(S) Ctr Hosp Reg & Univ Lil
ims, France(S) Ctr Hosp Reg & Univ Lille, 59037 Lille, France,3.0586,50.633(S) Fac Med Lille, 5
le, 59037 Lille, France(S) Fac Med Lille, 59045 Lille, France,3.0586,50.633(S) Childrens Hosp,
e, 59045 Lille, France(S) Childrens Hosp, Lille, France,3.0586,50.633(S) Timone Hosp, Marseille

```

FIGURE 7.17 – Exemple d'adresses géolocalisées à l'aide d'une grammaire locale étendue

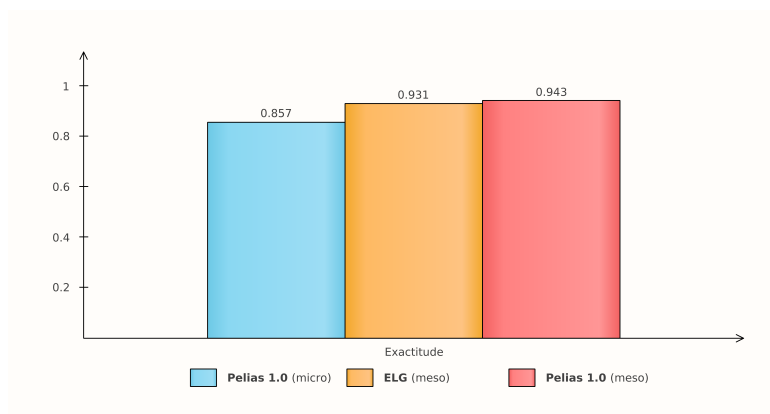


FIGURE 7.18 – Évaluation géolocalisation des adresses

Features	ELG	Pelias 1.0
Niveau du geocodage	Meso	Meso Micro
Adresses postales	✗	✓
Facile à adapter	✓	✗
Fouille de textes	✓	✗

FIGURE 7.19 – Comparaison grammaires locale étendue et Pelias

« Hélas ! les vices de l'homme, si pleins d'horreur qu'on les suppose, contiennent la preuve (quand ce ne serait que leur infinie expansion !) de son goût de l'infini »

Charles Baudelaire (1821–1867)

8

Conclusions

Dans ce travail de thèse nous avons introduit la notion de grammaire locale étendue, nous avons présenté sa mise en œuvre et étudié quelques stratégies pour extraire de l'information à l'aide du formalisme. Nous avons également soulevé le fait que la tâche d'extraction de l'information est un processus intégrant non seulement la reconnaissance d'entités nommées mais aussi la normalisation, l'enrichissement, la validation et la mise en relation des entités identifiées.

La notion de grammaires locales étendues fournit plus de moyens pour confronter des problématiques du traitement automatique du langage. Le formalisme pour mettre en place des fonctions qui ne sont pas connues à l'avance par l'analyseur syntaxique et qui sont évaluées à l'extérieur de la grammaire rend possible, entre autres, l'implémentation de stratégies variées pour traiter des problèmes de façon locale : une approche est mobilisée pour répondre à la question *est-ce que ce motif inconnu est proche du nom d'une personne ?* et une autre approche différente pour répondre à la question *ce nom est-il féminin ou masculin ?*

En outre, le formalisme permet d'enrichir les résultats des méthodes symboliques en complétant celles-ci par d'autres approches computationnelles, comme, entre autres, des calculs mathématiques, des manipulations de chaînes de caractères, l'interrogation de bases de données ou de ressources informatiques externes, ou enfin des analyses statistiques issues des modèles fondées sur l'apprentissage automatique.

Dans une tâche d'extraction d'information, l'adjonction des fonctions étendues ainsi que la capacité de réagir à des événements comme celui de lecture d'un symbole d'entrée, rend possible la recherche approximative de motifs et permet de tolérer le bruit textuel produit par exemple par les mots ou les termes inconnus des ressources. Les

grammaires étendues permettent d'augmenter de ce fait leur capacité de reconnaissance en présence de variations inattendues, telles que celles dans l'écriture des motifs, qui ne sont pas complètement prises en charge par les grammaires locales classiques.

Par ailleurs, les grammaires locales étendues peuvent être utilisées pour d'autres tâches liées à l'extraction d'information telles que l'anonymisation automatique des documents, la création de systèmes de questions-réponses, l'analyse des sentiments, la construction de systèmes de correction grammaticale et d'aide à l'écriture, etc.

Cependant, tous ces aspects doivent être pris en compte en tant que potentialités, en effet, contrairement au cas des approches purement statistiques, les performances d'une approche fondée sur la construction de grammaires étendues dépendront, comme dans le cas des grammaires locales, du développeur linguistique. Dans ce sens, il est pertinent de se questionner sur la démarche à suivre afin d'aider les développeurs de grammaires à tirer le meilleur parti du formalisme.

Le travail réalisé ouvre des perspectives relatives à l'analyse syntaxique des grammaires locales étendues. En effet, dans la mise en œuvre nous avons adapté l'analyseur syntaxique d'UNITEX, du type descendant (*top-down*) en profondeur d'abord (*depth-first*). On pourrait cependant envisager d'utiliser un algorithme de la famille des analyseurs Earley.

D'autres perspectives visent le stockage efficace des grammaires locales étendues : est-il possible de créer un format binaire et unifié constitué par la description de l'automate et les définitions des fonctions à exécuter ? En rapport à l'ergonomie, est-il possible de fournir un moyen pour déboguer les instructions d'une fonction étendue comme partie de la tâche permettant de déboguer la grammaire ?

Finalement, il est nécessaire de remarquer que des limitations hérités des grammaires locales sont aussi présentes dans les grammaires locales étendues, par exemple elles sont aussi complexes à maintenir et demandent un grand investissement pour être développées, la question l'assistance à leur construction reste donc à l'ordre du jour. En outre, de nouvelles limitations ont émergées, par exemple, pour l'instant, il n'y a aucun mécanisme pour interdire à une fonction étendue de boucler indéfiniment, intentionnellement ou pas, et bloquer le processus d'analyse.



Notions préliminaires

Dans cette annexe, nous présentons quelques notions préliminaires. Nous abordons les définitions d'ensembles et de suites, de fonctions et relations, de graphes et arbres, ainsi que celles des alphabets, mots et langages.

A.1 Ensembles et suites

Définition A.1 (Ensemble). Un **ensemble** est une collection d'*éléments*, aussi appelés *membres*, qui sont représentés comme une unité. Dans un ensemble, l'ordre des éléments n'a pas d'importance et les répétitions sont autorisées. Nous notons les ensembles par des lettres majuscules et ses éléments par des minuscules. La liste des éléments est entourée par des accolades et des virgules sont utilisés pour les séparer. Par exemple, l'ensemble contenant les éléments 9, 21, et 23 est noté :

$$X = \{9, 21, 23\}$$

De même, la **relation d'appartenance** d'un élément x à l'ensemble X est indiqué par $x \in X$ qui se lit « x appartient à X ». La négation de $x \in X$ s'écrit $x \notin X$ et se lit « x n'appartient pas à X ». Pour l'exemple ci-dessus nous avons alors :

$$21 \in \{9, 21, 23\} \text{ et } 3 \notin \{9, 21, 23\}$$

Un ensemble qui ne contient pas d'éléments est appelé **ensemble vide**, noté \emptyset , parfois aussi écrit comme $\{\}$. Un ensemble est dit **fini** s'il contient un nombre fini d'éléments. Lorsque un ensemble contient un nombre **infini** d'éléments, nous utilisons la notation \dots (points de suspension). Par exemple, pour décrire l'ensemble E de tous les nombres pairs non nuls, nous écrivons :

$$E = \{2, 4, 6, 8, \dots\}$$

Les points de suspension (...) peuvent être aussi utilisés afin d'abrégier l'énumération des éléments d'un ensemble fini qui se poursuit de manière évidente. Par exemple, l'ensemble de toutes les lettres minuscules de l'alphabet L pourrait être écrit comme suit :

$$L = \{a, b, \dots, z\}$$

Un ensemble peut être aussi défini en compréhension en spécifiant une propriété caractéristique sur les éléments, noté $\{x \mid \text{propriété caractéristique de } x\}$, où \mid se lit « *tel que* ». Les ensembles E et L des exemples précédents peuvent alors être définis comme :

$$\begin{aligned} E &= \{e \in \mathbb{N} \mid e > 0, e \text{ est pair}\} \\ L &= \{l \mid l \text{ est une lettre minuscules de l'alphabet}\} \end{aligned}$$

Définition A.2 (Sous-ensemble). Un ensemble A est dit **sous-ensemble** de B , noté $A \subseteq B$, lit « *A est inclus dans B* », si tout élément de l'ensemble A est élément de l'ensemble B ou si $A = \emptyset$. Par exemple :

$$\{9, 21, 23\} \subseteq \{9, 23, 9, 21\}$$

Lorsque A est inclus dans B mais B contient des éléments qui ne sont pas dans A , l'ensemble A est dit **sous-ensemble propre** de B . Il est noté $A \subsetneq B$ et se lit « *A est strictement inclus dans B* ». Par exemple :

$$\{e, f, g\} \subsetneq \{a, b, \dots, z\}$$

Définition A.3 (Cardinalité d'un ensemble fini). La **cardinalité** d'un ensemble fini X , notée $|X|$, est le nombre d'éléments de l'ensemble. Par exemple, le cardinal de l'ensemble $X = \{9, 21, 23\}$ est 3, soit $|X| = 3$. De même, le cardinal de l'ensemble vide est 0, soit $|\emptyset| = 0$.

Définition A.4 (Opérations entre ensembles). Étant donné deux ensembles A et B , les opérations les plus courantes entre eux sont l'union, l'intersection et la différence.

L'**union** de deux ensembles A et B , notée $A \cup B$, est l'ensemble obtenu en combinant tous les éléments de A et tous les éléments de B en un seul ensemble :

$$A \cup B = \{x \mid x \in A \text{ ou } x \in B\}$$

L'**intersection** de deux ensembles A et B , notée $A \cap B$, est l'ensemble des éléments qui sont à la fois dans A et B :

$$A \cap B = \{x \mid x \in A \text{ et } x \in B\}$$

En particulier si l'intersection des deux ensembles est égal à l'ensemble vide, formellement $A \cap B = \emptyset$, alors A et B sont dits **disjoints**, noté $A \amalg B$.

La **différence** de deux ensembles A et B , notée $A - B$, est l'ensemble des éléments qui sont dans A et pas dans B :

$$A - B = \{x \mid x \in A \text{ et } x \notin B\}$$

L'union, l'intersection et la différence des ensembles peuvent être représentées par des **diagrammes de Venn** comme les ensembles de zones colorées en gris, tels que représentés dans la figure A.1.

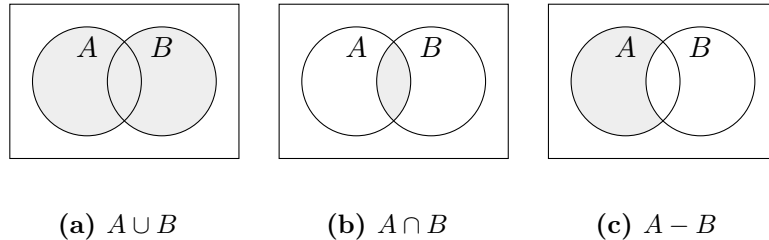


FIGURE A.1 – Union, intersection et différence de deux ensembles

Étant donné un ensemble de référence U , aussi connu comme **ensemble plein**, qui contient tous les éléments du domaine étudié, il est possible de définir une autre opération basique, le **complément** d'un ensemble A , noté \bar{A} , qui se lit « A barre », lequel est composé des éléments de U qui ne sont pas élément de A :

$$\bar{A} = \{x \mid x \in U \text{ et } x \notin A\}$$

Définition A.5 (Suite). Une **suite** est une collection ordonnée d'*éléments*, aussi appelés *termes*. Comme pour les ensembles, dans une suite les répétitions sont autorisées, cependant l'*ordre des membres a de l'importance*. À l'exception des mots (cf. définition A.26), nous notons les suites entre parenthèses et employons les virgules pour séparer les termes. Par exemple, la suite des termes 0, 1, 1, 2, 3, 5, 8, 13 et 21 est notée comme :

$$(0, 1, 1, 2, 3, 5, 8, 13, 21)$$

Définition A.6 (n -uplet). Une suite constituée par un nombre fini de termes est appelé **uplet**, ou **tuple** en anglais. Si $n \in \mathbb{N}$, alors une suite avec n termes est un n -uplet. Par exemple, la suite (0, 1, 1, 2, 3, 5, 8, 13, 21) est un 9-uplet. Certains uplets de longueurs spécifiques ont des noms représentatifs. Quelques noms d'uplets utilisés dans ce travail sont présentés dans le tableau A1.

N	n -uplet
0	uplet vide
1	singleton ¹
2	paire ordonnée ²
3	triplet
4	quadruplet
5	quintuplet
6	sextuplet
7	septuplet
8	octuplet

TABLEAU A1 – Noms pour des uplets de longueurs spécifiques

Définition A.7 (Ensemble des parties). Les ensembles et suites peuvent constituer les éléments d'autres ensembles et suites. L'**ensemble des parties** d'un ensemble X , noté $\mathcal{P}(X)$, est l'ensemble des sous-ensembles de X . Par exemple, si $X = \{0, 1, 2\}$, alors :

$$\mathcal{P}(X) = \{\emptyset, \{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\}\}$$

Remarquons d'abord que l'ensemble des parties d'un ensemble X n'est jamais vide, en effet, \emptyset et X sont toujours des membres de $\mathcal{P}(X)$. Finalement, si X est un ensemble fini, alors la cardinalité de $\mathcal{P}(X)$ est $|\mathcal{P}(X)| = 2^{|X|}$. Dans l'exemple précédent nous avons $|X| = 3$ et alors $2^{|X|} = 8$.

Définition A.8 (Produit cartésien). Étant donné deux ensembles A et B , leur **produit cartésien**, noté $A \times B$ qui se lit « A croix B », est l'ensemble de tous les 2-uplet dont le premier terme appartient à A et le second à B , formellement :

$$A \times B = \{(a, b) \mid a \in A, b \in B\}$$

Par exemple, si $A = \{p, q\}$ et $B = \{0, 1, 2\}$, alors :

$$A \times B = \{(p, 0), (p, 1), (p, 2), (q, 0), (q, 1), (q, 2)\}$$

Observons que, puisque il s'agit de paires ordonnées, l'ordre dans lequel les termes sont écrits est important. Dans l'exemple précédent, la paire $(p, 2)$ appartient à $A \times B$, tandis que $(2, p)$ n'appartient pas. Remarquons aussi que si $|A|$ et $|B|$ sont finis, alors $|A \times B|$ est égal à $|A| \cdot |B|$ ³. Pour l'exemple précédent nous avons $|A \times B| = |A| \cdot |B| = 2 \cdot 3 = 6$.

3. Ici le symbole \cdot représente la multiplication des nombres entiers

Pour trouver le produit cartésien de trois ensembles ou plus, nous pouvons étendre la définition du produit cartésien de deux ensembles. Nous définissons alors $X_1 \times X_2 \times \cdots \times X_n$ comme l'ensemble de tous les n -uplet dont le premier terme appartient à X_1 , le second à X_2 , ..., et le $n^{\text{ième}}$ à X_n , formellement :

$$X_1 \times X_2 \times \cdots \times X_n = \{(x_1, x_2, \dots, x_n) \mid x_i \in X_i\}$$

Comme pour le cas de deux ensembles, si chaque X_i a une cardinalité finie, alors $|X_1 \times X_2 \times \cdots \times X_n|$ est égal à $|X_1| \cdot |X_2| \cdot \dots \cdot |X_n|$.

A.2 Fonctions et relations

Définition A.9 (Fonction). Une **fonction** est une opération qui reçoit des **entrées**, aussi appelées **variables**, et retourne des **sorties**. Plus précisément, une fonction de X dans Y définit une relation entrée-sortie en permettant de faire correspondre aux entrées ($x \in X$) des sorties ($y \in Y$) selon une règle qui attribue à des éléments de l'ensemble X un *élément unique* de l'ensemble Y . Ceci noté comme :

$$f : X \longrightarrow Y$$

Tandis que le premier ensemble, c'est-à-dire, l'ensemble des entrées possibles de la fonction, est appelé l'**ensemble de départ**. Le deuxième ensemble, c'est-à-dire, l'ensemble des sorties possibles de la fonction, est appelé l'**ensemble d'arrivée**. En outre, si f est une fonction qui reçoit en entrée x et renvoie y , nous pouvons utiliser la notation :

$$f(x) = y$$

Ce qui indique que y est l'**image** de x et que x est un **antécédent** de y . De plus, l'ensemble de toutes les entrées $x \in X$ pour lesquels $f(x)$ existe est appelé ensemble de définition ou **domaine** de la fonction. Notons que le domaine de la fonction est alors un sous-ensemble de l'ensemble de départ. Dans certains cas, le domaine de la fonction est égal à l'ensemble de départ¹.

Considérons par exemple la fonction $y = x^2$, l'ensemble de départ de y est les nombres réels \mathbb{R} . De même, son domaine est aussi \mathbb{R} puisque elle est définie pour tout

1. Lorsque le domaine de la fonction est égale à l'ensemble de départ, c'est-à-dire, quand f est une fonction dont tout antécédent a exactement une image et une seule, il est possible d'utiliser le terme **application**. Cependant, dans nos travaux, nous employons le terme application comme synonyme de fonction.

nombre $x \in \mathbb{R}$. Observons aussi que pour cette fonction l'ensemble d'arrivée est les nombres réels \mathbb{R} mais, par contre, l'image de f est l'ensemble des nombres réels positifs ou nuls (\mathbb{R}^+).

Si le domaine d'une fonction est égal au produit cartésien de n ensembles, à savoir $X_1 \times X_2 \times \dots \times X_n$, alors l'entrée de la fonction est un n -uplet (x_1, x_2, \dots, x_n) dont chaque x_i est appelé **argument** de f . Si cette fonction, appelée **fonction n-aire**, possède un seul argument, soit $n = 1$, nous utilisons le nom de *fonction unaire*. De même, quand le nombre d'arguments est égal à 2, nous parlons de *fonction binaire* et pour $n = 3$ *fonction ternaire*. Lorsque n est supérieur à 3, nous utilisons des termes comme 4-aire.

Définition A.10 (Comportement asymptotique¹). Dans certains cas, lorsque les ensembles de départ et d'arrivée des fonctions sont des sous-ensembles de \mathbb{N} (l'ensemble des entiers naturels), nous décrivons le comportement des fonctions lorsque ces arguments deviennent très grands. Si $f(n)$ et $g(n)$ sont deux fonctions dans \mathbb{N} , nous utilisons la notation des ordres de magnitude présentée au tableau A2 pour décrire leur comportement au voisinage de l'infini.

NOM NOTATION	DESCRIPTION
	« f est dominée par g »
Grand O(micron) $f(n) = \mathcal{O}(g(n))$	S'il existe une constante positive réelle c telle que $ f(n) \leq c g(n) $ soit : $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
	« f n'est pas négligeable devant g »
Grand Omega $f(n) = \Omega(g(n))$	S'il existe une constante positive réelle c telle que $ f(n) \geq c g(n) $ soit : $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$
	« f et g sont semblables »
Grand Theta $f(n) = \Theta(g(n))$	S'ils existent deux constantes positives réelles c_1 et c_2 telles que $c_1 g(n) \leq f(n) \leq c_2 g(n) $ soit : $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c < \infty$

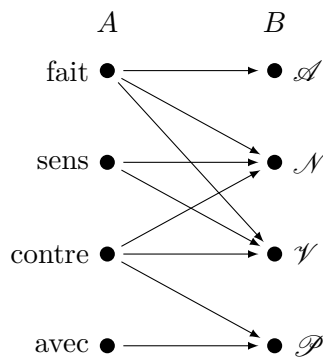
1. Pour une description approfondie du sujet, consultez [Small \(2010\)](#)

NOM NOTATION	DESCRIPTION
	« f est négligeable devant g »
Petit o $f(n) = o(g(n))$	Si pour toute constante positive réelle c $ f(n) \leq c g(n) $ soit : $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$ avec $0 < c < \infty$

TABLEAU A2 – Notations de Bachmann-Landau

Définition A.11 (Relation). Une **relation** R sur X_1, X_2, \dots, X_n est un sous-ensemble du produit cartésien $X_1 \times X_2 \times \dots \times X_n$, soit X^n . Les relations sont plus générales que les fonctions, tandis qu’une fonction attribue aux éléments de l’ensemble de départ un *élément unique* de l’ensemble d’arrivée, une relation peut leur attribuer plusieurs éléments de l’ensemble d’arrivée.

Par exemple, la figure A.2 illustre la relation R entre un ensemble de mots et quelques catégories morphosyntaxiques¹, soit entre un ensemble $A = \{fait, sens, contre, avec\}$ et un ensemble $B = \{\mathcal{A}, \mathcal{N}, \mathcal{V}, \mathcal{P}\}$. Nous pouvons observer que la relation n’est pas une fonction puisque certains éléments de l’ensemble A sont attribués à plusieurs éléments de l’ensemble B .



$$R = \{(fait, \mathcal{A}), (fait, \mathcal{N}), (fait, \mathcal{V}), (sens, \mathcal{N}), (sens, \mathcal{V}), (contre, \mathcal{N}), (contre, \mathcal{V}), (contre, \mathcal{P}), (avec, \mathcal{P})\}$$

FIGURE A.2 – Relation entre un ensemble de mots et des catégories syntaxiques

Une relation comme celle de la figure A.2, c’est-à-dire entre deux ensembles, est nommée **relation binaire**, autrement dit un sous-ensemble ordonné d’*éléments* du produit cartésien $A \times B = \{(a, b) \mid a \in A, b \in B\}$. En outre, si R est une relation binaire et a est en relation avec b , la notation « aRb » est parfois utilisé pour indiquer

1. \mathcal{N} :nom, \mathcal{V} :verbe, \mathcal{A} :adjective, \mathcal{P} :préposition

que la paire ordonnée (a, b) appartient à $A \times B$. Par exemple, pour montrer qu'une paire (a, b) est dans une **relation d'équivalence**, c'est-à-dire que a et b sont similaires par une propriété quelconque, nous écrivons :

$$a \equiv b$$

En général, une relation binaire R est une relation d'équivalence, noté par \equiv , si R satisfait à la fois trois conditions :

1. $x \equiv x$ pour tout élément x , soit R est **réflexive** ;
2. si $x \equiv y$, alors $y \equiv x$, soit R est **symétrique** ; et
3. si $x \equiv y$ et $y \equiv z$, alors $x \equiv z$, soit R est **transitive**.

Définition A.12 (Distance). Une distance sur un ensemble S est une application $\delta : \Sigma^* \times \Sigma^* \rightarrow \mathbb{R}$, si elle satisfait aux conditions suivantes :

1. $\forall x \in X, \forall y \in X, \delta(x, y) = 0 \iff x = y$; (strictement positifs)
2. $\forall x \in X, \forall y \in X, \delta(x, y) = \delta(y, x)$ (symétrique)
3. $\forall x \in X, \forall y \in X, \forall z \in X, \delta(x, z) \leq \delta(x, y) + \delta(y, z)$ (inégalité triangulaire)

Il faut remarquer que les conditions énumérées servent à formaliser certaines intuitions sur une relation spatiale donnée et ne préconisent pas une méthode de calcul de distance en particulier.

A.3 Graphes et arbres

Définition A.13 (Graphe). Un **graphe** \mathcal{G} est un ensemble V de **nœuds** reliés par un ensemble E d'**arcs**, formellement noté $\mathcal{G} = (V, E)$. Chaque arc est défini comme une *paire non ordonnée* (v_i, v_j) de $V \times V$ (l'**ensemble d'arcs**). Un graphe est dit **graphe fini** lorsque l'ensemble de nœuds est un ensemble fini. Il est possible de décrire un graphe en spécifiant V et E ou en créant une représentation graphique où les nœuds sont désignés par des cercles et les arcs sont des lignes qui les relient. Par exemple, étant donné un graphe $\mathcal{G}_1 = (\{0, 1, 2, 3\}, \{(0, 0), (0, 1), (1, 1), (1, 2), (2, 3)\})$, son diagramme associé est celui de la figure A.3a.

Définition A.14 (Graphe étiqueté). Un **graphe étiqueté** est un graphe dont les arcs sont affectés d'étiquettes indiquant des informations, par exemple une valeur numérique, un symbole ou un mot. Plus formellement, $\mathcal{G} = (V, E)$ est un graphe étiqueté s'il existe un ensemble T fini et *non-vide* tel que : $E \rightarrow T$, autrement dit, s'il existe une fonction de E dans T . En outre, étant donné $f(e) = t, t \in T$ est appelé l'**étiquette** de $e \in E$.

Définition A.15 (Sous-graphe). Étant donné un graphe $\mathcal{G} = (V, E)$, un **sous-graphe** $\mathcal{G}_s = (V_s, E_s)$ de \mathcal{G} est un graphe tel que les nœuds du sous-graphe (V_s), sont un sous-ensemble de V , soit $V_s \subseteq V$; et les arcs du sous-graphe, (E_s) sont aussi arcs de \mathcal{G} pour les nœuds correspondants, soit $E_s \subseteq E \cap V_s \times V_s$.

Définition A.16 (Graphe orienté). Un **graphe orienté** \mathcal{G} est un ensemble V de **nœuds** reliés par un ensemble E d'**arcs orientés**. Chaque arc est défini comme une *paire ordonnée* (v_i, v_j) de $V \times V$, où le premier terme (v_i) correspond au **nœud de départ** et le deuxième terme (v_j) au **nœud d'arrivée**. Ainsi, une paire (v_i, v_j) est différente d'une autre paire (v_j, v_i) . Étant donné $\mathcal{G}_2 = (\{0, 1, 2, 3\}, \{(0, 0), (0, 1), (1, 1), (1, 2), (2, 3), (3, 2)\})$, son diagramme associé est celui de la figure A.3b.

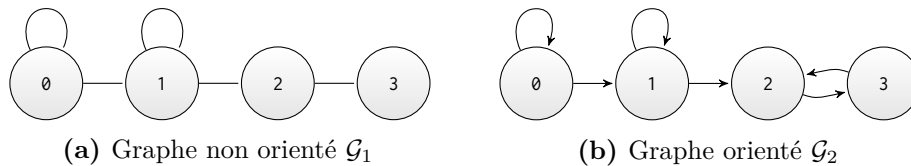


FIGURE A.3 – Diagrammes de graphes à 4 nœuds

N.B. A.17. Dorénavant, sans autre précision, le terme **graphe** est utilisé pour désigner un graphe orienté.

Définition A.18 (Chemin, circuit, boucle d'un graphe). Un **chemin** de \mathcal{G} est une suite de n nœuds (v_0, v_1, \dots, v_n) les reliant. Un **chemin élémentaire** est un chemin sans nœuds répétés. Un chemin de \mathcal{G} est un **circuit** si le nœud de départ est égal au nœud d'arrivée. Un **circuit élémentaire** est un circuit contenant au moins 3 nœuds dont le premier et le dernier n'existent que dans un unique exemplaire. Un arc qui relie un nœud à lui-même, est dit une **boucle**. Observons, par exemple, que dans le graphe \mathcal{G}_2 de la figure A.3a il existe deux arcs $(0, 0)$ et $(1, 1)$ qui sont des boucles.

Définition A.19 (Graphe acyclique). Un graphe orienté est un **graphe acyclique** s'il ne contient aucun circuit.

Définition A.20 (Arbre enraciné). Un arbre enraciné, désormais dénommé **arbre**, est un graphe acyclique qui possède un nœud distingué, appelé **racine**, tel qu'il existe exactement un seul chemin élémentaire à partir de la racine vers tout autre nœud.

Définition A.21 (Nœud parent et nœud fils). Si (v_i, v_j) de $V \times V$ est un arc $e \in E$ d'un arbre, alors v_i est appelée le nœud **parent** et v_j le nœud **fils**. Remarquons que la définition A.20 implique que la racine d'un arbre est le seul élément à ne pas avoir de parent.

Définition A.22 (Feuille). Si (v_i, v_j) de $V \times V$ est un arc $e \in E$ d'un arbre et il n'existe pas un autre arc dont v_j soit parent, alors v_j est appelée une **feuille**. L'ensemble de nœuds ne possédant pas de fils sont nommées alors les **feuilles** de l'arbre.

Définition A.23 (Profondeur d'un nœud). La **profondeur** associée à un nœud $v_i \in V$ est égal au nombre d'arcs dans le chemin partant de la racine au v_i .

Définition A.24 (Hauteur d'un arbre). La **hauteur** d'un arbre est égal à la valeur de la plus grande profondeur parmi toutes les profondeurs des feuilles de l'arbre.

A.4 Alphabets et mots

Définition A.25 (Alphabet). Un **alphabet**, conventionnellement noté Σ , mais aussi représenté par d'autres lettres capitales grecques telles que Γ ou Φ , est un ensemble fini et *non-vide* d'éléments appelés **symboles**. Par exemple :

$\Sigma_1 = \{0, 1\}$, l'alphabet binaire.

$\Sigma_2 = \{a, b, c, \dots, z\}$, l'alphabet des lettres minuscules.

$\Sigma_3 = \{\mathcal{A}, \mathcal{C}, \mathcal{G}, \mathcal{T}\}$, l'alphabet des quatre constituants des gènes.

Définition A.26 (Mot). Un **mot**, conventionnellement noté w , défini sur l'alphabet Σ , est une suite *finie* de symboles de Σ . Par exemple :

$$w_1 = 11111001, w_1 \in \Sigma_1$$

$$w_2 = page, w_2 \in \Sigma_2$$

$$w_3 = \mathcal{T}\mathcal{T}\mathcal{A}\mathcal{C}, w_3 \in \Sigma_3$$

Notons que les symboles d'un mot sont écrits de façon contiguë et ne sont pas séparés par des virgules.

Définition A.27 (Longueur d'un mot). La **longueur d'un mot** w , notée $|w|$, est le nombre de symboles constituant ce mot. Plus strictement, elle est égale au nombre des positions des symboles utilisées pour constituer le mot. Par exemple :

$$|w_1| = 8$$

$$|w_2| = 4$$

$$|w_3| = 4$$

Définition A.28 (Mot vide). Le **mot vide**, noté ε , correspond à une suite sans symboles, soit au mot qui satisfait les relations suivantes :

$$|\varepsilon| = 0$$

$$\varepsilon w = w\varepsilon = w, \text{ pour tout } w.$$

Nous utilisons la notation $\{\varepsilon\}$ pour représenter un ensemble ne contenant que le mot vide. Aussi, étant donné un alphabet Σ , nous symbolisons par la notation Σ_ε l'union entre l'alphabet Σ et un ensemble ne contenant que le mot vide $\{\varepsilon\}$, soit $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.

Définition A.29 (Concaténation des mots). La **concaténation** de deux mots $u = u_1u_2 \cdots u_n$ et $v = v_1v_2 \cdots v_m$, noté uv , est le mot de longueur $n + m$ ¹ obtenu par l'attachement de v à la fin de u , soit :

$$uv = u_1u_2 \cdots u_nv_1v_2 \cdots v_m$$

Pour indiquer la concaténation d'un mot w avec lui-même, c'est-à-dire la concaténation de n copies de w , il est possible d'utiliser la notation de puissance. Par exemple, w^n est la **puissance n-ième du mot** w . En particulier, $w^0 = \varepsilon$ pour tout mot w .

Définition A.30 (Ensemble des mots sur Σ). Si Σ est un alphabet, nous définissons Σ^k comme l'ensemble des mots de longueur k appartenant à Σ . En particulier, nous disons que Σ^* est l'**ensemble des mots sur un alphabet**, définie par $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$, aussi noté :

$$\Sigma^* = \bigcup_{i \geq 0} \Sigma^i$$

Observons que Σ^* est alors l'ensemble des mots que l'on peut construire à partir de Σ . Par exemple si $\Sigma = \{0, 1\}$, on a :

$$\Sigma^* = \{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$$

Étant donnée que $\Sigma^0 = \{\varepsilon\}$, il est convenable d'introduire une notation pour faire référence à l'ensemble des mots sur un alphabet sans inclure le mot vide, soit $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. Pour cela, nous utilisons $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \cdots$, aussi noté :

$$\Sigma^+ = \bigcup_{i \geq 1} \Sigma^i$$

Définition A.31 (Occurrences d'un mot). Si σ est un symbole de Σ alors $|w|_\sigma$ est le **nombre d'occurrences** de σ dans w . Par exemple :

$$\begin{aligned} |11111001|_0 &= 2 \\ |page|_x &= 0 \\ |\mathcal{T}\mathcal{T}\mathcal{A}\mathcal{C}|_{\mathcal{A}} &= 1 \end{aligned}$$

1. Notons que la longueur de uv est égal à $|uv| = |u| + |v| = n + m$.

Définition A.32 (Préfixe, Suffixe, Facteur). Soit un mot $w = w_1w_2 \cdots w_l$ sur Σ . Les mots $\varepsilon, w_1, w_1w_2, \dots, w_1 \cdots w_{l-1}$ et w , sont les **préfixes** de w . Par exemple, si $w = page$, alors l'ensemble de tous les préfixes de w est :

$$\text{Pref}(w) = \{\varepsilon, p, pa, pag, page\}$$

De même, les mots $\varepsilon, w_l, w_{l-1}w_l, \dots, w_2 \cdots w_l, w_1 \cdots w_l$, sont les **suffixes** de w . Par exemple, si $w = page$, alors l'ensemble de tous les suffixes de w est :

$$\text{Suff}(w) = \{\varepsilon, e, ge, age, page\}$$

Aussi, l'ensemble de mots $w_i \cdots w_j$ pour $1 \leq i \leq j \leq l$, sont les **facteurs** de w . Par exemple, si $w = page$, alors l'ensemble de tous les facteurs de w est :

$$\text{Fac}(w) = \{\varepsilon, p, a, g, e, pa, ag, ge, pag, age, page\}$$

Un préfixe de w est dit *propre* lorsque il est différent de ε et de w . De la même façon, un suffixe de w ou un facteur de w est dit propre lorsque il est différent de ε et de w .

Définition A.33 (Sous-mot). Soit un mot $v = v_1v_2 \cdots v_l$ sur Σ . Un **sous-mot** u de v , est un mot obtenu en supprimant des lettres dans certaines des positions, non nécessairement adjacents, en v . Par exemple, si $w = page$, alors l'ensemble de tous les sous-mots de w est :

$$\text{Sub}(w) = \{age, ge, e, \varepsilon, g, ae, a, ag, pge, pe, p, pg, pae, pa, pag\}$$

A.5 Langages

Définition A.34 (Langage). Un **langage** est un ensemble de mots sur un alphabet Σ , c'est-à-dire une partie de Σ^* . Considérons par exemple l'alphabet binaire $\Sigma = \{0, 1\}$, tel que $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$, on peut alors définir les langages suivants :

$$L_1 = \{0, 01, 001, 111\}$$

$$L_2 = \{0^n 1^n \mid n \geq 0\}$$

L_1 est un langage sur Σ et étant donnée qu'il a un *nombre fini* de mots, il s'agit d'un **langage fini**. De même, L_2 est aussi un langage sur Σ , mais étant donnée qu'il a un *nombre infini* de mots, il est dit **langage infini**. Si un langage ne contient aucun

mot, il est dit **langage vide**, noté \emptyset . Par ailleurs, deux langages sont dits **faiblement équivalents**, noté $L_1 = L_2$, s'ils contiennent les mêmes mots.

Remarquons que même si un langage est infini, la contrainte imposée est d'utiliser des alphabets qui sont par définition finies. Observons également que Σ^* , comme \emptyset (le **langage vide**) et $\{\varepsilon\}$ (le langage constitué simplement du mot vide) sont des langages pour n'importe quel alphabet.

Définition A.35 (Union, intersection et différence des langages). Étant donnée deux langages L_1 et L_2 , qui sont par définition des ensembles, les opérations d'union, intersection et différence sont similaires à celles présentées dans la définition A.4 :

$$\begin{aligned} L_1 \cup L_2 &= \{u \in \Sigma^* \mid u \in L_1 \text{ ou } u \in L_2\} \\ L_1 \cap L_2 &= \{u \in \Sigma^* \mid u \in L_1 \text{ et } u \in L_2\} \\ L_1 - L_2 &= \{u \in \Sigma^* \mid u \in L_1 \text{ et } u \notin L_2\} \end{aligned}$$

Définition A.36 (Complément d'un langage). Le **complément d'un langage** L , noté \bar{L} , est défini par rapport à Σ^* comme $\bar{L} = \Sigma^* - L$. Soit l'ensemble défini par :

$$\bar{L} = \{u \in \Sigma^* \mid u \notin L\}$$

Définition A.37 (Concaténation des langages). La **concaténation de deux langages** L_1 et L_2 , noté $L_1 \circ L_2$, est l'ensemble des tous les mots obtenus en concaténant n'importe quel élément u de L_1 avec n'importe quel élément v de L_2 . Précisément :

$$L_1 \circ L_2 = \{uv \in \Sigma^* \mid u \in L_1, v \in L_2\}$$

Par exemple, si $L_1 = \{00, 10\}$ et $L_2 = \{0, 1\}$, nous avons que $L_1 \circ L_2$ est égal à :

$$L_1 \circ L_2 = \left\{ \overbrace{0}^{L_1} \overbrace{0}^{L_2}, \overbrace{0}^{L_1} \overbrace{1}^{L_2}, \overbrace{1}^{L_1} \overbrace{0}^{L_2}, \overbrace{1}^{L_1} \overbrace{1}^{L_2} \right\}$$

Pour indiquer la concaténation d'un langage L avec lui-même, c'est-à-dire la concaténation de n copies de L il est possible d'utiliser la notation de puissance. Par exemple, L^n est la **puissance n-ième du langage** L , soit $L^n = L \circ L \circ \dots \circ L$. En particulier :

$$L^n = \begin{cases} \{\varepsilon\}, & \text{si } n = 0 \\ L^{n-1} \circ L, & \text{autrement} \end{cases}$$

Remarquons que $L^0 = \{\varepsilon\}$ et $L^1 = L^0 \circ L = L$ pour tout langage L .

Définition A.38 (Étoile de Kleene). Étant donnée un langage L , l'**étoile de Kleene**¹, notée L^* , est l'ensemble des tous les mots que l'on peut construire en concaténant un nombre arbitraire et fini de mots de L . Précisément, l'opération est définie par $L^* = L^0 \cup L^1 \cup L^2 \cup \dots$, aussi noté :

$$L^* = \bigcup_{i \geq 0} L^i$$

Observons que puisque $L^0 = \{\varepsilon\}$ alors L^* contient toujours ε et par conséquence $\emptyset^* = \{\varepsilon\}$ n'est pas vide². Afin de représenter un langage L qui ne contient ε que si L le contient, il est possible d'utiliser la notation L^+ définie par $L^+ = L^1 \cup L^2 \cup \dots$, aussi noté :

$$L^+ = \bigcup_{i \geq 1} L^i$$

En particulier $(L^*)^* = L^*$.

Définition A.39 (Langage formel). Un **langage formel** est un langage L pour lequel il existe un mécanisme ou machine abstraite capable d'une part de définir et engendrer tous les mots appartenant à L et, d'autre part, de reconnaître si un mot w appartient ou pas à L . Certaines façons de définir, engendrer ou analyser un langage sont donnés par des :

- **Grammaires formelles** : Engendrent tous les mots appartenant à un langage L .
- **Automates** : Reconnaittent si w appartient ou pas à un langage L .
- **Expressions régulières** : Définissent selon une syntaxe précise le langage engendré par L .

Ces mécanismes sont présentés au chapitre 2.

1. Aussi appelée *fermeture de Kleene*, *opération étoile*, *fermeture itérative*, ou *monoïde libre* sur Σ .
2. En particulier, $\emptyset^* = \{\varepsilon\}^* = \{\varepsilon\}$

Bibliographie

- Abramowicz, W. 2013, *Knowledge-based information retrieval and filtering from the Web*, vol. 746, Springer Science & Business Media.
- Agarwal, N., K. H. Ford et M. Shneider. 2005, « Sentence boundary detection using a maxent classifier », dans *Proceedings of MISC*, p. 1–6.
- Allauzen, C. et M. Riley. 2012, « A pushdown transducer extension for the openfst library », *Implementation and Application of Automata*, p. 66–77.
- Allauzen, C., M. Riley, J. Schalkwyk, W. Skut et M. Mohri. 2007, « Openfst : A general and efficient weighted finite-state transducer library », *Implementation and Application of Automata*, p. 11–23.
- Andor, D., C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov et M. Collins. 2016, « Globally normalized transition-based neural networks », *arXiv preprint arXiv :1603.06042*.
- Andreani, V. 2011, *Immersion dans des documents scientifiques et techniques : unités, modèles théoriques et processus*, thèse de doctorat, Université de Grenoble.
- Aubin, S. et M. Barbier. 2003, « Extraction d'information à partir de documents contractuels », dans *Actes de l'atelier Acquisition, apprentissage et exploitation de connaissances sémantiques pour l'accès au contenu textuel, plate-forme AFIA*, p. 17–28.
- Backurs, A. et P. Indyk. 2014, « Edit distance cannot be computed in strongly sub-quadratic time (unless SETH is false) », *arXiv Prepr. arXiv1412.0348*, vol. 2, n° 1, doi :0.1145/2746539.2746612, p. 0–14, ISSN 07378017.
- Bates, M. 1978, « The theory and practice of augmented transition network grammars », dans *Natural Language Communication with Computers*, Springer-Verlag, London, UK, UK, ISBN 3-540-08911-X, p. 191–259.
- Baudot, J. 1987, *Introduction aux grammaires formelles*, [Montréal] : Librairie de l'Université de Montréal, ISBN 2920286102.
- Bauer, G. 1985, « Namenkunde des deutschen. germanistische lehrbuchsammlung », .
- Berlocher, I., H.-g. Huh, E. Laporte et J.-s. Nam. 2006, « Morphological annotation of Korean with Directly Maintainable Resources », dans *Poster session of 5th International conference on Language Resources and Evaluation (LREC'06)*, Genoa.

-
- Berstel, J. 2013, *Transductions and context-free languages*, Springer-Verlag.
- Blanc, O. 2006, *Algorithmes d'analyse syntaxique par grammaires lexicalisées : optimisation et traitement de l'ambiguïté*, Thesis, Université Paris-Est.
- Blanc, O., M. Constant et E. Laporte. 2006, « Outilex, plate-forme logicielle de traitement de textes écrits », dans *13ème Colloque sur le traitement automatique des langues naturelles (TALN'06)*, *Cahiers du CENTAL*, vol. 2, édité par C. Fairon, , p. 83–92.
- Blanc, O. et A. Dister. 2004, « Automates lexicaux avec structure de traits », dans *8ème Rencontre des étudiants chercheurs en informatique pour le traitement automatique des langues (RECITAL'04)*, p. 1–11.
- Bolinger, D. 1986, *Intonation and Its Parts : Melody in Spoken English*, Stanford University Press.
- Boujelben, I., S. Jamoussi et A. B. Hamadou. 2014, « A hybrid method for extracting relations between arabic named entities », *Journal of King Saud University - Computer and Information Sciences*, vol. 26, n° 4, doi :10.1016/j.jksuci.2014.06.004, p. 425–440, ISSN 1319-1578. Special Issue on Arabic NLP.
- Boullier, P. et P. Deschamp. 1991, « Le systeme syntax : Manuel d'utilisation et de mise en œuvre sous unix », .
- Boytsov, L. 2011, « Indexing methods for approximate dictionary searching : Comparative analysis », *J. Exp. Algorithmics*, vol. 16, doi :10.1145/1963190.1963191, p. 11 :11–11 :191, ISSN 1084-6654.
- Calberg, M. 2003, « Traitement de la morphologie du finnois par transducteurs à nombre fini d'états », *Récital*. Poster.
- Carpenter, B. 2005, « Scaling high-order character language models to gigabytes », dans *Proceedings of the Workshop on Software*, Association for Computational Linguistics, p. 86–99.
- Cavnar, W. B., J. M. Trenkle et al.. 1994, « N-gram-based text categorization », *Ann Arbor MI*, vol. 48113, n° 2, p. 161–175.
- Chang, A. X. et C. D. Manning. 2012, « Sutime : A library for recognizing and normalizing time expressions. », dans *LREC*, vol. 2012, p. 3735–3740.
- Chein, M., A. Gutierrez et M. Leclère. 2015, *Un problème d'identification d'entités nommées dans des bases de données documentaires*, thèse de doctorat, LIRMM.
- Chinchor, N., P. Robinson et E. Brown. 1998, « Hub-4 named entity task definition version 4.8 », *Available by ftp from www.nist.gov/speech/hub4_98*.
- Chinchor, N. A. 1998, « Overview of muc-7/met-2 », cahier de recherche, SCIENCE APPLICATIONS INTERNATIONAL CORP SAN DIEGO CA.

-
- Chrobot, A., B. Courtois, M. H.-M. Carthy, M. Gross et K. Zellagui. 1999, « Dictionnaire électronique DELAC anglais : Noms composés », cahier de recherche 59, LADL, Université Paris 7.
- Church, K. W. 1988, « A stochastic parts program and noun phrase parser for unrestricted text », dans *Proceedings of the second conference on Applied natural language processing*, Association for Computational Linguistics, p. 136–143.
- Clocksin, W. et C. S. Mellish. 2003, *Programming in Prolog*, Springer Science & Business Media, ISBN 9783540110460.
- Collins, M. et Y. Singer. 1999, « Unsupervised models for named entity classification », dans *1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.
- Collobert, R. et J. Weston. 2008, « A unified architecture for natural language processing : Deep neural networks with multitask learning », dans *Proceedings of the 25th International Conference on Machine Learning, ICML '08*, ACM, New York, NY, USA, ISBN 978-1-60558-205-4, p. 160–167, doi :10.1145/1390156.1390177.
- Constant, M. 2003, *Grammaires locales pour l'analyse automatique de textes : méthodes de construction et outils de gestion*, thèse de doctorat, Université Paris-Est.
- Constant, M. 2004, « Vers la construction d'une bibliothèque en-ligne de grammaires linguistiques », *Lexicometrica*, vol. Actes du colloque "L'analyse de données textuelles : De l'enquête aux corpus littéraires", n° Numéro spécial, p. 14.
- Constant, M. 2007, « GraalWeb ou accéder à une bibliothèque décentralisée de grammaires locales », dans *Bases de données lexicales : construction et applications*, édité par N. Jeusun et P. Alain, 1, Observatoire de linguistique Sens-Texte, Canada, p. 79–87.
- Cornuéjols, A. et L. Miclet. 2011, *Apprentissage Artificiel : Concepts Et Algorithmes*, Algorithmes, Eyrolles, ISBN 9782212083019.
- Courtois, B., M. Garrigues, G. Gross, M. Gross, R. Jung, M. Mathieu-Colas, A. Monceaux, A. Poncet-Montange, M. Silberztein et R. Vivés. 1997, « Dictionnaire électronique delac : les mots composés binaires », *Rapport technique*, vol. 56.
- Courtois, B. et M. Silberztein. 1990, « Les dictionnaires électroniques du français. Larousse », *Langue française*, vol. 87, p. 3–9.
- Crochemore, M., G. M. Landau et M. Ziv-Ukelson. 2002, « A sub-quadratic sequence alignment algorithm for unrestricted cost matrices », dans *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, ISBN 089871513X, p. 679–688.
- Crochemore, M., G. M. Landau et M. Ziv-Ukelson. 2003, « A subquadratic sequence alignment algorithm for unrestricted scoring matrices », *SIAM J. Comput.*, vol. 32, doi :10.1137/S0097539702402007, p. 1654–1673, ISSN 0097-5397.

-
- Cruse, D. A. 1986, *Lexical semantics*, Cambridge University Press.
- Cruse, D. A. 1995, « Polysemy and related phenomena from a cognitive linguistic viewpoint », *Saint-Dizier and Viegas*, p. 33–49.
- Cruse, D. A. 2004, *Meaning in Language : An Introduction to Semantics and Pragmatics*, Oxford University Press.
- Daille, B., N. Fourour et E. Morin. 2000, « Catégorisation des noms propres : une étude en corpus », *Cahiers de Grammaire*, vol. 25, n° 25, p. 115–129.
- Darwish, K., A. Abdelali et H. Mubarak. 2014, « Using stem-templates to improve arabic pos and gender/number tagging. », dans *LREC*, p. 2926–2931.
- Davis, M. D., R. Sigal et E. J. Weyuker. 1994, *Computability, Complexity, and Languages (2nd Ed.) : Fundamentals of Theoretical Computer Science*, Academic Press Professional, Inc., San Diego, CA, USA, ISBN 0-12-206382-1.
- Declerck, T. et E. Andre. 2002, « L'indexation conceptuelle de documents multilingues et multimédias », dans *Multilinguisme et traitement de l'information (Traité des sciences et techniques de l'information)*, Lavoisier.
- Dinarelli, M. et S. Rosset. 2012, « Tree-structured named entity recognition on ocr data : Analysis, processing and results », dans *Language Resources Evaluation Conference (LREC)*.
- Dister, A., C. Fairon et al.. 2004, « Extension des ressources lexicales grâce à un corpus dynamique », *Lexicometrica*.
- Donabédian, A., V. Khurshudian et M. Silberztein. 2013, *Formalising Natural Languages with NooJ*, EBSCO ebook academic collection, Cambridge Scholars Publishing, ISBN 9781443850193.
- Ehrmann, M. 2008, *Les entités nommées, de la linguistique au TAL : Statut théorique et méthodes de désambiguïsation*, thèse de doctorat, Université Paris 7 - Denis Diderot.
- Ek, T., C. Kirkegaard, H. Jonsson et P. Nugues. 2011, « Named entity recognition for short text messages », *Procedia-Social and Behavioral Sciences*, vol. 27, p. 178–187.
- Ernst, A., T. Ruf et C. Kueblbeck. 2009, « A modular framework to detect and analyze faces for audience measurement systems », dans *2nd Workshop on Pervasive Advertising at Informatik*, p. 75–87.
- Ezzat, M. 2014, *Acquisition de relations entre entités nommées à partir de corpus*, thèse de doctorat, Paris, INALCO.
- Fairon, C. 1998, « Glossanet : Parsing a web site as a corpus », *Linguisticae Investigationes*, vol. 22, n° 1, p. 327–340.
- Fairon, C. et P. Watrin. 2003, « From extraction to indexation. collecting new indexation keys by means of ie techniques », dans *Proceedings of the Workshop on Finite-State Methods in Natural Language Processing (EACL-2003)*, p. 113–118.

-
- Faro, S. et T. Lecroq. 2011, « The exact string matching problem : A comprehensive experimental evaluation », dans *Proceedings of the Prague Stringology Conference 2011*, ISBN 9788001048702, p. 1–22.
- Florian, R., A. Ittycheriah, H. Jing et T. Zhang. 2003, « Named entity recognition through classifier combination », dans *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 168–171, doi :10.3115/1119176.1119201.
- Fourour, N. 2002, « Nemesis, un système de reconnaissance incrémentielle des entités nommées pour le français », dans *Actes de la 9ème conférence sur le Traitement Automatique des Langues Naturelles (TALN'02)*, p. 265–274.
- Fourour, N. 2004, *Identification et catégorisation automatique des entités nommées dans les textes français*, thèse de doctorat, Nantes.
- Friburger, N. 2002, *Reconnaissance automatique des noms propres : application à la classification automatique de textes journalistiques*, thèse de doctorat, Université François Rabelais.
- Friburger, N., A. Dister et D. Maurel. 2000, « Améliorer le découpage des phrases sous intex », *Revue Informatique et Statistique dans les Sciences Humaines*, vol. 36, n° 1-4, p. 181–200.
- Friburger, N. et D. Maurel. 2004, « Finite-state transducer cascades to extract named entities in texts », *Theor. Comput. Sci.*, vol. 313, n° 1, doi :10.1016/j.tcs.2003.10.007, p. 93–104, ISSN 0304-3975.
- Gasiglia, N. 2004, « Faire coopérer deux concordanciers-analyseurs pour optimiser les extractions en corpus », *Revue française de linguistique appliquée*, vol. 9, n° 1, p. 45–62.
- Geierhos, M., O. Blanc et S. Bsiri. 2008, « Relax - extraction de relations sémantiques dans les contextes biographiques », *Traitement Automatique des Langues*, vol. 49, p. 167–190.
- Giménez, J. et L. Marquez. 2004, « Fast and accurate part-of-speech tagging : The svm approach revisited », *Recent Advances in Natural Language Processing III*, p. 153–162.
- Ginsburg, S., S. A. Greibach et M. A. Harrison. 1967, « One-way stack automata », *Journal of the ACM (JACM)*, vol. 14, n° 2, p. 389–418.
- Gravier, G., G. Adda, N. Paulson, M. Carré, A. Giraudel et O. Galibert. 2012, « The etape corpus for the evaluation of speech-based tv content processing in the french language », dans *LREC-Eighth international conference on Language Resources and Evaluation*, p. na.

-
- Grishman, R. et B. Sundheim. 1996, « Message understanding conference-6 : A brief history », dans *Proceedings of the 16th conference on Computational linguistics-Volume 1*, Association for Computational Linguistics, p. 466–471.
- Gross, M. 1975, *Méthodes en syntaxe : régime des constructions complétives*, vol. 1365, Hermann.
- Gross, M. 1987, « The use of finite automata in the lexical representation of natural language », dans *LITP Spring School on Theoretical Computer Science*, Springer, p. 34–50.
- Gross, M. 1993, « Local grammars and their representation by finite automata », dans *Data, Description, Discourse. Papers on the English Language in honour of John McH Sinclair*, édité par M. Hoey, Harper-Collins, p. 26–38.
- Gross, M. 1996, « Construction de grammaires locales et automates finis », dans *Working Papers 5 del Centro linguistico*, Università commerciale Luigi Bocconi, p. 1–65.
- Gross, M. 1997, « The Construction of Local Grammars », dans *Finite-State Language Processing*, édité par E. R. . Y. Schabès, MIT Press, p. 329–354.
- Grover, C., S. Givon, R. Tobin et J. Ball. 2008, « Named entity recognition for digitised historical texts. », dans *LREC*.
- Grune, D. 2014, « $l_0 = h_e(l_2 \cap l_2)$ a most wonderful theorem », .
- Hassan, A., S. Noeman et H. Hassan. 2008, « Language independent text correction using finite state automata », *IJCNLP. Hyderabad, India*.
- Hatmi, M. 2014, *Reconnaissance des entités nommées dans des documents multimodaux*, thèse de doctorat, UNIVERSITÉ DE NANTES.
- Heinz, S. et J. Zobel. 2003, « Efficient single-pass index construction for text databases », *Journal of the Association for Information Science and Technology*, vol. 54, n° 8, p. 713–729.
- Hkiri, E., S. Mallat et M. Zrigui. 2016, « Système hybride pour la reconnaissance des entités nommées arabes à base des crf », dans *JEP-TALN-RECITAL 2016*.
- Hopcroft, J. E., R. Motwani et J. D. Ullman. 2000, *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*, Addison Wesley, ISBN 0201441241.
- Ierusalimschy, R. 2016, *Programming in Lua, Fourth Edition*, Lua.Org, ISBN 978-85-903798-6-7.
- Ierusalimschy, R., L. H. de Figueiredo et W. C. Filho. 1996, « Lua – an extensible extension language », *Software : Practice and Experience*, vol. 26, n° 6, doi :10.1002/(SICI)1097-024X(199606)26:6<635::AID-SPE26>3.0.CO;2-P, p. 635–652, ISSN 0038-0644.

-
- Ierusalimschy, R., L. H. de Figueiredo et W. C. Filho. 2007, « The evolution of lua », dans *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, HOPL III, ACM, New York, NY, USA, ISBN 978-1-59593-766-7, p. 2–1–2–26, doi :10.1145/1238844.1238846.
- Impagliazzo, R. et R. Paturi. 2001, « On the complexity of k-sat », *Journal of Computer and System Sciences*, vol. 62, n° 2, doi :10.1006/jcss.2000.1727, p. 367–375, ISSN 0022-0000.
- Irvine, A., C. Callison-Burch et A. Klementiev. 2010, « Transliterating from all languages », dans *Proceedings of the Conference of the Association for Machine Translation in the Americas (AMTA)*, p. 100–110.
- Jacquet, M. E.-G. 2006, « Vers une double annotation des entités nommées », *Traitement automatique des langues*, , n° 47/1, p. 63–88.
- Johnson, D. E. et P. M. Postal. 1981, *Arc pair grammar*, Princeton University Press.
- Joshi, A. K. et Y. Schabes. 1997, « Tree-adjointing grammars », *Handbook of formal languages*, vol. 3, p. 69–124.
- Kaplan, R. M. 1972, « Augmented transition networks as psychological models of sentence comprehension », *Artificial Intelligence*, vol. 3, p. 77–100.
- Kaplan, R. M. et J. Bresnan. 1982, « Lexical-functional grammar : A formal system for grammatical representation », *Formal Issues in Lexical-Functional Grammar*, p. 29–130.
- Karlsson, F. 1990, « Constraint grammar as a framework for parsing running text », dans *Proceedings of the 13th conference on Computational linguistics- Volume 3*, Association for Computational Linguistics, p. 168–173.
- Karlsson, F., A. Voutilainen, J. Heikkilae et A. Anttila. 1995, *Constraint Grammar : a language-independent system for parsing unrestricted text*, vol. 4, Walter de Gruyter.
- Karttunen, L., J.-P. Chanod, G. Grefenstette et A. Schille. 1996, « Regular expressions for language engineering », *Natural Language Engineering*, vol. 2, n° 4, p. 305–328.
- Kevers, L. 2006, « L’information biographique : modélisation, extraction et organisation en base de connaissances », dans *Rencontre des étudiants chercheurs en informatique pour le traitement automatique des langues, Leuven, 10-13 avril 2006*, rencontre des étudiants chercheurs en informatique pour le traitement automatique des langues, Leuven, 10-13 avril 2006 (RECITAL’06), UCL, Presses Universitaires de Louvain, Leuven, Belgium, p. 680–689.
- Kiss, T. et J. Strunk. 2006, « Unsupervised multilingual sentence boundary detection », *Computational Linguistics*, vol. 32, n° 4, p. 485–525.
- Klarsfeld, G. et M. H.-M. Carthy. 1991, « Dictionnaire électronique du ladl pour les mots simples de l’anglais (DELASa) », cahier de recherche, LADL, Université Paris 7.

-
- Kogkitsidou, E. et G. Antoniadis. 2016, « L'architecture d'un modèle hybride pour la normalisation de sms », *JEP-TALN-RECITAL 2016*, vol. 2, p. 355–363.
- Kohlschütter, C., P. Fankhauser et W. Nejdl. 2010, « Boilerplate detection using shallow text features », dans *Proceedings of the third ACM international conference on Web search and data mining*, ACM, p. 441–450.
- Kooli, N. 2016, *Rapprochement de données pour la reconnaissance d'entités dans les documents ocrisés*, thèse de doctorat, Université de lorraine.
- Krstev, C., I. Obradović, M. Utvić et D. Vitas. 2013, « A system for named entity recognition based on local grammars », *Journal of Logic and Computation*, vol. 24, n° 2, p. 473–489.
- Krstev, C., D. Vitas et S. Gucul. 2005, « Recognition of personal names in serbian texts », dans *International Conference Recent Advances in Natural Language Processing (RANLP'05)*, p. 288–292.
- Kupiec, J. 1992, « Robust part-of-speech tagging using a hidden markov model », *Computer Speech & Language*, vol. 6, n° 3, p. 225–242.
- Kyriacopoulou, T., C. Martineau et T. Mavropoulos. 2011, « Les noms propres de personne en français et en grec : reconnaissance, extraction et enrichissement de dictionnaire », dans *30th International Conference on Lexis and Grammar*, Honoré Champion, p. 467–479.
- Lafferty, J. D., A. McCallum et F. C. N. Pereira. 2001, « Conditional random fields : Probabilistic models for segmenting and labeling sequence data », dans *Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN 1-55860-778-1, p. 282–289.
- Lam, Y.-K. et Q. Huo. 2005, « A data structure using hashing and tries for efficient chinese lexical access », dans *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, IEEE, p. 506–510.
- Laporte, E. 2005, « Symbolic Natural Language Processing », dans *Applied Combinatorics on Words*, édité par Lothaire, Cambridge University Press, p. 164–209.
- Laporte, É. et A. Monceaux. 1999, « Elimination of lexical ambiguities by grammars : The elag system », *Linguisticæ Investigationes*, vol. 22, n° 1, p. 341–367.
- Le Pevedic, S. et D. Maurel. 2016, « Retour sur les annotations des entités nommées dans les campagnes d'évaluation françaises et comparaison avec la tei », *Corela. Cognition, représentation, langage*, , n° 14-2.
- Levenshtein, V. I. 1966, « Binary codes capable of correcting deletions, insertions, and reversals », dans *Soviet physics doklady*, vol. 10, p. 707–710.
- Linz, P. 2016, *An Introduction to Formal Languages and Automata*, Jones and Bartlett Publishers, Inc, ISBN 1284077241.

-
- Liu, X., S. Zhang, F. Wei et M. Zhou. 2011, « Recognizing named entities in tweets », dans *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics : Human Language Technologies-Volume 1*, Association for Computational Linguistics, p. 359–367.
- Liu, X., M. Zhou, F. Wei, Z. Fu et X. Zhou. 2012, « Joint inference of named entity recognition and normalization for tweets », dans *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics : Long Papers-Volume 1*, Association for Computational Linguistics, p. 526–535.
- Lopez-Moreno, I., J. Gonzalez-Dominguez, O. Plchot, D. Martinez, J. Gonzalez-Rodriguez et P. Moreno. 2014, « Automatic language identification using deep neural networks », dans *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, IEEE, p. 5337–5341.
- Makhoul, J., F. Kubala, R. Schwartz, R. Weischedel et al.. 1999, « Performance measures for information extraction », dans *Proceedings of DARPA broadcast news workshop*, p. 249–252.
- Manning, C. D., M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard et D. McClosky. 2014, « The Stanford CoreNLP natural language processing toolkit », dans *Association for Computational Linguistics (ACL) System Demonstrations*, p. 55–60.
- Marcus, M. P., M. A. Marcinkiewicz et B. Santorini. 1993, « Building a large annotated corpus of english : The penn treebank », *Comput. Linguist.*, vol. 19, n° 2, p. 313–330, ISSN 0891-2017.
- Martineau, C., E. Tolone et S. Voyatzi. 2007, « Les entités nommées : usage et degrés de précision et de désambiguïation », dans *26^{ème} Colloque international sur le Lexique et la Grammaire (LGC'07)*, p. pages–105.
- Martineau, C., S. Voyatzi, L. Varga, S. Brizard et A. Migeotte. 2011, « Détection fine d'opinion et sentiments : attribution fine de la polarité et calcul incrémental de l'intensité », dans *30th International Conference on Lexis and Grammar*, édité par F. Kakoyianni-Doa, Honoré Champion, Nicosia, Cyprus, p. 319–334.
- Masek, W. J. et M. S. Paterson. 1980, « A faster algorithm computing string edit distances », *Journal of Computer and System Sciences*, vol. 20, n° 1, doi :10.1016/0022-0000(80)90002-1, p. 18–31, ISSN 0022-0000.
- Maurel, D. 1993, « Reconnaissance automatique d'un groupe nominal prépositionnel. exemple des adverbes de date », *Lexique 11/Les prépositions-méthodes d'analyse*, vol. 11, p. 147.
- Maurel, D. 2004, « Les mots inconnus sont-ils des noms propres », *Actes des JADT*.
- Maurel, D., N. Friburger, J.-Y. Antoine, I. Eshkol et D. Nouvel. 2011, « Cascades de transducteurs autour de la reconnaissance des entités nommées », *Traitement automatique des langues*, vol. 52, n° 1, p. 69–96.

-
- McCallum, A. et W. Li. 2003, « Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons », dans *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, Association for Computational Linguistics, p. 188–191.
- Meur, C., S. Galliano et E. Geoffrois. 2004, « Conventions d'annotations en entités nommées-ester », *Rapport technique de la campagne Ester*.
- Miller, D., S. Boisen, R. Schwartz, R. Stone et R. Weischedel. 2000, « Named entity extraction from noisy input : speech and ocr », dans *Proceedings of the sixth conference on Applied natural language processing*, Association for Computational Linguistics, p. 316–324.
- Mitkov, R. 1999, *Anaphora resolution : the state of the art*, School of Languages and European Studies, University of Wolverhampton.
- Monceaux, A. 1995, « Le dictionnaire des mots simples anglais : Mots nouveaux et variantes orthographiques », cahier de recherche 15, IGM, Université de Marne-la-Vallée.
- Monnier, S., A. Lacheret-Dujour et M. Morel. 2003, « From syntactic and semantic analysis to prosodic perspectives : an automatic approach », dans *actes du 15th ICPPhS*, éd. UAB, Barcelone, Spain, p. 1137–1140.
- Morel, M. et A. Lacheret-Dujour. 2001, « Kali, synthèse vocale à partir du texte : De la conception à la mise en oeuvre », *Traitement automatique des langues*, vol. 42, p. 193–221.
- Myers, E. W. 1994, « A sublinear algorithm for approximate keyword searching », *Algorithmica*, vol. 12, n° 4-5, doi :10.1007/BF01185432, p. 345–374, ISSN 01784617.
- Nakamura, T. 2004, « Analyse automatique d'un discours spécialisé au moyen de grammaires locales », dans *7^{ième} Journées internationales d'Analyse Statistique des Données Textuelles (JADT'04)*, vol. 2, édité par A. D. Gérard Purnelle, Cédric Fairon, Presses Universitaires de Louvain, Louvain-la-Neuve, Belgium, p. 837–847.
- Nam, J.-s. et K.-s. Choi. 1997, « A local-grammar-based approach to recognizing of proper names in korean texts », dans *Proceedings of the Workshop on Very Large Corpora, ACL/Tsing-hua University/Hong-Kong University of Science and Technology*, p. 273–288.
- Nietzio, A. 2002, « Support vector machines for part-of-speech tagging », dans *6. Konferenz zur Verarbeitung natürlicher Sprache (KONVENS 2002)*, p. 223–226.
- Nouvel, D. 2012, *Reconnaissance des entités nommées par exploration de règles d'annotation-Interpréter les marqueurs d'annotation comme instructions de structuration locale*, thèse de doctorat, Université François Rabelais-Tours.
- Packer, T. L., J. F. Lutes, A. P. Stewart, D. W. Embley, E. K. Ringger, K. D. Seppi et L. S. Jensen. 2010, « Extracting person names from diverse and noisy ocr text »,

-
- dans *Proceedings of the fourth workshop on Analytics for noisy unstructured text data*, ACM, p. 19–26.
- Paik, W., E. D. Liddy, E. Yu et M. McKenna. 1996, « Categorizing and standardizing proper nouns for efficient information retrieval », *Corpus processing for lexical acquisition*, p. 61–73.
- Palmer, D. D. et D. S. Day. 1997, « A statistical profile of the named entity task », dans *Proceedings of the fifth conference on Applied natural language processing*, Association for Computational Linguistics, p. 190–193.
- Patel, N., P. Accorsi, D. Inkpen, C. Lopez et M. Roche. 2013, « Approaches of anonymisation of an sms corpus », dans *CICLing : Conference on Intelligent Text Processing and Computational Linguistics*, 7816, Springer-Verlag, p. 77–88.
- Paumier, S. 2003a, *De la reconnaissance des formes linguistiques à l'analyse syntaxique*, thèse de doctorat, Université de Marne-la-Vallée. Thèse de doctorat dirigée par Gross, Maurice et Laporte, Eric Informatique linguistique Université de Marne-la-Vallée 2003.
- Paumier, S. 2003b, « A time-efficient token representation for parsers », dans *Proceedings of the EACL Workshop on Finite-State Methods in Natural Language Processing*, p. 83–90.
- Paumier, S. 2016, « Unitex/GramLab Users' Manual v3.1 », <http://releases.unitexgramlab.org/3.1/man/Unitex-GramLab-3.1-usermanual-en.pdf>.
- Perez-Marin, D. 2011, *Conversational Agents and Natural Language Interaction : Techniques and Effective Practices : Techniques and Effective Practices*, IGI Global.
- Perlmutter, D. M. 1980, « Relational grammar in current approaches to syntax. », *Syntax and Semantics Ann Arbor, Mich.*, vol. 13, p. 195–229.
- Perrin, D. 1990, « Finite automata », dans *Formal Models and Semantics*, Elsevier, p. 1–57.
- Pirovani, J. P. C. et E. Silva de Oliveira. 2015, « Extraction of person names from texts in Portuguese : an approach with local grammars », dans *Computer on the Beach*, édité par A. M. da Rocha Fernandes, Anais do Computer on the Beach, Universidade do Vale do Itajaí (UNIVALI) and Centro de Ciências Tecnológicas da Terra e do Mar (CTTMar), Florianópolis, Brazil, p. 1–10. ISSN : 2358-0852.
- Plamondon, L., G. Lapalme et F. Pelletier. 2004, « Anonymisation de décisions de justice », dans *XIe Conférence sur le Traitement Automatique des Langues Naturelles (TALN 2004)*, p. 367–376.
- Poibeau, T. 2005, « Sur le statut référentiel des entités nommées », dans *Conférence Traitement Automatique des Langues 2005*, édité par M. Jardino, Association pour le Traitement Automatique des Langues / LIMSI, Dourdan, France, p. 173–183. [Http ://taln.limsi.fr/site/talnRecital05/tome1/P18.pdf](http://taln.limsi.fr/site/talnRecital05/tome1/P18.pdf).

-
- Poibeau, T., A. Acoulon, C. Avaux, L. Beroff-Bénéat, A. Cadeau, M. Calberg, A. Delale, L. De Temmerman, A.-L. Guenet, D. Huis, M. Jamalpour, A. Krul, A. Marcus, F. Picoli et C. Plancq. 2003, « The multilingual named entity recognition framework », dans *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 2*, Association for Computational Linguistics, p. 155–158.
- Polifroni, J., I. Kiss et M. Adler. 2010, « Bootstrapping named entity extraction for the creation of mobile services. », dans *LREC*.
- Pollard, C. et I. A. Sag. 1994, *Head-driven phrase structure grammar*, University of Chicago Press.
- Rabin, M. O. et D. Scott. 1959, « Finite automata and their decision problems », *IBM J. Res. Dev.*, vol. 3, n° 2, doi :10.1147/rd.32.0114, p. 114–125, ISSN 0018-8646.
- Ramage, D., D. Hall, R. Nallapati et C. D. Manning. 2009, « Labeled lda : A supervised topic model for credit attribution in multi-labeled corpora », dans *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing : Volume 1-Volume 1*, Association for Computational Linguistics, p. 248–256.
- Ranchhod, E., P. Carvalho, C. Mota et A. Barreiro. 2004, « Portuguese large-scale language resources for NLP applications », dans *Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004, May 26-28, 2004, Lisbon, Portugal*, p. 1755–1758.
- Ratnaparkhi, A. 1997, « A simple introduction to maximum entropy models for natural language processing », *IRCS Technical Reports Series*, p. 81.
- Revuz, D. 1992, « Minimisation of acyclic deterministic automata in linear time », *Theoretical Computer Science*, vol. 92, n° 1, p. 181–189.
- Ritter, A., S. Clark, O. Etzioni et al.. 2011, « Named entity recognition in tweets : an experimental study », dans *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, p. 1524–1534.
- Roman, M., C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell et K. Nahrstedt. 2002, « A middleware infrastructure for active spaces », *IEEE Pervasive Computing*, vol. 1, n° 4, doi :10.1109/MPRV.2002.1158281, p. 74–83, ISSN 1536-1268.
- Sætre, R. 2003, « Natural language understanding (nlu) automatic information extraction (ie) from biomedical texts », .
- Sagot, B. et P. Boullier. 2005, « From raw corpus to word lattices : robust pre-parsing processing with sxpipe », *Archives of Control Sciences*, vol. 15, n° 4, p. 653–662.
- Sagot, B. et P. Boullier. 2008, « Sxpipe 2 : architecture pour le traitement pré-syntaxique de corpus bruts », *Traitement Automatique des Langues*, vol. 49, n° 2, p. 155–188.

-
- Sagot, B. et D. Fišer. 2008, « Building a free french wordnet from multilingual resources », dans *OntoLex*.
- Sagot, B. et K. Gábor. 2014, « Détection et correction automatique d'entités nommées dans des corpus ocrisés », dans *Traitement Automatique du Langage Naturel 2014*.
- Saidi, A. S. 2004, « Using linguistic structures in textual information extraction », dans *Fifth International Workshop on Multimedia Data Mining*, p. 94.
- Sakarovitch, J. 2009, *Elements of Automata Theory*, Cambridge University Press, New York, NY, USA, ISBN 0521844258, 9780521844253.
- Sastre, J. M. 2011, *Efficient finite-state algorithms for the application of local grammars*, Theses, Université Paris-Est.
- Sastre, J. M. et M. L. Forcada. 2009, « Efficient parsing using recursive transition networks with output », dans *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5642 LNCS, ISBN 3642029787, ISSN 03029743, p. 241–244, doi :10.1007/978-3-642-02979-0_28.
- Sekine, S., K. Sudo et C. Nobata. 2002, « Extended named entity hierarchy. », dans *The Third International Conference on Language Resources and Evaluation (LREC). Iles Canaries, Espagne*.
- Sigogne, A. 2010, « Hybridtagger : un étiqueteur hybride pour le français », dans *Actes de MajecSTIC 2010*, p. 8.
- Silberztein, M. 1993, *Dictionnaires électroniques et analyse automatique de textes : Le système Intex*, Collection informatique linguistique, Masson, ISBN 9782225841576.
- Silberztein, M. 2000, « Intex : An fst toolbox », *Theoretical Computer Science*, vol. 231, n° 1, doi :10.1016/S0304-3975(99)00015-8, p. 33–46, ISSN 0304-3975.
- Silberztein, M. 2003, « Nooj v5.0 manual », *Download from <http://www.nooj-association.org>*, p. 218.
- Silberztein, M. 2004, « NooJ : a cooperative object oriented architecture for NLP », dans *INTEX pour la linguistique et le traitement automatique des langues, Cahiers de la MSH Ledoux*, vol. 1, Presses Universitaires de Franche-Comté, p. 351–362.
- Silberztein, M. 2015, *La formalisation des langues : l'approche de NooJ*, ISTE, ISBN 978-1-78405-053-5.
- Silberztein, M. et A. Tutin. 2005, « NooJ, un outil TAL pour l'enseignement des langues. Application pour l'étude de la morphologie lexicale en FLE », *Apprentissage des Langues et Systèmes d'Information et de Communication*, vol. 08, n° 1, p. 123–134. Cet article fait partie d'une sélection d'articles de la journée Atala d'octobre 2004 à Grenoble.

-
- Silberztein, M. D. 1994, « Intex : A corpus processing system », dans *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*, COLING '94, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 579–583, doi : 10.3115/991886.991988.
- Sipser, M. 2012, *Introduction to the Theory of Computation*, vol. 3, Cengage Learning, ISBN 113318779X.
- Small, C. G. 2010, *Expansions and asymptotics for statistics*, Chapman and Hall/CRC Press, ISBN 1584885904.
- T. Bocek, B. S., E. Hunt. 2007, « Fast Similarity Search in Large Dictionaries », cahier de recherche ifi-2007.02, Department of Informatics, University of Zurich. [Http ://fastss.csg.uzh.ch/](http://fastss.csg.uzh.ch/).
- Teahan, W. J. et D. J. Harper. 2003, « Using compression-based language models for text categorization », dans *Language modeling for information retrieval*, Springer, p. 141–165.
- Thorne, J. P., P. Bratley et H. Dewar. 1968, « The syntactic analysis of english by machine », dans *Machine Intelligence 3*, édité par D. Michie, Edinburgh University Press, Edinburgh, Scotland, p. 281–310.
- Tjong Kim Sang, E. 2002, « Introduction to the conll-2002 shared task : language-independent named entity recognition, proceedings of the 6th conference on natural language learning », *August*, vol. 31, p. 1–4.
- Tjong Kim Sang, E. F. et F. De Meulder. 2003, « Introduction to the conll-2003 shared task : Language-independent named entity recognition », dans *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 142–147, doi :10.3115/1119176.1119195.
- Traboulsi, H. 2004, *A Local Grammar for Proper Names*, mémoire de maîtrise, University of Surrey, UK.
- Traboulsi, H. N. 2005, « Towards the automatic acquisition of local grammars », dans *Proceedings of the 3rd Annual PhD Conference*, Department of Computing, School of Electronics and Physical Sciences, University of Surrey, p. 13–18.
- Uszkoreit, H. 1986, « Categorical unification grammars », dans *Proceedings of the 11th conference on Computational linguistics*, Association for Computational Linguistics, p. 187–194.
- Vicente, M. R. 2005, « La glose comme outil de désambiguïsation référentielle des noms propres purs », *Corela. Cognition, représentation, langage*, , n° HS-2.
- Wagner, R. A. et M. J. Fischer. 1974, « The string-to-string correction problem », *J. ACM*, vol. 21, n° 1, doi :10.1145/321796.321811, p. 168–173, ISSN 0004-5411.

-
- Watrin, P., L. De Viron, D. Lebailly, M. Constant et S. Weiser. 2014, « Named entity recognition for german using conditional random fields and linguistic resources », dans *GermEval 2014 Named Entity Recognition Shared Task-KONVENS 2014 Workshop*.
- Watson, B. W. 2003, « A new algorithm for the construction of minimal acyclic s », *Science of Computer Programming*, vol. 48, n° 2-3, p. 81–97.
- Wong, C. K. et A. K. Chandra. 1976, « Bounds for the string editing problem », *J. ACM*, vol. 23, n° 1, doi :10.1145/321921.321923, p. 13–16, ISSN 0004-5411.
- Woods, W. A. 1970, « Transition network grammars for natural language analysis », *Commun. ACM*, vol. 13, n° 10, doi :10.1145/355598.362773, p. 591–606, ISSN 0001-0782.

Glossaire

C | F | I | P

C

Collocation Co-occurrence de deux ou plusieurs mots qui sont liés sémantiquement 56

F

Framework (*cadriciel*) Ensemble cohérent d'outils, de composants logiciels, d'APIs et de conventions qui définissent une architecture d'application de base 98

I

Interface de Programmation d'Application (API, *Application Programming Interface*) Un en-

semble de règles et de spécifications qu'un programme peut suivre pour accéder et utiliser les services et les ressources fournis par un autre logiciel particulier qui implémente cette API..... 193

P

Pile Structure de données capable de stocker et récupérer de l'information suivant un mode d'accès dit « *dernier arrivé, premier sorti* » 34

Prosodie Étude de l'accentuation et de l'intonation et de leur contribution à la sémantique de l'annonce 61

Abréviations

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [L](#) | [N](#) | [O](#) | [P](#) | [R](#) | [S](#) | [T](#) | [W](#)

A

API Interface de Programmation d'Application (*Application Programming Interface*) .. 193, *Glossaire* : [Interface de Programmation d'Application](#)
ATN réseau de transitions augmenté (*augmented transition network*) 72, 73

B

BNF forme de Backus–Naur (*Backus–Naur form*)
6, 43, 144

C

CFG grammaire non-contextuelle (*context-free grammar*) 31, 67
CFL langage non-contextuel (*context-free language*) 31
CRF champs aléatoires conditionnels (*conditional random fields*) 133

D

DELA dictionnaires électroniques du LADL... 47
DELAC dictionnaires électroniques de formes composées 138
DELACF dictionnaires électroniques de formes composées fléchies 138
DELAFL dictionnaires électroniques de formes fléchies 12, 138
DELAP dictionnaires électroniques de formes phonémiques 138
DELAS dictionnaires électroniques de formes simples 138
DFA automate fini déterministe (*deterministic finite automaton*) 25

E

ELAG grammaires de levée d'ambiguïtés (*elimination of lexical ambiguity through grammars*) 133

ELG grammaire locale étendue (*extended local grammar*) xxiv, xxix, 68, 98

F

FSA automate fini (*finite-state automaton*) .. xix, 41, 44, 76
FST transducteur fini (*finite-state transducer*) xix, 46, 47

G

GRAAL Bibliothèque de grammaires et d'automates (*Grammar and automata library*) 62

H

HMM modèle de Markov cachés (*hidden Markov models*) 133

L

LADL Laboratoire d'Automatique Documentaire et Linguistique 61
LG grammaire locale (*local grammar*) xix, xxix, 6, 41, 67
LIGM Laboratoire d'Informatique Gaspard-Monge 48

N

NER reconnaissance d'entités nommées (*named-entity recognition*) 2
NFA automate fini non-déterministe (*non-deterministic finite automaton*) xix, 25, 27, 76
NPDA automate à pile non-déterministe (*non-deterministic pushdown automata*) 34

O	S
OCR reconnaissance optique de caractères (<i>optical character recognition</i>)..... 149	SVM machines à vecteurs de support (<i>support vector machines</i>)..... 133
P	T
PDA automate à pile (<i>pushdown automata</i>)... 34	TAL traitement automatique des langues 2, 43, 67
PDT transducteur à pile (<i>pushdown transducer</i>).. 49	TM Machine de Turing (<i>Turing Machine</i>)..... 38
R	W
RTN réseau de transitions récursif (<i>recursive transition network</i>)..... xix, 6, 43, 67	WFST transducteur pondéré (<i>weighted finite-state transducer</i>)..... 48

Index

Symbols

ε -NFAxix, 27, 76

A

alphabet 170
API 193, 193
ATN 72, 73, 193

B

BNF 6, 43, 48, 144, 193

C

categorie
 syntaxique 31, 32, 167
CFG 31, 32, 37, 67, 193
CFL 31, 193
collocation 56, 191
concaténation 171
CRF 133, 193

D

DELA 47, 193
DELAC 138, 193
DELACF 138, 193
DELAf 12, 138, 193
DELAP 138, 193
DELAS 138, 193
DFA 25, 193
diagramme d'états-transitions xxix, 23
diagrammes de Venn 163
distance
 euclidienne 63

E

ELAG 133, 193
ELG
 init_event 99
 load_event 99
 onbacktrack_event 99
 onenter_event 99
 onexit_event 99
 onfail_event 99
 onmatch_event 99
 slide_event 99

 token_event 99
 unload_event 99
ELGxxiv, xxv, xxix, xxxii, 68, 70, 73–75, 81, 83,
 84, 98, 102, 103, 193
ensemble 161
 vide 161
 cardinalité 162
 complément 163
 des parties 164
 différence 163
 disjoint 163
 fini 161
 infini 161
 intersection 162
 plein 163
 produit cartésien 164
 union 162

F

fonction 165
 antécédent 165
 application 165
 argument 72, 166
 binaire 166
 domaine 165
 ensemble d'arrivée 165
 ensemble de départ 165
 entrée 75, 165
 image 165
 n-aire 166
 sortie 75, 165
 unaire 166
 variable 75, 165
framework 98, 191
FSA xix, 41, 44, 46, 47, 76, 193
FST xix, 46, 47, 193

G

GRAAL 62, 193

H

HMM 133, 193

- I**
Interface de Programmation d'Application **191**,
193
- L**
LADL **61**, **193**
langage **172**
 équivalent **173**
 étoile de Kleene **174**
 complémentaire **173**
 concaténation **173**
 fini **172**
 formel **174**
 infini **172**
 monoïde **174**
 opérations **173**
 puissance **173**
 vide **173**
LGxix, xxv, xxix, xxxii, 6, 41–44, 46, 47, 67, 72,
75, 81, 102, 103, **193**
LIGM **48**, **193**
- M**
mot **170**
 ensemble **171**
 facteur **172**
 facteur propre **172**
 longueur **170**
 occurrences **171**
 préfixe **172**
 préfixe propre **172**
 puissance **171**
 sous-mot **172**
 suffixe **172**
 suffixe propre **172**
 vide **170**
- N**
NER **2**, **193**
NFA xix, 25, 27, 31, 34, **193**
notation de Bachmann-Landau **166**
NPDA **34**, **193**
- O**
OCR **149**, **194**
Outils
 CASEN **153**
 CORDIAL **62**
 CORENLP **150**, **153**
 GLOSSANET **55**, **57**
 GRAALWEB **62**
 HYBRIDTAGGER **133**
 INTEX xxix, 7, 44, 47, 48
 KALI **61**
 LGFINDER **63**
 LIBPOSTAL **156**
 NOOJ 7, 44, 48, 54, 72, 133
 OPENFST 7, 44, 48, 49, 81
 OPENNLP **150**, **153**
 OUTILEX 7, 44, 48, 59, 81
 PELIAS **156**
 SXPIPE 6, 43, 48, 144, 145
 SYNTAX 6, 43, 48, 144
 UNITEX/GRAMLAB **48**
 UNITEX . xxix, xxxii, 7, 11, 12, 44, 48, 59,
 62, 63, 72, 81, 91, 92, 94, 97, 99, 100,
 133, 149, 160
 UNITEX2SXPIPE **144**
- P**
PDA **34**, **194**
PDT **49**, **194**
pile **34**, **191**
prosodie **61**, **191**
- R**
relation **167**
 binaire **167**
 d'équivalence **168**
 d'appartenance **161**
 réflexive **168**
 symétrique **168**
 transitive **168**
RTN xix, 6, 7, 43–47, 49, 59, 67, 72, 73, **194**
- S**
sous-ensemble **162**
 propre **162**
suite **163**, **170**
SVM **133**, **194**
symboles **74**, **170**
- T**
TAL **2**, **3**, **6**, **43**, **67**, **194**
TM **38**, **194**
tuple **163**
- U**
uplet **163**
- W**
WFST **48**, **49**, **194**