

Table des matières

Introduction	9
1 Installation d’Unitex	11
1.1 Licences	11
1.2 Environnement d’exécution Java	11
1.3 Installation sous Windows	12
1.4 Installation sous Linux et MacOS	12
1.5 Première utilisation	12
1.6 Ajout de nouvelles langues	12
1.7 Désinstallation	13
2 Chargement d’un texte	15
2.1 Sélection de la langue	15
2.2 Format des textes	15
2.3 Edition de textes	17
2.4 Ouverture d’un texte	18
2.5 Prétraitement du texte	20
2.5.1 Normalisation des séparateurs	21
2.5.2 Découpage en phrases	21
2.5.3 Normalisation de formes non ambiguës	23
2.5.4 Découpage du texte en unités lexicales	23
2.5.5 Application de dictionnaires	25
2.5.6 Analyse des mots composés libres en allemand, norvégien et russe	28
3 Dictionnaires	29
3.1 Les dictionnaires DELA	29
3.1.1 Format des DELAF	29
3.1.2 Format des DELAS	32
3.1.3 Contenu des dictionnaires	32
3.2 Vérification du format d’un dictionnaire	34
3.3 Tri	36
3.4 Flexion automatique	37
3.5 Compression	39
3.6 Application de dictionnaires	41
3.6.1 Priorités	41

3.6.2	Règles d'application des dictionnaires	41
3.7	Bibliographie	42
4	Recherche d'expressions rationnelles	43
4.1	Définition	43
4.2	Unités lexicales	43
4.3	Motifs	44
4.3.1	Symboles spéciaux	44
4.3.2	Références aux dictionnaires	44
4.3.3	Contraintes grammaticales et sémantiques	45
4.3.4	Contraintes flexionnelles	45
4.3.5	Négation d'un motif	46
4.4	Concaténation	47
4.5	Union	48
4.6	Étoile de Kleene	48
4.7	Filtres morphologiques	48
4.8	Recherche	50
4.8.1	Configuration de la recherche	50
4.8.2	Affichage des résultats	51
5	Grammaires locales	55
5.1	Formalisme des grammaires locales	55
5.1.1	Grammaires algébriques	55
5.1.2	Grammaires algébriques étendues	56
5.2	Édition de graphes	56
5.2.1	Importation d'un graphe Intex	56
5.2.2	Création d'un graphe	57
5.2.3	Sous-graphes	59
5.2.4	Manipulation des boîtes	60
5.2.5	Sortie	61
5.2.6	Utilisation des variables	61
5.2.7	Copie de listes	62
5.2.8	Symboles spéciaux	63
5.2.9	Commandes de la barre d'icônes	64
5.3	Options de présentation	65
5.3.1	Tri des lignes d'une boîte	65
5.3.2	Zoom	65
5.3.3	Antialiasing	66
5.3.4	Alignement des boîtes	67
5.3.5	Présentation, polices et couleurs	67
5.4	Les graphes en dehors d'Unitex	70
5.4.1	Inclusion d'un graphe dans un document	70
5.4.2	Impression d'un graphe	71

6	Utilisation avancée des graphes	73
6.1	Les types de graphes	73
6.1.1	Graphes de flexion	73
6.1.2	Graphes de prétraitement	74
6.1.3	Graphes de normalisation de l'automate du texte	75
6.1.4	Graphes syntaxiques	76
6.1.5	Grammaires ELAG	76
6.1.6	Graphes paramétrés	76
6.2	Compiler une grammaire	76
6.2.1	Compilation d'un graphe	76
6.2.2	Approximation par un transducteur à états finis	77
6.2.3	Contraintes sur les grammaires	78
6.2.4	Détection d'erreurs	81
6.3	Exploration des chemins d'une grammaire	81
6.4	Collection de graphes	83
6.5	Règles d'application des transducteurs	84
6.5.1	Insertion à gauche du motif reconnu	84
6.5.2	Application en avançant	85
6.5.3	Priorité à gauche	85
6.5.4	Priorité aux séquences les plus longues	86
6.5.5	Sorties à variables	86
6.6	Application des graphes aux textes	88
6.6.1	Configuration de la recherche	88
6.6.2	Concordance	89
6.6.3	Modification du texte	90
6.6.4	Extraction des occurrences	91
7	Automate du texte	93
7.1	Présentation	93
7.2	Construction	94
7.2.1	Règles de construction de l'automate du texte	94
7.2.2	Normalisation de formes ambiguës	95
7.2.3	Normalisation des pronoms clitiques en portugais	97
7.2.4	Conservation des meilleurs chemins	97
7.3	Levée d'ambiguïtés lexicales avec ELAG	100
7.3.1	Grammaires de levée d'ambiguïtés	100
7.3.2	Compilation des grammaires ELAG	102
7.3.3	Levée d'ambiguïtés	102
7.3.4	Ensembles de grammaires	105
7.3.5	Fenêtre de processing d'ELAG	106
7.3.6	Description du jeu d'étiquettes	107
7.3.7	Optimiser les grammaires	112
7.4	Manipulation de l'automate du texte	114
7.4.1	Affichage des automates de phrases	114
7.4.2	Modifier manuellement l'automate du texte	115

7.4.3	Paramètres de présentation	116
8	Lexique-grammaire	117
8.1	Les tables de lexique-grammaire	117
8.2	Conversion d'une table en graphes	118
8.2.1	Principe des graphes paramétrés	118
8.2.2	Format de la table	118
8.2.3	Les graphes paramétrés	119
8.2.4	Génération automatique de graphes	120
9	Utilisation des programmes externes	125
9.1	CheckDic	125
9.2	Compress	125
9.3	Concord	126
9.4	Convert	127
9.5	Dico	128
9.6	Elag	129
9.7	ElagComp	129
9.8	Evamb	130
9.9	ExploseFst2	130
9.10	Extract	130
9.11	Flatten	130
9.12	Fst2Grf	131
9.13	Fst2List	131
9.14	Fst2Txt	132
9.15	Grf2Fst2	133
9.16	ImploseFst2	133
9.17	Inflect	133
9.18	Locate	133
9.19	MergeTextAutomaton	134
9.20	Normalize	134
9.21	PolyLex	135
9.22	Reconstrucao	135
9.23	Reg2Grf	135
9.24	SortTxt	136
9.25	Table2Grf	136
9.26	TextAutomaton2Mft	136
9.27	Tokenize	136
9.28	Txt2Fst2	137
10	Formats de fichiers	139
10.1	Codage Unicode Little-Endian	139
10.2	Fichiers d'alphabet	140
10.2.1	Alphabet	140
10.2.2	Alphabet de tri	141

10.3	Graphes	141
10.3.1	Format .grf	141
10.3.2	Format .fst2	144
10.4	Textes	145
10.4.1	Fichiers .txt	145
10.4.2	Fichiers .snt	146
10.4.3	Fichier text.cod	146
10.4.4	Fichier tokens.txt	146
10.4.5	Fichiers tok_by_alph.txt et tok_by_freq.txt	146
10.4.6	Fichier enter.pos	146
10.5	Automate du texte	146
10.5.1	Fichier text.fst2	146
10.5.2	Fichier cursentence.grf	147
10.5.3	Fichier sentenceN.grf	147
10.5.4	Fichier cursentence.txt	148
10.6	Concordances	148
10.6.1	Fichier concord.ind	148
10.6.2	Fichier concord.txt	149
10.6.3	Fichier concord.html	149
10.7	Dictionnaires	150
10.7.1	Fichiers .bin	150
10.7.2	Fichiers .inf	151
10.7.3	Fichier CHECK_DIC.TXT	152
10.8	Fichiers d'ELAG	154
10.8.1	Fichier tagset.def	154
10.8.2	Fichiers .lst	154
10.8.3	Fichiers .elg	154
10.8.4	Fichiers .rul	154
10.9	Fichiers de configuration	155
10.9.1	Fichier Config	155
10.9.2	Fichier system_dic.def	156
10.9.3	Fichier user_dic.def	157
10.9.4	Fichier user.cfg	157
10.10	Fichiers divers	157
10.10.1	Fichiers dlf.n, dlc.n et err.n	157
10.10.2	Fichier stat_dic.n	157
10.10.3	Fichier stats.n	157
10.10.4	Fichier concord.n	158

Annexe A - GNU General Public License **159**

Annexe B - GNU Lesser General Public License **165**

Introduction

Unitex est un ensemble de logiciels permettant de traiter des textes en langues naturelles en utilisant des ressources linguistiques. Ces ressources se présentent sous la forme de dictionnaires électroniques, de grammaires et de tables de lexique-grammaire. Elles sont issues de travaux initiés sur le français par Maurice Gross au Laboratoire d'Automatique Documentaire et Linguistique (LADL). Ces travaux ont été étendus à d'autres langues au travers du réseau de laboratoires RELEX.

Les dictionnaires électroniques décrivent les mots simples et composés d'une langue en leur associant un lemme ainsi qu'une série de codes grammaticaux, sémantiques et flexionnels. La présence de ces dictionnaires constitue une différence majeure par rapport aux outils usuels de recherche de motifs, car on peut faire référence aux informations qu'ils contiennent et ainsi décrire de larges classes de mots avec des motifs très simples. Ces dictionnaires sont représentés selon le formalisme DELA et ont été élaborés par des équipes de linguistes pour plusieurs langues (français, anglais, grec, italien, espagnol, allemand, thaï, coréen, polonais, norvégien, portugais, etc...).

Les grammaires sont des représentations de phénomènes linguistiques par réseaux de transitions récursifs (RTN), un formalisme proche de celui des automates à états finis. De nombreuses études ont mis en évidence l'adéquation des automates aux problèmes linguistiques et ce, aussi bien en morphologie qu'en syntaxe ou en phonétique. Les grammaires manipulées par Unitex reprennent ce principe, tout en reposant sur un formalisme encore plus puissant que les automates. Ces grammaires sont représentées au moyen de graphes que l'utilisateur peut aisément créer et mettre à jour.

Les tables de lexique-grammaire sont des matrices décrivant les propriétés de certains mots. De telles tables ont été élaborées pour tous les verbes simples du français dont elles décrivent les propriétés syntaxiques. L'expérience ayant montré que chaque mot a un comportement quasi unique, ces tables permettent de donner la grammaire de chaque élément de lexique, d'où le nom de lexique-grammaire. Unitex permet de construire des grammaires à partir de telles tables.

Unitex est un moteur permettant d'exploiter ces ressources linguistiques. Ses caractéristiques techniques sont la portabilité, la modularité, la possibilité de gérer des langues possédant des systèmes d'écritures particuliers comme certaines langues asiatiques et l'ouverture, grâce à une distribution en logiciel libre. Ses caractéristiques linguistiques sont celles qui ont motivé l'élaboration des ressources: la précision, l'exhaustivité et la prise en compte des phénomènes de figement, notamment en ce qui concerne le recensement des mots composés.

Le premier chapitre décrit l'installation et le lancement d'Unitex.

Le chapitre 2 présente les différentes étapes du traitement d'un texte.

Le chapitre 3 décrit le formalisme des dictionnaires électroniques DELA ainsi que les différentes opérations qui peuvent leur être appliquées.

Les chapitres 4 et 5 présentent les différents moyens d'effectuer des recherches de motifs dans des textes. Le chapitre 5 décrit en détail l'utilisation de l'éditeur de graphes.

Le chapitre 6 est consacré aux différentes utilisations possibles des grammaires. Les particularités de chaque type de grammaires y sont présentées.

Le chapitre 7 introduit le concept d'automate du texte et décrit les particularités de cet objet. Il décrit également les opérations que l'on peut effectuer sur cet objet, notamment la levée d'ambiguïtés lexicales au moyen du programme ELAG.

Le chapitre 8 est constitué d'une introduction aux tables de lexique-grammaire, suivie par la description de la méthode permettant de construire des grammaires à partir de ces tables.

Le chapitre 9 décrit en détail les différents programmes externes qui constituent Unitex.

Le chapitre 10 donne la description de tous les formats des fichiers utilisés par le système.

Chapitre 1

Installation d'Unitex

Unitex est un système multi-plateformes capable de fonctionner aussi bien sous Windows que sous Linux ou MacOS. Ce chapitre décrit l'installation et le lancement d'Unitex pour chacun de ces systèmes. Il présente également les procédures d'ajout de nouvelles langues et de désinstallation.

1.1 Licences

Unitex est un logiciel libre. Cela signifie que les sources des programmes sont distribuées avec le logiciel, et que chacun peut les modifier et les redistribuer. Le code des programmes d'Unitex est sous licence LGPL ([22]), à l'exception de la bibliothèque de manipulation d'expressions régulières TRE de Ville Laurikari ([35]), qui est sous licence GPL ([21]). La licence LGPL est plus permissive que la licence GPL, car elle permet d'utiliser du code LGPL dans des logiciels non libres. Du point de vue de l'utilisateur, il n'y a pas de différence, car dans les deux cas, le logiciel peut être librement utilisé et distribué.

1.2 Environnement d'exécution Java

Unitex est composé d'une interface graphique écrite en Java et de programmes externes écrits en *C/C++*. Ce mélange de langages de programmation permet d'avoir une application rapide et portable sous différents systèmes d'exploitation. Afin de pouvoir utiliser l'interface graphique, il faut préalablement installer un environnement d'exécution, communément appelé machine virtuelle Java ou JRE (Java Runtime Environment).

Pour fonctionner en mode graphique, Unitex nécessite une version 1.4 (ou plus récente) de Java. Si vous avez une version trop ancienne de Java, Unitex se bloquera après que vous ayez choisi votre langue de travail. Vous pouvez télécharger librement la machine virtuelle correspondant à votre système d'exploitation sur le site de Sun Microsystems [37] à l'adresse suivante: <http://java.sun.com>. Si vous travaillez sous Linux ou MacOS, ou si vous utilisez une version de Windows gérant des comptes personnels pour les utilisateurs, il vous faudra demander à votre administrateur système d'installer Java.

1.3 Installation sous Windows

Si vous désirez installer Unitex sur une machine Windows multi-utilisateurs, il est préférable de demander à votre administrateur de le faire. Si vous êtes l'utilisateur unique de votre machine, vous pouvez effectuer l'installation vous-même.

Décompressez le fichier `Unitex_1.2.zip` (vous pouvez télécharger ce fichier à l'adresse suivante: <http://www-igm.univ-mlv.fr/~unitex>) dans un répertoire `Unitex` que vous aurez préalablement créé, de préférence dans `Program Files`. Après la décompression, le répertoire `Unitex` contient plusieurs sous-répertoires dont un nommé `App`. Ce dernier répertoire contient un fichier nommé `Unitex.jar`. Ce fichier est l'exécutable Java qui lance l'interface graphique. Il vous suffit de double-cliquer dessus pour lancer le programme. Pour faciliter le lancement du programme, il est conseillé de créer un raccourci vers ce fichier sur le bureau.

1.4 Installation sous Linux et MacOS

Pour installer Unitex sous Linux et MacOS, il est recommandé d'être administrateur système. Décompressez le fichier `Unitex_1.2.zip` dans un répertoire nommé `Unitex`, au moyen de la commande suivante:

```
unzip Unitex_1.2.zip -d Unitex
```

Placez-vous ensuite dans le répertoire `Unitex/Src/C++`, et lancez la compilation des programmes au moyen de la commande:

```
make install
```

Créez ensuite un alias sur le modèle suivant:

```
alias unitex='cd /.../Unitex/App/ ; java -jar Unitex.jar'
```

1.5 Première utilisation

Si vous travaillez sous Windows, le programme vous demandera de choisir un répertoire personnel de travail, que vous pourrez changer ultérieurement. Pour créer un répertoire, cliquez sur l'icône représentant un dossier (voir figure 1.3).

Sous Linux et MacOS, le programme créera automatiquement un répertoire `/unitex` dans votre répertoire `$HOME`. Ce répertoire vous permettra de stocker vos données personnelles. Pour chaque langue que vous utiliserez, le programme copiera l'arborescence de la langue dans votre répertoire personnel, à l'exception des dictionnaires. Vous pourrez ainsi modifier à votre guise votre copie des données sans risquer d'endommager les données du système.

1.6 Ajout de nouvelles langues

Il y a deux manières d'ajouter des langues. Si vous désirez ajouter une nouvelle langue accessible à tous les utilisateurs, il vous faut copier le répertoire correspondant à cette langue

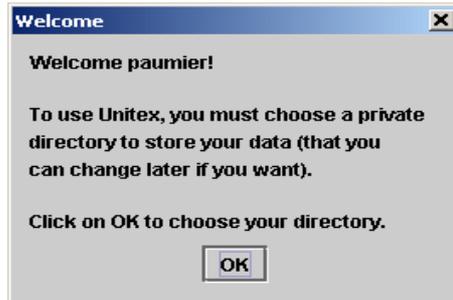


FIG. 1.1 – Première utilisation sous Windows

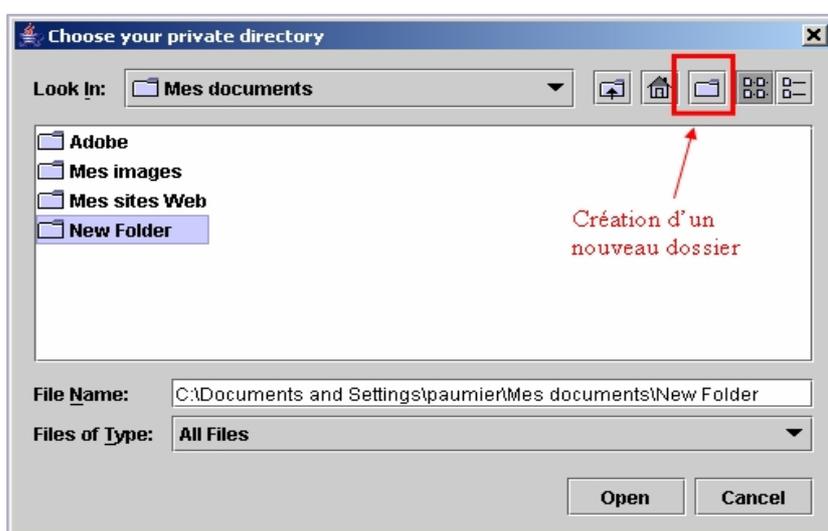


FIG. 1.2 – Première utilisation sous Linux

dans le répertoire **Unitex** du système, ce qui nécessite d'avoir les droits d'accès à ce répertoire (il vous faudra peut-être demander à votre administrateur système de le faire). En revanche, si la langue ne concerne qu'un seul utilisateur, celui-ci peut copier le répertoire en question dans son répertoire personnel. Il pourra ainsi travailler sur cette langue, sans qu'elle soit proposée aux autres utilisateurs.

1.7 Désinstallation

Quelque soit le système sous lequel vous travaillez, il vous suffit de supprimer le répertoire **Unitex** pour effacer tous les fichiers du système. Sous Windows, vous devrez ensuite supprimer le raccourci vers **Unitex.jar** si vous en avez créé un; même chose sous Linux ou MacOS si vous avez créé un alias.

FIG. 1.3 – *Création du dossier personnel*

Chapitre 2

Chargement d'un texte

Une des principales fonctionnalités d'Unitex est la recherche d'expressions dans des textes. Pour cela, les textes doivent subir plusieurs opérations de prétraitement telles que la normalisation de formes non ambiguës et le découpage du texte en phrases. Une fois ces opérations effectuées, des dictionnaires électroniques sont appliqués aux textes. On peut alors effectuer des recherches sur ces textes en leur appliquant des grammaires.

Ce chapitre décrit les différentes étapes du prétraitement des textes.

2.1 Sélection de la langue

Lors du lancement d'Unitex, le programme vous demande de choisir la langue dans laquelle vous allez travailler (voir figure 2.1). Les langues proposées sont celles qui sont présentes dans le répertoire système **Unitex** ainsi que celles éventuellement installées dans votre répertoire personnel. Si vous utilisez une langue pour la première fois, Unitex recopie le répertoire système de cette langue dans votre répertoire personnel, à l'exception des dictionnaires. Le choix de la langue permet d'indiquer à Unitex où trouver certaines données, comme par exemple le fichier `alphabet`. Vous pouvez à tout moment changer de langue en cliquant sur "Change Language..." dans le menu "Text". Si vous changez de langue, le programme fermera, s'il y en a, toutes les fenêtres relatives au texte courant. La langue courante est indiquée sur la barre de titre de l'interface graphique.

2.2 Format des textes

Unitex manipule des textes Unicode. Unicode est un standard qui décrit un codage universel des caractères. Chaque caractère se voit attribuer un numéro unique, ce qui permet de représenter des textes sans avoir à tenir compte des codages propres aux différentes machines et/ou systèmes d'exploitation. Unitex utilise une représentation codée sur deux octets du standard Unicode 3.0, appelée Unicode Little-Endian (pour plus de détails, voir [49]).

Les textes fournis avec Unitex sont déjà au format Unicode. Si vous essayez d'ouvrir un texte qui n'est pas au format Unicode, le programme vous proposera de le convertir automatiquement (voir figure 2.2). Cette conversion se base sur la langue courante: si vous travaillez en



FIG. 2.1 – Sélection de la langue au lancement d'Unitex

français, Unitex vous proposera de convertir votre texte ¹ en supposant qu'il est codé avec une page de codes française. Par défaut, Unitex vous propose soit de remplacer le texte original, soit de renommer le fichier d'origine en insérant `.old` avant son extension. Par exemple, si l'on dispose d'un fichier ASCII nommé `balzac.txt`, le processus de conversion va créer une copie de ce fichier ASCII nommée `balzac.old.txt`, et va remplacer le contenu de `balzac.txt` par son équivalent en Unicode.



FIG. 2.2 – Conversion automatique d'un texte non Unicode

Si le codage proposé par défaut n'est pas le bon, ou si vous voulez renommer le fichier autrement qu'avec le suffixe `.old`, vous pouvez utiliser la commande "Transcode Files" dans le menu "File Edition". Cette commande vous permet de choisir les codages d'origine et de destination des documents à convertir (voir figure 2.3). Par défaut, le codage source proposé est celui qui correspond à la langue courante, et le codage de destination est Unicode Little-Endian. Vous pouvez modifier ces choix, en sélectionnant n'importe quels codages de source

1. Unitex propose également de convertir automatiquement les graphes et dictionnaires qui ne sont pas en Unicode Little-Endian.

et destination. Ainsi, vous pouvez si vous le souhaitez convertir vos données dans d'autres codages, comme par exemple UTF-8 si vous voulez en faire des pages web. Le bouton "Add Files" vous permet de sélectionner les fichiers à convertir. Le bouton "Remove Files" permet de retirer de la liste des fichiers sélectionnés par erreur. Le bouton "Transcode" lancera la conversion de tous les fichiers. Si une erreur survient lors du traitement d'un fichier (par exemple, un fichier qui serait déjà en Unicode), le traitement continue avec le fichier suivant.

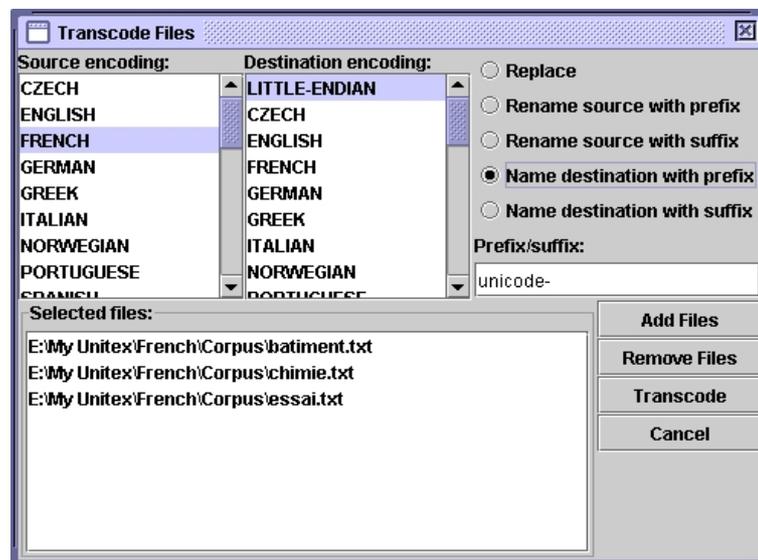


FIG. 2.3 – Conversion de fichiers

Pour obtenir du texte au bon format, vous pouvez également utiliser un traitement de texte comme le logiciel libre OpenOffice.org ([47]) ou Microsoft Word, et sauvegarder votre document au format "Texte unicode".

Par défaut, le codage proposé sur un PC est toujours Unicode Little-Endian. Les textes ainsi obtenus ne contiennent plus d'informations de formatage (police, couleurs, etc.) et sont prêts à être utilisés avec Unitex.

2.3 Edition de textes

Vous avez également la possibilité d'utiliser l'éditeur de texte intégré à Unitex, accessible via la commande "Open..." du menu "File Edition". Cet éditeur vous propose des fonctionnalités de recherche et remplacement propres aux textes et dictionnaires manipulés par Unitex. Pour y accéder, cliquez sur l'icône "Find" (jumelles). Vous verrez alors apparaître une fenêtre divisée en trois onglet. L'onglet "Find" correspond aux opérations de recherche habituelles. Si vous ouvrez un texte découpé en phrases, vous aurez la possibilité de faire une recherche par numéro de phrase dans l'onglet "Find Sentence". Enfin, l'onglet "Dictionary Search", visible sur la figure 2.5, vous permet d'effectuer des opérations propres aux dictionnaires électroniques. En particulier, vous pouvez effectuer une recherche en spécifiant si elle doit porter sur la forme fléchie, le lemme, les codes grammaticaux et sémantiques et/ou les codes flexionnels.

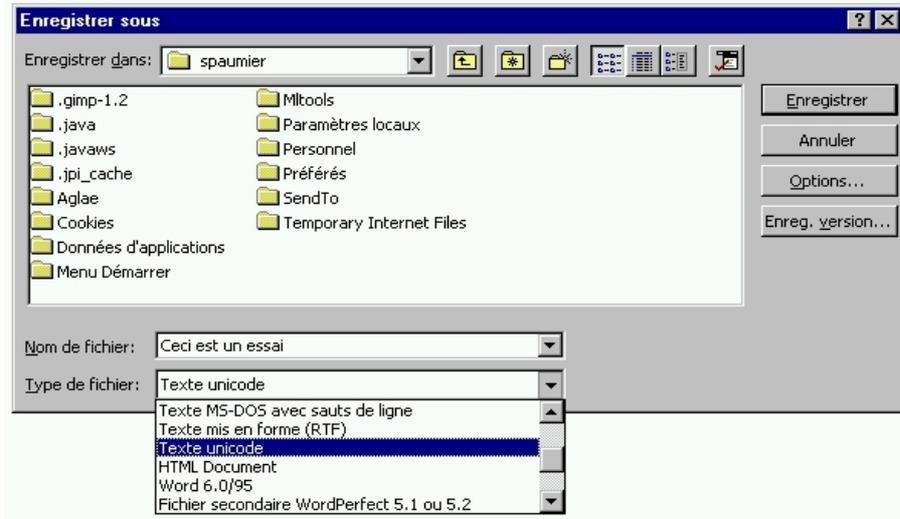
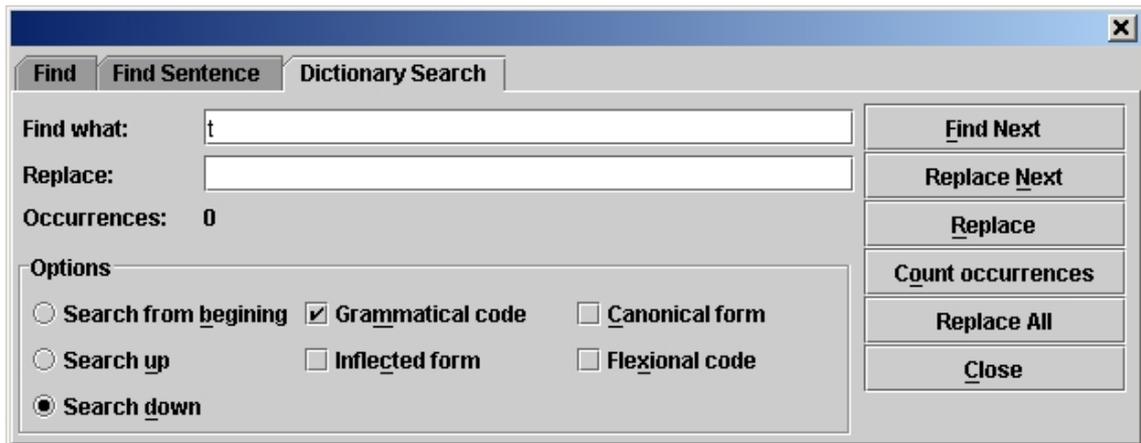


FIG. 2.4 – Sauvegarde en Unicode avec Microsoft Word

Ainsi, si vous voulez rechercher tous les verbes qui ont le trait sémantique t , marquant la transitivité, il vous suffit de chercher t en cochant "Grammatical code". Vous obtiendrait ainsi les entrées voulues, sans ambiguïtés avec toutes les autres occurrences de la lettre t .

FIG. 2.5 – Recherche du trait sémantique t dans un dictionnaire électronique

2.4 Ouverture d'un texte

Unitex propose d'ouvrir deux types de fichiers texte. Les fichiers portant l'extension `.snt` sont des fichiers textes prétraités par Unitex qui sont prêts à être manipulés par les différentes fonctions du système. Les fichiers portant l'extension `.txt` sont des fichiers textes bruts. Pour

utiliser un texte, il faut donc commencer par ouvrir le fichier `.txt` correspondant en cliquant sur "Open..." dans le menu "Text".



FIG. 2.6 – *Menu Text*

Choisissez le type de fichier "Raw Unicode Texts", et sélectionnez votre texte.

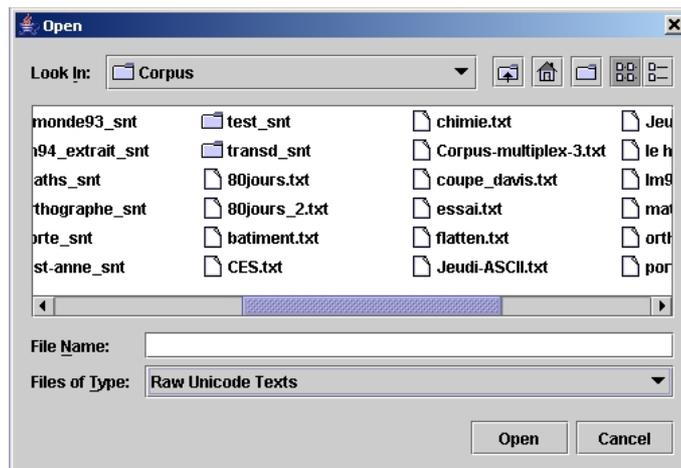


FIG. 2.7 – *Ouverture d'un texte Unicode*

Les fichiers texte dépassant 5 méga-octets ne sont pas affichés; le message "This file is too large to be displayed. Use a wordprocessor to view it." s'affiche dans la fenêtre. Cette remarque concerne tous les fichiers texte (liste des unités lexicales, dictionnaires, etc.). Pour modifier cette limite, allez dans le menu "Info>Préférences", et modifiez la valeur "Maximum Text File Size" dans l'onglet "Text Presentation" (voir figure 4.7, page 53).

2.5 Prétraitement du texte

Une fois le texte sélectionné, Unitex vous propose de le prétraiter. Le prétraitement du texte consiste à lui appliquer les opérations suivantes: normalisation des séparateurs, découpage en unités lexicales, normalisation de formes non ambiguës, découpage en phrases et application des dictionnaires. Si vous refusez le prétraitement, le texte sera néanmoins normalisé et découpé en unités lexicales, car ces opérations sont indispensables au fonctionnement d'Unitex. Il vous sera toujours possible d'effectuer le prétraitement plus tard, en cliquant sur "Preprocess text..." dans le menu "Text". Si vous acceptez le prétraitement, Unitex vous proposera de le paramétrer grâce à la fenêtre de la figure 2.8.

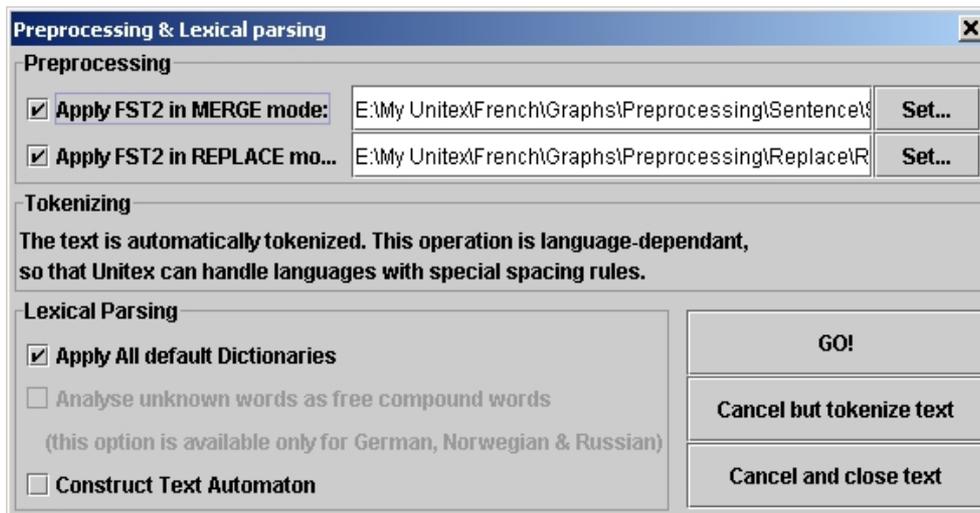


FIG. 2.8 – Fenêtre de prétraitement

L'option "Apply FST2 in MERGE mode" sert à effectuer le découpage du texte en phrases. L'option "Apply FST2 in REPLACE mode" est utilisée pour effectuer des remplacements dans le texte, le plus souvent des normalisations de formes non ambiguës. L'option "Apply All default Dictionaries" permet d'appliquer au texte des dictionnaires au format DELA (Dictionnaires Electroniques du LADL). L'option "Analyse unknown words as free compound words" est utilisée en norvégien pour analyser correctement les mots composés libres formés par soudure de mots simples. Enfin, l'option "Construct Text Automaton" est utilisée pour construire l'automate du texte. Cette option est désactivée par défaut, car elle entraîne une forte consommation de mémoire et d'espace disque si le texte est trop volumineux. La construction de l'automate du texte sera abordée dans le chapitre 7.

NOTE: si vous cliquez sur "Cancel but tokenize text", le programme effectuera malgré tout la normalisation des séparateurs et le découpage en unités lexicales; cliquez sur "Cancel and close text" pour annuler complètement l'opération.

2.5.1 Normalisation des séparateurs

Les séparateurs usuels sont l'espace, la tabulation et le retour à la ligne. On peut rencontrer plusieurs séparateurs consécutifs dans des textes, mais comme cela n'est d'aucune utilité pour une analyse linguistique, on normalise ces séparateurs selon les règles suivantes:

- toute suite de séparateurs contenant au moins un retour à la ligne est remplacée par un unique retour à la ligne;
- toute autre suite de séparateurs est remplacée par un espace.

La distinction entre espace et retour à la ligne est conservée à cette étape car la présence de retours à la ligne peut intervenir dans le découpage du texte en phrases. Le résultat de la normalisation d'un fichier appelé `mon_texte.txt` est un fichier situé dans le même répertoire que le `.txt` et dont le nom est `mon_texte.snt`.

NOTE: lorsque l'on prétraite un texte depuis l'interface graphique, un répertoire nommé `mon_texte_snt` est créé immédiatement après la normalisation. Ce répertoire, appelé répertoire du texte, contiendra toutes les données relatives à ce texte.

2.5.2 Découpage en phrases

Le découpage en phrases est une étape importante du prétraitement car elle va permettre de définir des unités de traitement linguistique. Ce découpage sera utilisé par le programme de construction de l'automate du texte. Contrairement à ce que l'on pourrait penser, la recherche des limites de phrases n'est pas un problème trivial. Considérons le texte suivant:

La famille a appelé le Dr. Martin en urgence.

Le point qui suit *Dr* est suivi d'un mot commençant par une majuscule; il pourrait donc être considéré comme un point de fin de phrase, ce qui serait faux. Afin d'éviter les problèmes de ce genre, dus à des ambiguïtés des symboles de ponctuation, on utilise des grammaires qui décrivent les différents contextes où peuvent apparaître les limites de phrases. La figure 2.9 montre un exemple de grammaire de découpage en phrases.

Lorsqu'un chemin de la grammaire reconnaît une séquence dans le texte et que ce chemin produit le symbole séparateur de phrases `{S}`, on insère ce symbole dans le texte. Ainsi, le chemin le plus haut de la grammaire de la figure 2.9 reconnaît la séquence composée d'un point d'interrogation et d'un mot commençant par une majuscule et insère le symbole `{S}` entre le point d'interrogation et le mot suivant. Le texte suivant:

Quelle heure est-il? Huit heures.

deviendrait donc:

Quelle heure est-il?{S} Huit heures.

Une grammaire de découpage peut manipuler les symboles spéciaux suivants:

- `<E>`: mot vide, ou epsilon. Reconnaît la séquence vide;
- `<MOT>`: reconnaît n'importe quelle suite de lettres;

L'application de cette grammaire à un texte s'effectue grâce au programme `Fst2Txt` en mode `MERGE`. Cela signifie que les sorties produites par la grammaire, en l'occurrence le symbole `{S}`, sont insérées dans le texte. Ce programme prend en entrée un fichier `.snt` et le modifie.

2.5.3 Normalisation de formes non ambiguës

Certaines formes présentes dans les textes peuvent être normalisées (par exemple, la séquence française *l'on* est équivalente à la forme *on*). Chaque utilisateur peut donc vouloir effectuer des remplacements en fonction de ses besoins. Toutefois, il faut faire attention à ce que les formes normalisées soient non ambiguës, ou à ce que la disparition de l'ambiguïté soit sans conséquence pour l'application recherchée. Si l'on décide de remplacer la forme *audit* par *à le-dit*, la phrase:

La cour a procédé à un audit des comptes de cette société.

sera remplacée par la phrase incorrecte:

La cour a procédé à un à le-dit des comptes de cette société.

Il faut donc être très prudent lorsque l'on manipule la grammaire de normalisation.

Il faut également faire attention aux espaces. En effet, si l'on remplace *c'* par *ce* non suivi par un espace, la phrase:

Est-ce que c'était toi?

sera remplacée par la séquence incorrecte:

Est-ce que ceétait toi?

Les symboles acceptés par les grammaires de normalisation sont les mêmes que ceux autorisés dans les grammaires de découpage en phrases. La grammaire utilisée se nomme `Replace.fst2` et se trouve dans le répertoire suivant:

```
/(répertoire personnel)/(langue courante)/Graphs/Preprocessing/Replace
```

Comme pour le découpage en phrases, cette grammaire est utilisée avec le programme `Fst2Txt`, mais cette fois en mode `REPLACE`, ce qui signifie que les entrées reconnues par la grammaire sont remplacées par les séquences produites par celle-ci. On peut voir sur la figure 2.10 une grammaire qui résoud certaines élisions en français.

2.5.4 Découpage du texte en unités lexicales

Certaines langues, en particulier les langues asiatiques, utilisent les séparateurs de façon différente des langues occidentales; les espaces peuvent être interdits, facultatifs ou obligatoires. Pour pouvoir gérer ces particularités au mieux, `Unitex` découpe les textes d'une manière

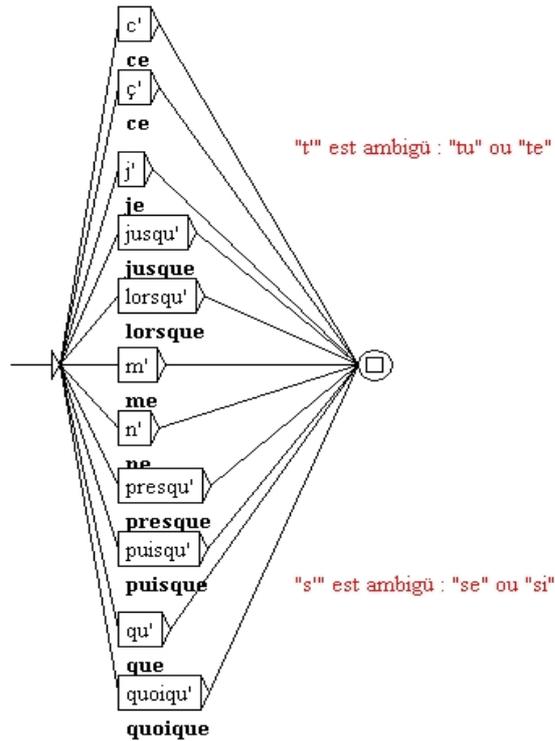


FIG. 2.10 – Grammaire de normalisation de quelques élisions du français

dépendante de la langue. Ainsi, les langues comme le français sont traitées selon le principe suivant:

Une unité lexicale peut être:

- soit le séparateur de phrases {S};
- soit une étiquette lexicale {aujourd'hui, .ADV};
- soit une suite contiguë de lettres (les lettres étant définies par le fichier alphabet de la langue);
- soit un caractère qui n'est pas une lettre; s'il s'agit d'un retour à la ligne, il est remplacé par un espace.

Pour les autres langues, le découpage est effectué caractère par caractère, à l'exception du séparateur de phrases {S} et des étiquettes lexicales. Ce découpage basique garantit le fonctionnement d'Unitex, mais limite l'optimisation des opérations de recherche de motifs. Quelque soit le mode de découpage, les retours à la ligne présents dans un texte sont remplacés par des espaces. Ce découpage est effectué par le programme `Tokenize`. Ce programme produit plusieurs fichiers, stockés dans le répertoire du texte:

- `tokens.txt` contient la liste des unités lexicales dans l'ordre où elles ont été trouvées dans le texte;

- `text.cod` contient un tableau d'entiers; chaque entier correspondant à l'indice d'une unité lexicale dans le fichier `tokens.txt`;
- `tok_by_freq.txt` contient la liste des unités lexicales triée par ordre de fréquence;
- `tok_by_alph.txt` contient la liste des unités lexicales triée par ordre alphabétique;
- `stats.n` contient quelques statistiques sur le texte.

Le découpage du texte:

Un sou c'est un sou.

donne la liste d'unités lexicales suivantes: *Un* ESPACE *sou* c ' *est* un .

On peut remarquer qu'il est tenu compte de la casse (*Un* et *un* sont deux unités distinctes), mais que chaque unité n'est codée qu'une fois. En numérotant ces unités de 0 à 7, ce texte peut être représenté par la séquence d'entiers décrite dans le tableau suivant:

Indice	0	1	2	1	3	4	5	1	6	1	2	7
Unité lexicale correspondante	<i>Un</i>		<i>sou</i>		<i>c</i>	'	<i>est</i>		<i>un</i>		<i>sou</i>	.

TAB. 2.1 – Représentation du texte *Un sou c'est un sou.*

Pour plus de détails, voir le chapitre 10.

2.5.5 Application de dictionnaires

L'application de dictionnaires consiste à construire le sous-ensemble des dictionnaires ne contenant que les formes présentes dans le texte. Ainsi, le résultat de l'application des dictionnaires du français au texte *Igor mange une pomme de terre* produit le dictionnaire de mots simples suivant:

```
de, .DET+z1
de, .PREP+z1
de, .XI+z1
mange, manger. V+z1:P1s:P3s:S1s:S3s:Y2s
pomme, .A+z1:ms:fs:mp:fp
pomme, .N+z1:fs
pomme, pommer. V+z3:P1s:P3s:S1s:S3s:Y2s
terre, .N+z1:fs
terre, terrer. V+z1:P1s:P3s:S1s:S3s:Y2s
une, .N+z1:fs
une, un. DET+z1:fs
```

ainsi que le dictionnaire de mots composés contenant l'unique entrée:

```
pomme de terre, .N+z1:fs
```

Count	Token
82295	
8435	,
5772	the
3500	of
3161	"
2584	and
2454	.
2374	to
2343	(S)
1710	-
1578	a
1340	his
1172	in
802	with
792	I
786	which
771	he
744	that
744	was
738	as
714	;
591	-
563	by

FIG. 2.11 – Unités lexicales d'un texte anglais triées par fréquence

La séquence *Igor* n'étant ni un mot simple du français, ni une partie de mot composé, a été considérée comme mot inconnu. L'application de dictionnaires s'effectue avec le programme *Dico*. Les trois fichiers produits (*dlf* pour les mots simples, *dlc* pour les mots composés et *err* pour les mots inconnus) sont placés dans le répertoire du texte. On appelle dictionnaires du texte les fichiers *dlf* et *dlc*. Une fois l'application des dictionnaires effectuée, Unitex présente par ordre alphabétique les mots simples, composés et inconnus trouvés dans une fenêtre. La figure 2.12 montre les résultats pour un texte français.

Il est également possible d'appliquer des dictionnaires en dehors du prétraitement du texte. Pour cela, il faut cliquer sur "Apply Lexical Resources..." dans le menu "Text". Unitex affiche alors une fenêtre (voir figure 2.13) qui permet de choisir la liste des dictionnaires à appliquer.

La liste "User resources" recense tous les dictionnaires compressés présents dans le répertoire (*langue courante*)/*Dela* de l'utilisateur. Les dictionnaires du système sont listés dans le cadre intitulé "System resources". Utilisez <Ctrl+click> pour sélectionner plusieurs dictionnaires. Le bouton "Set Default" vous permet de définir la sélection courante de dictionnaires

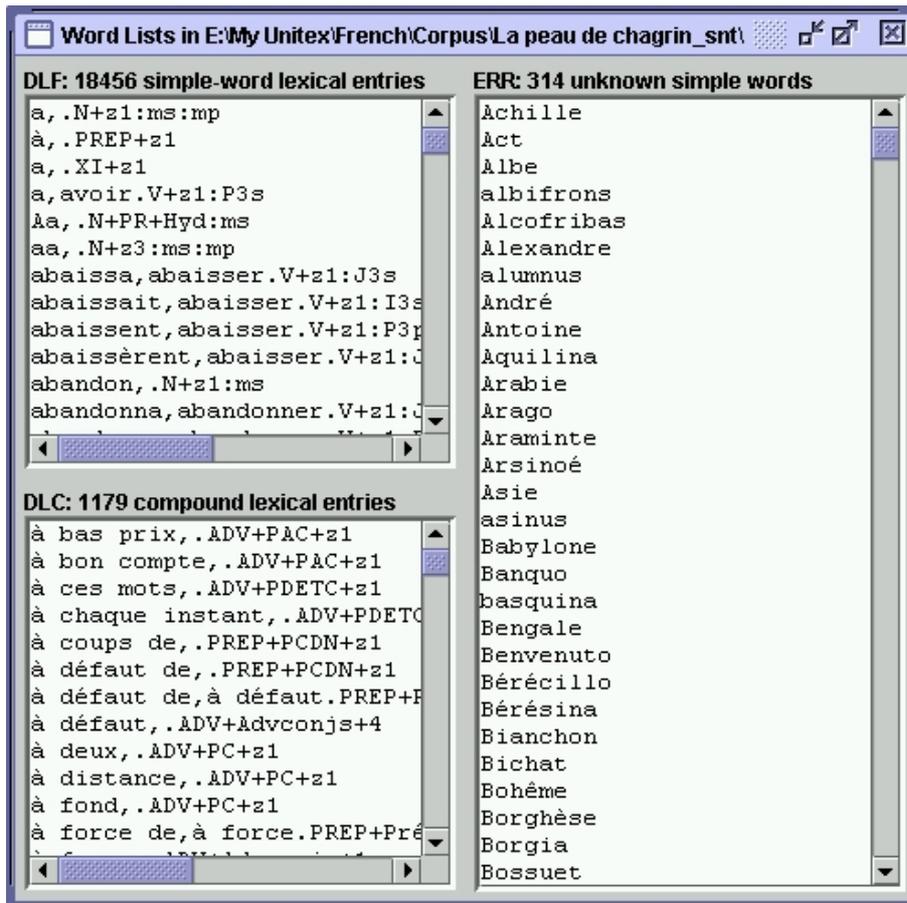


FIG. 2.12 – Résultats de l'application de dictionnaires sur un texte français

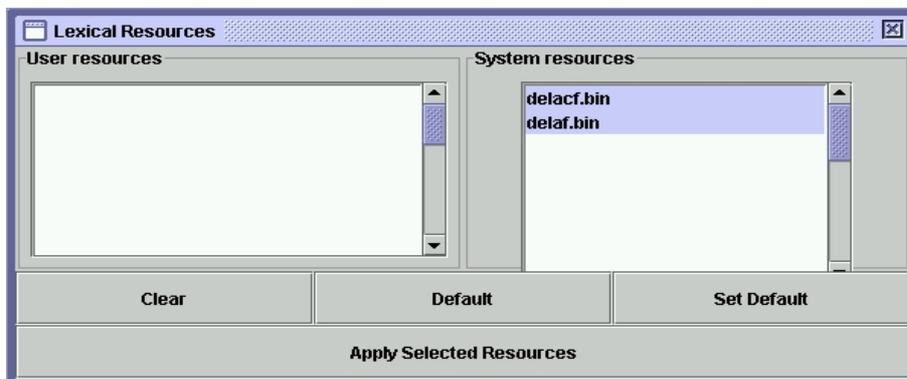


FIG. 2.13 – Paramétrage de l'application des dictionnaires

comme sélection par défaut. Cette sélection par défaut sera utilisée lors du prétraitement si vous choisissez l'option "Apply All default Dictionaries".

2.5.6 Analyse des mots composés libres en allemand, norvégien et russe

Dans certaines langues comme le norvégien, il est possible de former des mots composés libres en soudant leurs éléments. Par exemple, le mot *aftenblad* signifiant *journal du soir* est obtenu en combinant les mots *aften* (*soir*) et *blad* (*journal*). Le programme PolyLex ([42]) explore la liste des mots inconnus après application des dictionnaires au texte et essaye d'analyser chacun de ces mots comme un mot composé. Si un mot possède au moins une analyse, il est retiré de la liste des mots inconnus et les lignes de dictionnaires produites pour ce mot sont ajoutées au dictionnaire des mots simples du texte.

Chapitre 3

Dictionnaires

3.1 Les dictionnaires DELA

Les dictionnaires électroniques utilisés par Unitex utilisent le formalisme des DELA (Dictionnaires Electroniques du LADL). Ce formalisme permet de décrire les entrées lexicales simples et composées d'une langue en leur associant de façon optionnelle des informations grammaticales, sémantiques et flexionnelles. On distingue deux sortes de dictionnaires électroniques. Le type que l'on utilise le plus couramment est le dictionnaire de formes fléchies, appelé DELAF (DELA de formes Fléchies) ou encore DELACF (DELA de formes Composées Fléchies) lorsqu'il s'agit d'un dictionnaire de mots composés. Le second type est le dictionnaire de formes non fléchies appelé DELAS (DELA de formes Simples) ou DELAC (DELA de formes Composées). Les programmes d'Unitex ne font pas de distinction entre les dictionnaires de formes simples et composées. Nous utiliserons donc les termes DELAF et DELAS pour désigner les deux sortes de dictionnaires que leurs entrées soit simples, composées ou mixtes.

3.1.1 Format des DELAF

Syntaxe d'une entrée

Une entrée d'un DELAF est une ligne de texte terminée par un retour à la ligne qui respecte le schéma suivant:

```
mercantiles,mercantile.A+z1:mp:fp/ceci est un exemple
```

Les différents éléments qui forment cette ligne sont les suivants:

- **mercantiles** est la forme fléchie de l'entrée. Cette forme fléchie est obligatoire;
- **mercantile** est la forme canonique de l'entrée. Pour les noms et les adjectifs, il s'agit en général de la forme au masculin singulier; pour les verbes, la forme canonique est l'infinitif. Cette information peut être omise comme dans l'exemple suivant:

```
boîte à merveilles,.N+z1:fs
```

Cela signifie alors que la forme canonique est identique à la forme fléchie. La forme canonique est séparée de la forme fléchie par une virgule;

- **A+z1** est la séquence d'informations grammaticales et sémantiques. Dans notre exemple, **A** désigne un adjectif, et **z1** indique qu'il s'agit d'un mot courant (voir tableau 3.2). Toute entrée doit comporter au moins un code grammatical ou sémantique, séparé de la forme canonique par un point. S'il y a plusieurs codes, ceux-ci doivent être séparés par le caractère +;
- **:mp:fp** est la séquence d'informations flexionnelles. Ces informations décrivent le genre, le nombre, les temps et modes de conjugaisons, les déclinaisons pour les langues à cas, etc. Ces informations sont facultatives. Un code flexionnel est composé d'un ou plusieurs caractères codant chacun une information. Les codes flexionnels doivent être séparés par le caractère:. Dans notre exemple, **m** signifie masculin, **p** pluriel et **f** féminin (voir tableau 3.3). Le caractère **:** s'interprète comme un OU logique. **:mp:fp** signifie donc "masculin pluriel" ou "féminin pluriel". Comme chaque caractère correspond à une information, il est inutile d'utiliser plusieurs fois un même caractère. Ainsi, coder le participe passé avec le code **:PP** serait strictement équivalent à utiliser **:P** seul;
- **/ceci est un exemple** est un commentaire. Les commentaires sont facultatifs et doivent être introduits par le caractère /. Les commentaires sont supprimés lorsque l'on compile les dictionnaires.

REMARQUE IMPORTANTE: Il est possible d'utiliser le point et la virgule dans une entrée de dictionnaire. Pour cela, il faut les déspecialiser avec le caractère \:

```
3\,1415,PI.NOMBRE
Organisation des Nations Unies,O\N\U\..SIGLE
```

ATTENTION: chaque caractère est pris en compte dans une ligne de dictionnaire. Par exemple, si vous introduisez des espaces, ceux-ci seront considérés comme faisant partie intégrante des informations. Dans la ligne suivante:

```
gît,gésir.V+z1:P3s /voir ci-gît
```

l'espace qui précède le caractère / sera considéré comme faisant partie d'un code flexionnel à 4 caractères composé de **P**, **3**, **s** et d'un espace.

Il est possible d'insérer des commentaires dans un dictionnaire DELAF ou DELAS, en faisant débiter la ligne par le caractère /. Exemple:

```
/ L'entrée nominale pour 'par' est un terme de golf
par,.N+z3:ms
```

Mots composés avec espace ou tiret

Certains mots composés comme *grand-mère* peuvent s'écrire avec des espaces ou avec des tirets. Pour éviter de devoir dédoubler toutes les entrées, il est possible d'utiliser le caractère =. Lors de la compression du dictionnaire, le programme **Compress** vérifie pour chaque ligne si la forme fléchie ou la forme canonique contient le caractère = non protégé par le caractère de déspecialisation \. Si c'est le cas, le programme remplace l'entrée par deux entrées: une où le caractère = est remplacé par un espace, et une où il est remplacé par un tiret. Ainsi, l'entrée suivante:

```
grand=mères,grand=mère.N:fp
```

est remplacée par les deux lignes suivantes:

```
grand mères,grand mère.N:fp
grand-mères,grand-mère.N:fp
```

NOTE: si vous souhaitez écrire une entrée contenant le caractère =, déspecialisez-le avec le caractère \ comme dans l'exemple suivant:

```
E\=mc2,.FORMULE
```

Cette opération de remplacement a lieu lors de la compression du dictionnaire. Une fois le dictionnaire comprimé, les signes = déspecialisés sont remplacés par de simples =. Ainsi, si l'on comprime un dictionnaire contenant les lignes suivantes:

```
E\=mc2,.FORMULE
grand=mère,.N:fs
```

et que l'on applique ce dictionnaire au texte:

Ma grand-mère m'a expliqué la formule $E=mc^2$.

on obtiendra les lignes suivantes dans le dictionnaire de mots composés du texte:

```
E=mc2,.FORMULE
grand-mère,.N:fs
```

Factorisation d'entrées

Plusieurs entrées ayant les mêmes formes fléchie et canonique peuvent être regroupées en une seule à condition qu'elle aient les mêmes codes grammaticaux et sémantiques. Cela permet entre autres de regrouper des conjugaisons identiques pour un même verbe:

```
glace,glacer.V+z1:P1s:P3s:S1s:S3s:Y2s
```

Si les informations grammaticales et sémantiques diffèrent, il faut créer des entrées distinctes:

```
glace,.N+z1:fs
glace,glacer.V+z1:P1s:P3s:S1s:S3s:Y2s
```

Certaines entrées ayant les mêmes codes grammaticaux et sémantiques peuvent avoir des sens différents, comme c'est le cas pour le mot *poêle* qui désigne un appareil de chauffage ou un voile au masculin et un instrument de cuisine au féminin. On peut donc distinguer les entrées dans ce cas:

```
poêle,.N+z1:fs/ poêle à frire
poêle,.N+z1:ms/ voile, linceul; appareil de chauffage
```

NOTE: dans la pratique, cette distinction n'a pas d'autre conséquence qu'une augmentation du nombre d'entrées du dictionnaire. Les différents programmes qui composent Unitex donneront exactement les mêmes résultats si l'on fusionne ces entrées en:

```
poêle,.N+z1:fs:ms
```

L'intérêt de cette distinction est donc laissée à l'appréciation des personnes qui construisent des dictionnaires.

3.1.2 Format des DELAS

Le format des DELAS est très similaire à celui des DELAF. La différence est qu'on ne mentionne qu'une forme canonique suivie de codes grammaticaux et/ou sémantiques. La forme canonique est séparée des différents codes par une virgule. Voici un exemple d'entrée:

```
cheval,N4+An1
```

Le premier code grammatical ou sémantique sera interprété par le programme de flexion comme le nom de la grammaire à utiliser pour fléchir l'entrée. L'entrée de l'exemple ci-dessus indique que le mot *cheval* doit être fléchi avec une grammaire nommée N4. Il est possible d'ajouter des codes flexionnels aux entrées, mais la nature de l'opération de flexion limite l'intérêt de cette possibilité. Pour plus de détails, voir plus loin dans ce chapitre la section 3.4.

3.1.3 Contenu des dictionnaires

Les dictionnaires fournis avec Unitex contiennent des descriptions des mots simples et composés. Ces descriptions indiquent la catégorie grammaticale de chaque entrée, ses éventuels codes de flexion, ainsi que des informations sémantiques diverses. Les tableaux suivants donnent un aperçu des différents codes utilisés dans les dictionnaires fournis avec Unitex. Ces codes ont la même signification pour presque toutes les langues, même si certains d'entre eux sont propres à certaines langues (*i.e.* marque du neutre, etc.).

NOTE: les descriptions des temps du tableau 3.3 correspondent au français. Néanmoins, la plupart de ces définitions se retrouvent dans plusieurs langues (infinitif, présent, participe passé, etc.).

Code	Signification	Exemples
A	adjectif	fabuleux
ADV	adverbe	réellement, à la longue
CONJC	conjonction de coordination	mais
CONJS	conjonction de subordination	puisque, à moins que
DET	déterminant	ses, trente-six
INTJ	interjection	adieu, mille millions de mille sabords
N	nom	prairie, vie sociale
PREP	préposition	sans, à la lumière de
PRO	pronom	tu, elle-même
V	verbe	continuer, copier-coller

TAB. 3.1 – Codes grammaticaux usuels

Code	Signification	Exemple
z1	langage courant	blague
z2	langage spécialisé	sépulcre
z3	langage très spécialisé	houer
Abst	abstrait	bon goût
An1	animal	cheval de race
An1Coll	animal collectif	troupeau
Conc	concret	abbaye
ConcColl	concret collectif	décombres
Hum	humain	diplomate
HumColl	humain collectif	vieille garde
t	verbe transitif	foudroyer
i	verbe intransitif	fraterniser
en	particule pré-verbale (PPV) obligatoire	en imposer
se	verbe pronominal	se marier
ne	verbe à négation obligatoire	ne pas cesser de

TAB. 3.2 – Quelques codes sémantiques

Malgré une base commune à la plupart des langues, les dictionnaires contiennent des particularités de codage propres à chaque langue. Ainsi, les codes de déclinaisons variant beaucoup d'une langue à une autre, n'ont pas été décrits ici. Pour une description exhaustive de tous les codes utilisés dans un dictionnaire, nous vous recommandons de vous adresser directement à l'auteur du dictionnaire.

Toutefois, ces codes ne sont absolument pas limitatifs. Chaque utilisateur peut introduire ces propres codes, et créer ses propres dictionnaires. Par exemple, on pourrait dans un but pédagogique introduire dans les dictionnaires anglais des marques indiquant les faux-amis français:

bless, .V+faux-ami/bénir
 cask, .N+faux-ami/tonneau

Code	Signification
m	masculin
f	féminin
n	neutre
s	singulier
p	pluriel
1, 2, 3	1 ^{ere} , 2 ^{eme} , 3 ^{eme} personne
P	présent de l'indicatif
I	imparfait de l'indicatif
S	présent du subjonctif
T	imparfait du subjonctif
Y	présent de l'impératif
C	présent du conditionnel
J	passé simple
W	infinitif
G	participe présent
K	participe passé
F	futur

TAB. 3.3 – Codes flexionnels usuels

journey, .N+faux-ami/voyage

Il est également possible d'utiliser les dictionnaires pour stocker des informations particulières. Ainsi, on pourrait utiliser la forme fléchie d'une entrée pour décrire un sigle et la forme canonique pour en donner la forme complète:

ADN,Acide DésoxyriboNucléique.SIGLE

LADL,Laboratoire d'Automatique Documentaire et Linguistique.SIGLE

SAV,Service Après-Vente.SIGLE

3.2 Vérification du format d'un dictionnaire

Lorsque les dictionnaires sont de taille importante, il devient fastidieux de les vérifier à la main. Unitex contient le programme `CheckDic` qui vérifie automatiquement les dictionnaires DELAF et DELAS.

Ce programme effectue une vérification de la syntaxe des entrées. Pour chaque entrée mal formée, le programme affiche le numéro de ligne, le contenu de cette ligne et la nature de l'erreur. Les résultats de l'analyse sont sauves dans un fichier nommé `CHECK_DIC.TXT` qui est affiché une fois la vérification terminée. En plus des éventuels messages d'erreurs, ce fichier contient la liste de tous les caractères utilisés dans les formes fléchies et canoniques, la liste des codes grammaticaux et sémantiques, ainsi que la liste des codes flexionnels utilisés. La liste des caractères permet de vérifier que les caractères présents dans le dictionnaire sont cohérents avec ceux présents dans le fichier alphabet de la langue. Chaque caractère est suivi par sa valeur en notation hexadécimale. Les listes de codes peuvent être utilisées pour vérifier qu'il n'y a pas de faute de frappe dans les codes du dictionnaire.

Le programme fonctionne avec des dictionnaires non comprimés, c'est-à-dire sous forme de fichiers texte. La convention généralement appliquée est de donner l'extension `.dic` à ces dictionnaires. Pour vérifier le format d'un dictionnaire, il faut tout d'abord l'ouvrir en cliquant sur "Open..." dans le menu "DELA".



FIG. 3.1 – Menu "DELA"

Chargons le dictionnaire de la figure 3.2:



FIG. 3.2 – Exemple de dictionnaire

Pour lancer la vérification automatique, cliquez sur "Check Format..." dans le menu "DELA". la fenêtre de la figure 3.3 apparaît alors:

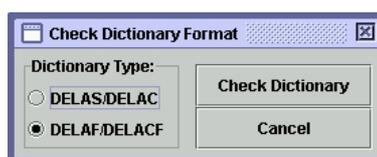


FIG. 3.3 – Vérification automatique d'un dictionnaire

Cette fenêtre vous permet de choisir le type de dictionnaire que vous voulez vérifier. Les résultats de la vérification du dictionnaire de la figure 3.2. sont présentés sur la figure 3.4.

La première erreur est due au fait que le programme n'ait pas trouvé de point. Le seconde, au fait qu'il n'ait pas trouvé de virgule marquant la fin de la forme fléchie. La troisième erreur indique que le programme n'a trouvé aucun code grammatical ou sémantique.

```

Check Results
Line 1: no point found
agreeably,ADV
Line 2: no comma found
agreed.INTJ
Line 4: no grammatical code
ah,.
-----
---- All chars used in forms ----
-----
. (002E)
À (0041)
D (0044)
I (0049)
J (004A)
N (004E)
T (0054)
V (0056)
a (0061)
b (0062)
d (0064)
e (0065)
g (0067)
h (0068)
i (0069)
l (006C)
r (0072)
y (0079)
-----
---- 3 grammatical/semantic codes used in dictionary ----
-----
V
i
N
-----
---- 8 inflectional codes used in dictionary ----
-----
K
I1s
I2s
I3s
I1p
I2p
I3p
s

```

FIG. 3.4 – Résultats d'une vérification automatique

3.3 Tri

Unitex manipule les dictionnaires sans se soucier de l'ordre des entrées. Toutefois, pour des raisons de présentation, il est souvent préférable de trier les dictionnaires. L'opération de tri varie selon plusieurs critères, à commencer par la langue du texte à trier. Ainsi, le tri d'un dictionnaire thaï s'effectue selon un ordre différent de l'ordre alphabétique, si bien qu'Unitex utilise un mode de tri développé spécialement pour le thaï (voir chapitre 9).

Pour les langues européennes, le tri s'effectue généralement selon l'ordre lexicographique, avec toutefois quelques variantes. En effet, certaines langues comme le français considèrent certains caractères comme équivalents. Par exemple, la différence entre les caractères **e** et **é** est ignorée lorsque l'on veut comparer les mots **manger** et **mangés**, car les contextes **r** et **s** permettent de décider de l'ordre. La distinction n'est faite que lorsque les contextes sont identiques, ce qui est le cas si l'on compare **pêche** et **pèche**.

Afin de prendre en compte ce phénomène, le programme de tri `SortTxt` utilise un fichier qui définit des équivalences de caractères. Ce fichier s'appelle `Alphabet_sort.txt` et se trouve

dans le répertoire de la langue courante de l'utilisateur. Voici les premières lignes du fichier utilisé par défaut pour le français:

```
AÃÄÅaãää
Bb
CÇcç
Dd
EÉÊËeéèë
```

Les caractères présents sur une même ligne sont considérés comme équivalents quand le contexte le permet. Lorsqu'il faut comparer deux caractères équivalents, on les compare selon l'ordre dans lequel ils apparaissent de gauche à droite sur la ligne. On peut voir sur l'extrait ci-dessus qu'on ne fait pas de différence entre minuscules et majuscules, et qu'on ignore les accents ainsi que la cédille.

Pour trier un dictionnaire, ouvrez-le, puis cliquez sur "Sort Dictionary" dans le menu "DELA". Par défaut, le programme cherche toujours à utiliser le fichier `Alphabet_sort.txt`. Si ce fichier est absent, le tri se fait selon l'indice des caractères dans le codage Unicode. En modifiant ce fichier, vous pouvez définir vos propres préférences de tri.

Remarque: après l'application des dictionnaires sur un texte, les fichiers `dlf`, `dlc` et `err` sont automatiquement triés avec ce programme.

3.4 Flexion automatique

Comme décrit dans la section 3.1.2, une ligne de DELAS se compose généralement d'une forme canonique et d'une séquence de codes grammaticaux ou sémantiques:

```
bocal,N4+Conc
cheval,N4+An1
local,N4
```

Le premier code rencontré est interprété comme le nom de la grammaire à utiliser pour fléchir la forme canonique. Les grammaires de flexion doivent avoir été compilées (voir chapitre 5). Dans l'exemple ci-dessus, toutes les entrées seront fléchies avec une grammaire nommée `N4`.

Pour lancer la flexion, cliquez sur "Inflect..." dans le menu "DELA". La fenêtre de la figure 3.5 vous permet d'indiquer au programme de flexion le répertoire dans lequel se trouvent vos grammaires de flexion. Par défaut, le sous-répertoire **Inflection** du répertoire de la langue courante est utilisé. L'option "Add ':' before inflectional codes if necessary" insère automatiquement le caractère ':' avant les codes flexionnels, dans le cas où ceux-ci ne débuteraient pas par ce caractère. L'option "Remove class numbers" permet de remplacer les codes avec numéros utilisés dans le DELAS par des codes sans numéros, prêts à être utilisés. Exemple: `V17` sera remplacé par `V`.

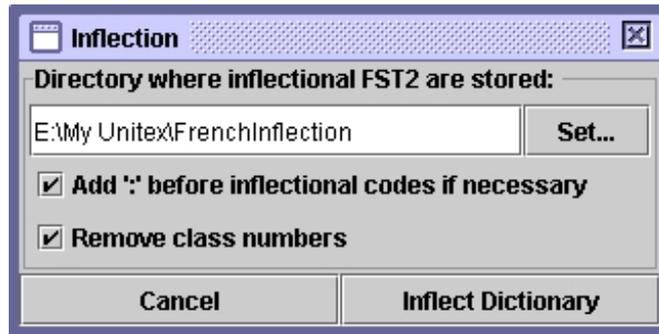
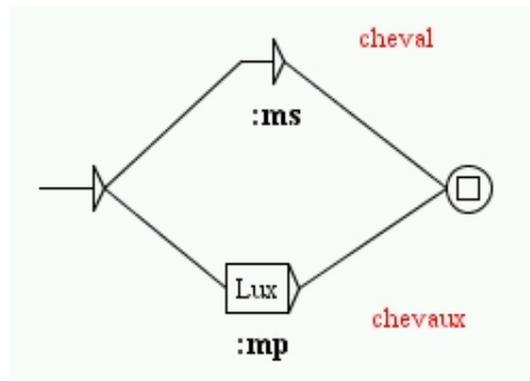


FIG. 3.5 – Configuration de la flexion automatique

FIG. 3.6 – Grammaire de flexion N_4

La figure 3.6 présente un exemple de grammaire de flexion. Les chemins décrivent les suffixes à ajouter ou à retrancher pour obtenir la forme fléchie à partir de la forme canonique, et les sorties (texte en gras sous les boîtes) donnent les codes flexionnels à ajouter à l'entrée du dictionnaire. Dans notre exemple, deux chemins sont possibles. Le premier ne modifie pas la forme canonique et ajoute le code flexionnel `:ms`. Le second retranche une lettre grâce à l'opérateur `L`, ajoute ensuite le suffixe `ux` et ajoute le code flexionnel `:mp`. Deux opérateurs sont possibles:

- `L` (left) enlève une lettre à l'entrée;
- `R` (right) rétablit une lettre de l'entrée. En français, beaucoup de verbes du premier groupe se conjuguent au présent à la troisième personne du singulier en retirant le `r` de l'infinitif et en changeant la 4^{ème} lettre en partant de la fin en `è`: `peler` → `pèle`, `acheter` → `achète`, `gérer` → `gère`, etc. Plutôt que d'écrire un suffixe de flexion pour chaque verbe (`LLLLèle`, `LLLLète` et `LLLLère`), on peut utiliser l'opérateur `R` pour n'en écrire qu'un seul: `LLLLèRR`.
- `C` (copy) duplique une lettre de l'entrée, en décalant tout ce qui se trouve à sa droite. Supposons par exemple que l'on souhaite générer automatiquement des adjectifs en `able` à partir de noms. Dans des cas comme `regrettable` ou `réquisitionnable`, on observe

un doublement de la consonne finale du nom. Pour éviter d'écrire un graphe de flexion pour chaque consonne finale possible, on peut utiliser l'opérateur **C** afin de dupliquer la consonne finale, quelle qu'elle soit.

Le programme de flexion **Inflect** explore tous les chemins de la grammaire de flexion en engendrant toutes les formes fléchies possibles. Afin d'éviter de devoir remplacer les noms des grammaires de flexion par de vrais codes grammaticaux dans le dictionnaire obtenu, le programme remplace ces noms par leurs plus longs préfixes composés de lettres. Ainsi, **N4** est remplacé par **N**. En choisissant judicieusement les noms des grammaires de flexion, on peut donc engendrer directement un dictionnaire prêt à l'emploi.

Voici le dictionnaire obtenu après flexion du DELAS de notre exemple:



```

E:\My Unitex\French\Delas\delasflx.dic
bocaux,bocal.N+Conc:mp
bocal,bocal.N+Conc:ms
chevaux,cheval.N+Anl:mp
cheval,cheval.N+Anl:ms
locaux,local.N:mp
local,local.N:ms

```

FIG. 3.7 – *Résultat de la flexion automatique*

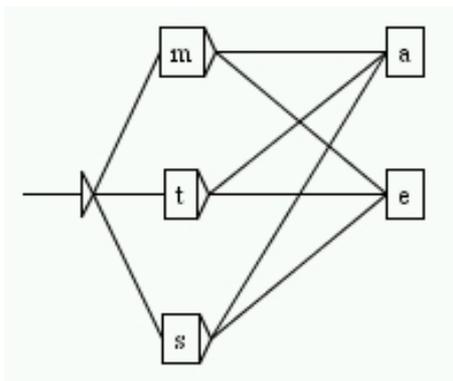
3.5 Compression

Unitex applique aux textes des dictionnaires comprimés. La compression permet de réduire la taille des dictionnaires et d'en accélérer la consultation. Cette opération s'effectue avec le programme **Compress**. Celui-ci prend en entrée un dictionnaire sous forme de fichier texte (par exemple `mon_dico.dic`) et produit deux fichiers:

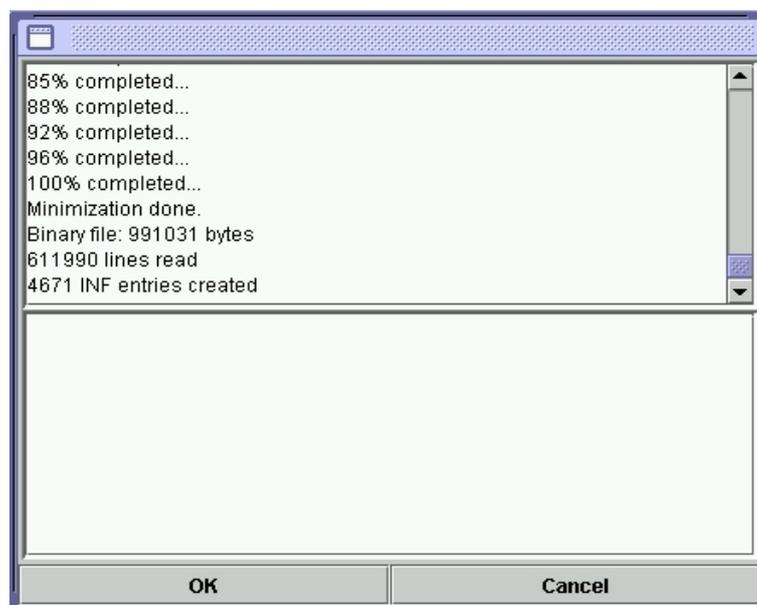
- `mon_dico.bin` contient l'automate minimal des formes fléchies du dictionnaires;
- `mon_dico.inf` contient des codes qui permettent de reconstruire le dictionnaire d'origine à partir des formes fléchies contenues dans `mon_dico.bin`.

L'automate minimal contenu dans `mon_dico.bin` est une représentation des formes fléchies où tous les préfixes et suffixes communs sont factorisés. Par exemple, l'automate minimal des mots `me`, `te`, `se`, `ma`, `ta` et `sa` peut être représenté par le graphe de la figure 3.8.

Pour comprimer un dictionnaire, ouvrez-le puis cliquez sur "Compress into FST" dans le menu "DELA". La compression est indépendante de la langue et du contenu du dictionnaire. Les messages produits par le programme sont affichés dans une fenêtre qui ne se ferme pas

FIG. 3.8 – *Représentation d'un exemple d'automate minimal*

automatiquement. Vous pouvez ainsi voir la taille du fichier `.bin` obtenu, le nombre de lignes lues ainsi que le nombre de codes flexionnels produits. La figure 3.9 montre le résultat de la compression d'un dictionnaire de mots simples:

FIG. 3.9 – *Résultat d'une compression*

À titre indicatif, les taux de compression généralement observés sont d'environ 95% pour les dictionnaires de mots simples et 50% pour ceux de mots composés.

3.6 Application de dictionnaires

Les dictionnaires peuvent être appliqués soit lors du prétraitement, soit explicitement en cliquant sur "Apply Lexical Resources..." dans le menu "Text" (voir section 3.6). Nous allons maintenant détailler les règles de l'application de dictionnaires.

3.6.1 Priorités

La règle de priorité est la suivante: si un mot du texte a été trouvé dans un dictionnaire, ce mot ne sera plus pris en compte lors de l'application de dictionnaires ayant une priorité inférieure.

Cela permet d'éliminer certaines ambiguïtés lors de l'application des dictionnaires. Par exemple, le mot *par* a une interprétation nominale dans le domaine du golf. Si l'on ne veut pas envisager cet emploi, il suffit de créer un dictionnaire filtre ne contenant que l'entrée `par, .PREP` et de le sauver en lui donnant la priorité la plus haute. De cette manière, même si le dictionnaire des mots simples contient l'autre entrée, celle-ci sera ignorée grâce au jeu des priorités.

Il y a trois niveaux de priorités. Les dictionnaires dont les noms sans extension se terminent par `-` ont la priorité la plus grande; ceux dont le nom se termine par `+` ont la priorité la plus faible; les autres dictionnaires sont appliqués avec une priorité moyenne. L'ordre d'application de plusieurs dictionnaires ayant la même priorité est sans importance. En ligne de commande, l'instruction:

```
Dico ex.snt alph.txt Pays+.bin Villes-.bin Fleuves.bin Regions-.bin
```

appliquerait donc les dictionnaires dans l'ordre suivant (`ex.snt` est le texte auquel sont appliqués les dictionnaires, et `alph.txt` est le fichier alphabet utilisé):

1. `Villes-.bin`
2. `Regions-.bin`
3. `Fleuves.bin`
4. `Pays+.bin`

3.6.2 Règles d'application des dictionnaires

Outre la règle de priorités, l'application des dictionnaires s'effectue en respectant les majuscules et les espaces. La règle du respect des majuscules est la suivante:

- s'il y a une majuscule dans le dictionnaire, alors il doit y avoir une majuscule dans le texte;
- s'il y a une minuscule dans le dictionnaire, il peut y avoir soit une minuscule soit une majuscule dans le texte.

Ainsi, l'entrée `pierre, .N:fs` reconnaîtra les mots `pierre`, `Pierre` et `PIERRE`, alors que `Pierre, .N+Prénom` ne reconnaîtra que `Pierre` et `PIERRE`. Les lettres minuscules et majuscules sont définies par le fichier `alphabet` passé en paramètre au programme `Dico`.

Le respect des espacements est une règle très simple: pour qu'une séquence du texte soit reconnue par une entrée de dictionnaire, elle doit avoir exactement les mêmes espaces. Par exemple, si le dictionnaire contient `aujourd'hui, .ADV`, la séquence `Aujourd' hui` ne sera pas reconnue à cause de l'espace qui suit l'apostrophe.

3.7 Bibliographie

Le tableau 3.4 donne quelques références relatives aux dictionnaires électroniques de mots simples et composés. Pour plus de détails, consultez la page de références sur le site web d'Unitex (<http://www-igm.univ-mlv.fr/~unitex>).

Langue	Mots simples	Mots composés
anglais	[27], [39]	[10], [44]
français	[12], [13], [31]	[13], [23], [45], [25]
grec moderne	[1], [11], [28]	[29], [30]
italien	[17], [18]	[51]
espagnol	[4]	[3]

TAB. 3.4 – *Quelques références bibliographiques sur les dictionnaires électroniques*

Chapitre 4

Recherche d'expressions rationnelles

Nous allons voir dans ce chapitre comment rechercher des motifs simples dans un texte au moyen des expressions rationnelles.

4.1 Définition

Le but de ce chapitre n'est pas de faire une introduction aux langages formels, mais de montrer comment utiliser les expressions rationnelles dans Unitex pour rechercher des motifs simples. Le lecteur intéressé par une présentation plus formelle pourra se reporter aux nombreux ouvrages qui traitent du sujet.

Une expression rationnelle peut-être:

- une unité lexicale (`livre`) ou un motif (`<manger.V>`);
- la concaténation de deux expressions rationnelles (`je mange`);
- l'union de deux expressions rationnelles (`Pierre+Paul`);
- l'étoile de Kleene d'une expression rationnelle (`très*`).

4.2 Unités lexicales

Dans une expression rationnelle, une unité lexicale est une suite de caractères. Les symboles point, plus, étoile, inférieur ainsi que les parenthèses ouvrantes et fermantes ont une signification particulière, il faut donc les déspecialiser avec le caractère `\` si l'on souhaite les rechercher. Voici quelques exemples d'unités lexicales valides:

```
chat
0\.N\.U\.
\ (1984\ )
{S}
```

Par défaut, Unitex tolère que des motifs avec des minuscules reconnaissent des mots écrits avec des majuscules. Il est possible de forcer le respect de la casse en utilisant les guillemets. Ainsi, `"pierre"` ne reconnaît que la forme `pierre` et non pas `Pierre` ou `PIERRE`.

NOTE: si l'on souhaite rendre la présence d'un espace obligatoire, il faut le mettre entre guillemets.

4.3 Motifs

4.3.1 Symboles spéciaux

Il y a deux sortes de motifs. La première catégorie regroupe tous les symboles présentés à la section 2.5.2, à l'exception du symbole `<^>` qui reconnaît un retour à ligne. Tous les retours à la ligne ayant été remplacés par des espaces, ce symbole n'a plus aucune utilité lors de la recherche de motifs. Ces symboles, également appelés *métas*, sont les suivants:

- `<E>` : mot vide, ou epsilon. Reconnaît la séquence vide;
- `<TOKEN>` : reconnaît n'importe quelle unité lexicale;
- `<MOT>` : reconnaît n'importe quelle unité lexicale formée de lettres;
- `<MIN>` : reconnaît n'importe quelle unité lexicale formée de lettres minuscules;
- `<MAJ>` : reconnaît n'importe quelle unité lexicale formée de lettres majuscules;
- `<PRE>` : reconnaît n'importe quelle unité lexicale formée de lettres et commençant par une majuscule;
- `<DIC>` : reconnaît n'importe quel mot figurant dans les dictionnaires du texte;
- `<SDIC>` : reconnaît n'importe quel mot simple figurant dans les dictionnaires du texte;
- `<CDIC>` : reconnaît n'importe quel mot composé figurant dans les dictionnaires du texte;
- `<NB>` : reconnaît n'importe quelle suite de chiffres contigus (1234 est reconnu mais pas 1 234);
- `#` : interdit la présence de l'espace.

4.3.2 Références aux dictionnaires

La seconde sorte de motifs regroupe ceux qui font appel aux informations contenues dans les dictionnaires du texte. Les quatre formes possibles sont:

- `<lire>`: reconnaît toutes les entrées qui ont **lire** comme forme canonique;
- `<lire.V>`: reconnaît toutes les entrées qui ont **lire** comme forme canonique et qui ont le code grammatical **V**;
- `<V>`: reconnaît toutes les entrées qui ont le code grammatical **V**;
- `{lirons,lire.V}` ou `<lirons,lire.V>`: reconnaît toutes les entrées qui ont **lirons** comme forme fléchie, **lire** comme forme canonique et qui ont le code grammatical **V**. Ce type de motif n'a d'intérêt que si l'on travaille sur l'automate du texte où sont explicitées les ambiguïtés des mots. Lorsque l'on effectue une recherche sur le texte, ce motif reconnaît la même chose que la simple unité lexicale **lirons**.

4.3.3 Contraintes grammaticales et sémantiques

La référence au dictionnaire (V) des exemples ci-dessus est élémentaire. Il est possible d'exprimer des motifs plus complexes en indiquant plusieurs codes grammaticaux ou sémantiques, séparés par le caractère +. Une entrée de dictionnaire ne sera alors reconnue que si elle possède tous les codes présents dans le motif. Le motif <N+z1> reconnaît ainsi les entrées:

```
broderies, broderie.N+z1:fp
capitales européennes, capitale européenne.N+NA+Conc+HumColl+z1:fp
```

mais pas:

```
Descartes, René Descartes.N+Hum+NPropre:ms
habitué,.A+z1:ms
```

Il est possible d'exclure des codes en les faisant précéder du caractère - au lieu de +. Pour être reconnue, une entrée doit contenir tous les codes autorisés par le motif et aucun des codes interdits. Le motif <A-z3> reconnaît donc tous les adjectifs qui ne possèdent pas le code z3 (voir tableau 3.2). Si l'on souhaite faire référence à un code contenant le caractère -, il faut déspecialiser ce caractère en le faisant précéder du caractère \. Ainsi, le motif <N+faux\ -ami> pourra reconnaître toutes les entrées de dictionnaires contenant les codes N et faux-ami.

L'ordre dans lequel les codes apparaissent dans le motif n'a aucune importance. Les trois motifs suivants sont équivalents:

```
<N-Hum+z1>
<z1+N-Hum>
<-Hum+z1+N>
```

NOTE: il n'est pas possible d'utiliser un motif n'ayant que des codes interdits. <-N> et <-A-z1> sont donc des motifs incorrects.

4.3.4 Contraintes flexionnelles

On peut également spécifier des contraintes portant sur les codes flexionnels. Ces contraintes doivent obligatoirement être précédées par au moins un code grammatical ou sémantique. Elles se présentent comme les codes flexionnels présents dans les dictionnaires.

Voici quelques exemples de motifs utilisant des contraintes flexionnelles:

- <A:m> reconnaît un adjectif au masculin;
- <A:mp:f> reconnaît un adjectif qui est soit au masculin pluriel, soit au féminin;
- <V:2:3> reconnaît un verbe à la 2^{eme} ou 3^{eme} personne; cela exclut tous les temps qui n'ont ni 2^{eme} ni 3^{eme} personne (infinitif, participe passé, et participe présent) ainsi que les temps conjugués à la première personne.

Pour qu'une entrée de dictionnaire E soit reconnue par un motif M , il faut qu'au moins un code flexionnel de E contienne tous les caractères d'un code flexionnel de M . Considérons l'exemple suivant:

```
 $E$ =sépare ,séparer .V+z1:P1s:P3s:S1s:S3s:Y2s
 $M$ =<V:P2s:Y2>
```

Aucun code flexionnel de E ne contient à la fois les caractères P, 2 et s. Cependant, le code Y2s de E contient bien les caractères Y et 2. Le code Y2 est inclus dans au moins un code de E , le motif M reconnaît donc l'entrée E . L'ordre des caractères à l'intérieur d'un code flexionnel est sans importance.

4.3.5 Négation d'un motif

Il est possible de faire la négation d'un motif au moyen du caractère ! placé immédiatement après le caractère <. La négation est possible sur les motifs <MOT>, <MIN>, <MAJ>, <PRE>, <DIC>, ainsi que sur les motifs ne comportant que des codes grammaticaux, sémantiques ou flexionnels (*i.e.* <!V-z3:P3>). Les motifs # et " " sont la négation l'un de l'autre. Le motif <!MOT> peut reconnaître toutes les unités lexicales qui ne sont pas formées de lettres, sauf le séparateur de phrases.

La négation est interprétée d'une façon particulière dans les motifs <!DIC>, <!MIN>, <!MAJ> et <!PRE>. Au lieu de reconnaître toutes les formes qui ne sont pas reconnues par le motif sans la négation, ces motifs ne donnent que des formes qui sont des séquences de lettres. Ainsi, le motif <!DIC> permet d'obtenir les mots inconnus du texte. Ces formes inconnues sont le plus souvent des noms propres, des néologismes et des fautes d'orthographe.

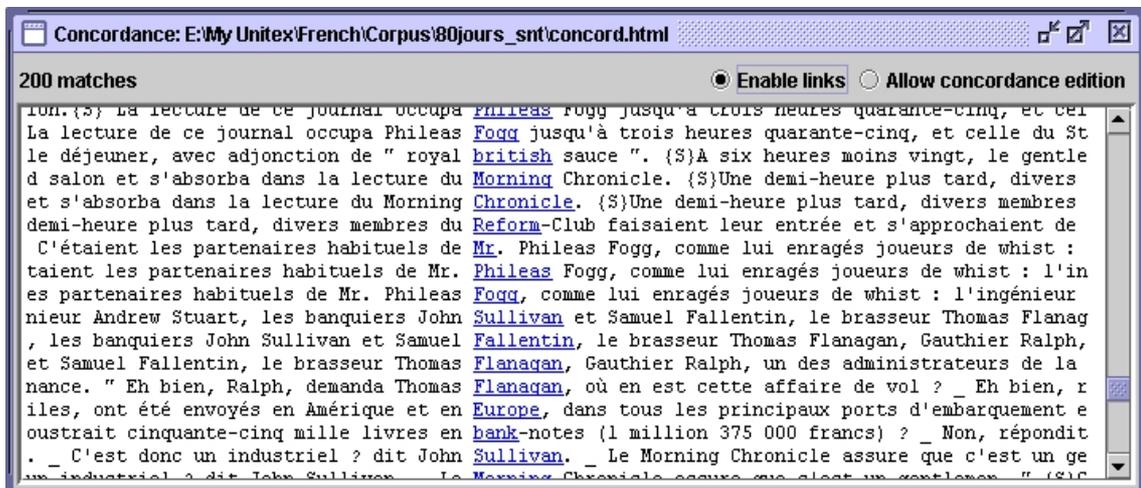


FIG. 4.1 – Résultat de la recherche du motif <!DIC>

Voici plusieurs exemples de motifs mélangeant les différentes sortes de contraintes:

- <A-Hum:fs>: adjectif non humain au féminin singulier;
- <lire.V:P:F>: le verbe *lire* au présent ou au futur;
- <suis,suivre.V>: le mot *suis* en tant que forme conjuguée du verbe *suivre* (par opposition à la forme du verbe *être*);
- <facteur.N-Hum>: toutes les entrées nominales ayant *facteur* comme forme canonique et ne possédant pas le code sémantique Hum;
- <!ADV>: tous les mots qui ne sont pas des adverbes;
- <!MOT>: tous les caractères, qui ne sont pas des lettres, sauf le séparateur de phrases (voir figure 4.2).

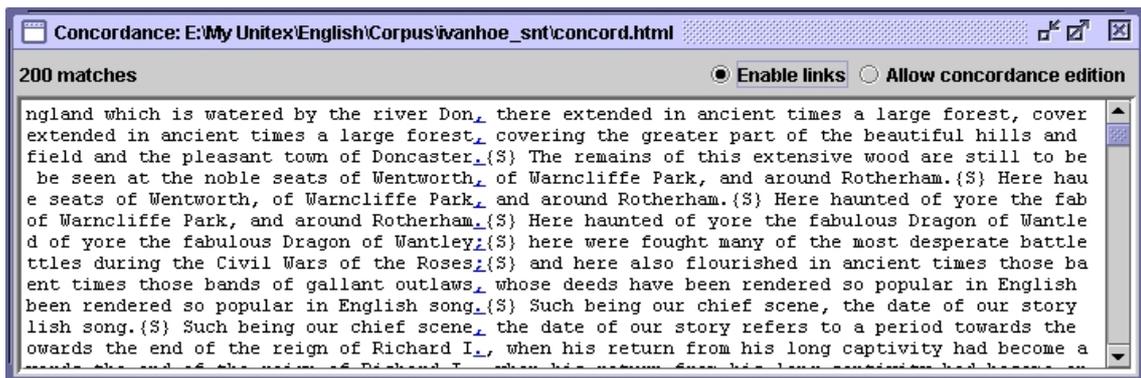


FIG. 4.2 – Résultat de la recherche du motif <!MOT>

4.4 Concaténation

On peut concaténer des expressions rationnelles de trois façons. La première consiste à utiliser l'opérateur de concaténation représenté par le point. Ainsi, l'expression:

<DET>.<N>

reconnaît un déterminant suivi par un nom. L'espace peut également servir à concaténer. L'expression de l'exemple suivant:

le <A> chat

reconnaît l'unité lexicale *le*, suivie d'un adjectif et de l'unité lexicale *chat*. Enfin, il est possible d'omettre le point et l'espace avant une parenthèse ouvrante ou le caractère <, ainsi qu'après une parenthèse fermante ou le caractère >. Les parenthèses servent à délimiter une expression rationnelle. Toutes les expressions suivantes sont équivalentes:

```
le <A> chat
(le <A>)chat
le.<A>.chat
(le)<A>.chat
(le(<A>))(chat)
```

4.5 Union

L'union d'expressions rationnelles se fait en les séparant par le caractère +. L'expression:

`(je+tu+il+elle+on+nous+vous+ils+elles) <V>`

reconnaît un pronom suivi par un verbe. Si l'on veut rendre un élément facultatif dans une expression, il suffit de faire l'union de cet élément avec le mot vide epsilon.

Exemples:

`le(petit+<E>)chat` reconnaît les séquences *le chat* et *le petit chat*

`(<E>+franco-)(anglais+belge)` reconnaît *anglais*, *belge*, *franco-anglais* et *franco-belge*

4.6 Étoile de Kleene

L'étoile de Kleene, représentée par le caractère *, permet de reconnaître zéro, une ou plusieurs occurrences d'une expression. L'étoile doit être placée à droite de l'élément concerné. L'expression:

`il fait très* froid`

reconnaît *il fait froid*, *il fait très froid*, *il fait très très froid*, etc. L'étoile est prioritaire sur les autres opérateurs. Il faut utiliser les parenthèses pour appliquer l'étoile à une expression complexe. L'expression:

`0,(0+1+2+3+4+5+6+7+8+9)*`

reconnaît un zéro, suivie d'une virgule et d'une suite éventuellement vide de chiffres.

ATTENTION: il est interdit de rechercher le mot vide avec une expression rationnelle. Si l'on essaye de chercher `(0+1+2+3+4+5+6+7+8+9)*`, le programme signalera une erreur comme le montre la figure 4.3.

4.7 Filtres morphologiques

Il est possible d'appliquer des filtres morphologiques aux unités lexicales recherchées. Pour cela, il faut faire suivre immédiatement l'unité lexicale considérée par un motif entre double-angles:

motif lexical<<*motif morphologique*>>

Les motifs morphologiques s'expriment sous la forme d'expressions régulières au format POSIX (voir [35] pour une syntaxe détaillée). Voici quelques exemples de filtres élémentaires:

- <<ss>>: contient **ss**
- <<^a>>: commence par **a**

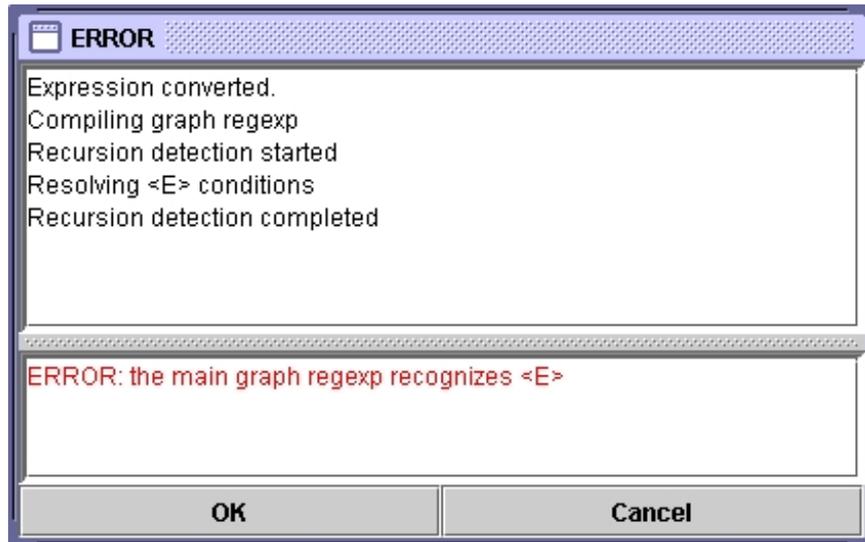


FIG. 4.3 – Erreur lors de la recherche d'une expression reconnaissant le mot vide

- <<ez\$>>: finit par ez
- <<a.s>>: contient a suivi par un caractère quelconque, suivi par s
- <<a.*s>>: contient a suivi par un nombre de caractères quelconque, suivi par s
- <<ss|tt>>: contient ss ou tt
- <<[aeiouy]>>: contient une voyelle non accentuée
- <<[aeiouy]{3,5}>>: contient une séquence de voyelles non accentuées, de longueur comprise entre 3 et 5
- <<ée?>>: contient é suivi par un e facultatif
- <<ss[~e]?>>: contient ss suivi par un caractère qui n'est pas e

Il est possible de combiner ces filtres élémentaires pour former des filtres plus complexes:

- <<[ai]ble\$>>: finit par able ou ible
- <<^(anti|pro)-?>>: commence par anti ou pro, suivi par un tiret facultatif
- <<^([rst][aeiouy]){2,}\$>>: mot formé de 2 ou plus séquences commençant par un r, s ou t suivi d'une voyelle non accentuée
- <<^(~1|1[~e])>>: ne commence pas par l ou alors la deuxième lettre n'est pas un e, c'est-à-dire n'importe quel mot sauf ceux qui commencent par le

Par défaut, un filtre morphologique tout seul est considéré comme s'appliquant au motif <TOKEN>, c'est-à-dire à n'importe quelle unité lexicale. En revanche, lorsqu'un filtre suit immédiatement un motif lexical, il s'applique à ce qui reconnu par le motif lexical. Voici quelques exemples de telles combinaisons:

- <V:K><<i\$>>: participe passé finissant par i
- <CDIC><<->>: mot composé contenant un tiret

- <CDIC><< .* >>: mot composé contenant deux espaces
- <A:fs><<^pro>>: adjectif féminin singulier commençant par **pro**
- <DET><<^([^u] | (u[^n]) | (...+)) >>: déterminant différent de **un**
- <!DIC><<es\$>>: mot qui n'est pas dans le dictionnaire et qui se termine par **es**
- <V:S:T><<uiss>>: verbe au subjonctif passé ou présent, contenant **uiss**

NOTE: par défaut, les filtres morphologiques sont soumis aux mêmes variations de casse que les motifs lexicaux. Ainsi, le motif <<^é>> va reconnaître tous les mots commençant par **é**, mais également ceux qui commencent par **E** ou **É**. Pour forcer le respect exact de la casse du motif, il faut ajouter **_f_** immédiatement après celui-ci. Exemple: <A><<^é>>_f_

4.8 Recherche

4.8.1 Configuration de la recherche

Pour pouvoir rechercher une expression, il faut tout d'abord ouvrir un texte (voir chapitre 2). Cliquez ensuite sur "Locate Pattern..." dans le menu "Text". La fenêtre de la figure 4.4 apparaît alors.

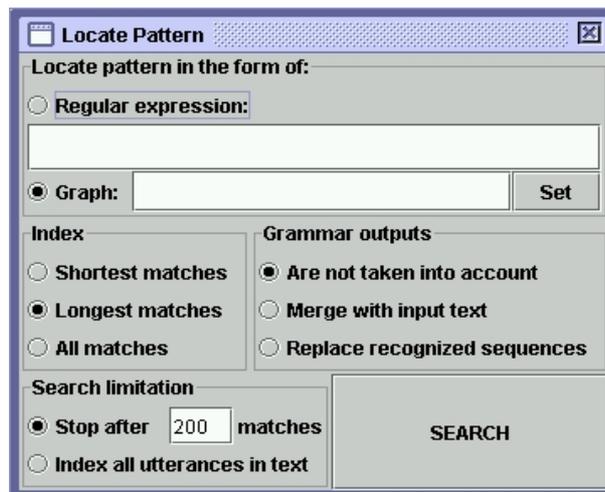


FIG. 4.4 – Fenêtre de recherche d'expressions

Le cadre "Locate pattern in the form of" permet de choisir entre une expression rationnelle et une grammaire. Cliquez sur "Regular expression".

Le cadre "Index" permet de sélectionner le mode de reconnaissance:

- "Shortest matches": donne la priorité aux séquences les plus courtes;
- "Longest matches": donne la priorité aux séquences les plus longues. C'est le mode utilisé par défaut;

- "All matches" : donne toutes les séquences reconnues.

Le cadre "Search limitation" permet de limiter ou non la recherche à un certain nombre d'occurrences. Par défaut, la recherche est limitée aux 200 premières occurrences.

Les options du cadre "Grammar outputs" ne concernent pas les expressions rationnelles. Elles sont décrites à la section 6.6.

Entrez une expression et cliquez sur "Search" pour lancer la recherche. Unitex va transformer l'expression en une grammaire au format `.grf`. Cette grammaire va ensuite être compilée en une grammaire au format `.fst2` qui sera utilisée par le programme de recherche.

4.8.2 Affichage des résultats

Une fois la recherche terminée, la fenêtre de la figure 4.5 apparaît, indiquant le nombre d'occurrences trouvées, le nombre d'unités lexicales reconnues, ainsi que le rapport entre ce nombre et le nombre total d'unités lexicales du texte.



FIG. 4.5 – Résultats de la recherche

Après avoir cliqué sur "OK", vous verrez apparaître la fenêtre de la figure 4.6 permettant de configurer l'affichage de la liste des occurrences trouvées. Vous pouvez également faire apparaître cette fenêtre en cliquant sur "Display Located Sequences..." dans le menu "Text". On appelle *concordance* la liste d'occurrences.

Le cadre "Modify text" offre la possibilité de remplacer les occurrences trouvées par les sorties produites. Cette possibilité sera examinée au chapitre 6.

Le cadre "Extract units" vous permet de construire un fichier texte avec toutes les phrases contenant ou non des occurrences. Le bouton "Set File" vous permet de sélectionner le fichier de sortie. Cliquez ensuite sur "Extract matching units" ou "Extract unmatching units" selon que vous voulez extraire les phrases contenant les occurrences ou non.

Dans le cadre "Show Matching Sequences in Context", vous pouvez sélectionner la longueur en caractères des contextes gauche et droit des occurrences qui seront affichées dans la

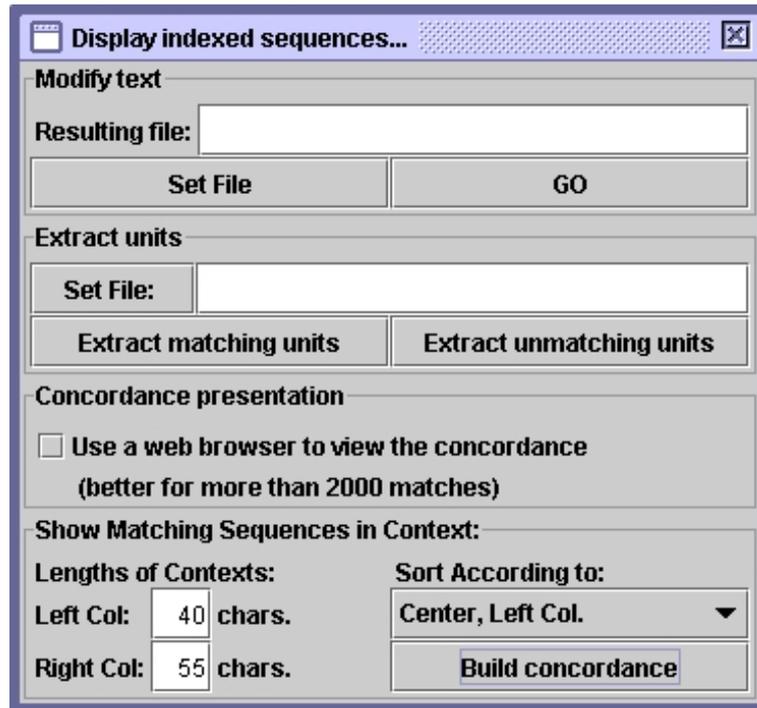


FIG. 4.6 – Configuration de l'affichage des occurrences trouvées

concordance. Si une occurrence a une longueur inférieure à la taille du contexte droit, la ligne de concordance sera complétée avec le nombre de caractères nécessaire. Si une occurrence a une longueur supérieure à la taille du contexte droit, elle est affichée en entier.

NOTE: en thaï, la taille des contextes est mesurée en caractères affichables et non en caractères réels. Cela permet de conserver l'alignement des lignes de concordance malgré la présence des caractères diacritiques qui se combinent à d'autres lettres au lieu de s'afficher comme des caractères normaux.

Vous pouvez sélectionner le mode de tri à appliquer dans la liste "Sort According to". Le mode "Text Order" affiche les occurrences dans l'ordre où elles apparaissent dans le texte. Les six autres modes permettent de trier en colonnes. Les trois zones d'une ligne sont le contexte gauche, l'occurrence et le contexte droit. Les occurrences et les contextes droits sont triés de gauche à droite. Les contextes gauches sont triés de droite à gauche. Le mode utilisé par défaut est "Center, Left Col.". La concordance est produite sous la forme d'un fichier HTML.

Lorsque les concordances atteignent plusieurs milliers d'occurrences, il est préférable de les afficher avec un navigateur web (Mozilla [7], Netscape [8], etc.). Pour cela, cochez la case "Use a web browser to view the concordance" (voir figure 4.6). Cette option est activée par défaut lorsque le nombre d'occurrences est supérieur à 3000. Pour définir le navigateur qui sera utilisé, cliquez sur "Preferences..." dans le menu "Info". Cliquez sur l'onglet "Text Presentation" et sélectionnez le programme à utiliser dans le cadre "Html Viewer" (voir figure 4.7).

Si vous choisissez d'ouvrir la concordance à l'intérieur d'Unitex, vous verrez une fenêtre comme celle de la figure 4.8. L'option "Enable links" activée par défaut permet de considérer les occurrences comme des liens hypertextes. Ainsi, quand on clique sur une occurrence, cela ouvre la fenêtre du texte et y sélectionne la séquence reconnue. De plus, si l'automate du texte est construit et que cette fenêtre n'est pas réduite sous forme d'icône, l'automate de la phrase contenant l'occurrence cliquée est chargé. Si l'on sélectionne l'option "Allow concordance edition", on ne peut pas cliquer ainsi sur les occurrences, mais l'on peut éditer la concordance comme du texte. Cela permet entre autres de s'y déplacer avec un curseur, ce qui peut être pratique si l'on travaille sur une concordance avec de grands contextes.

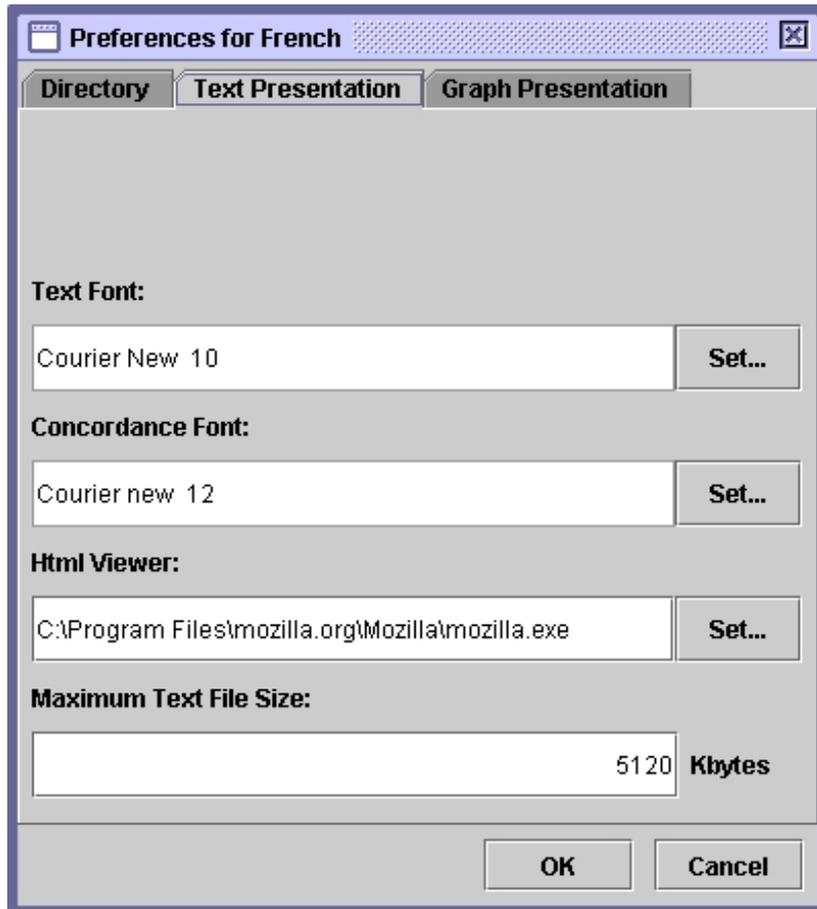


FIG. 4.7 – Sélection d'un navigateur pour l'affichage des concordances

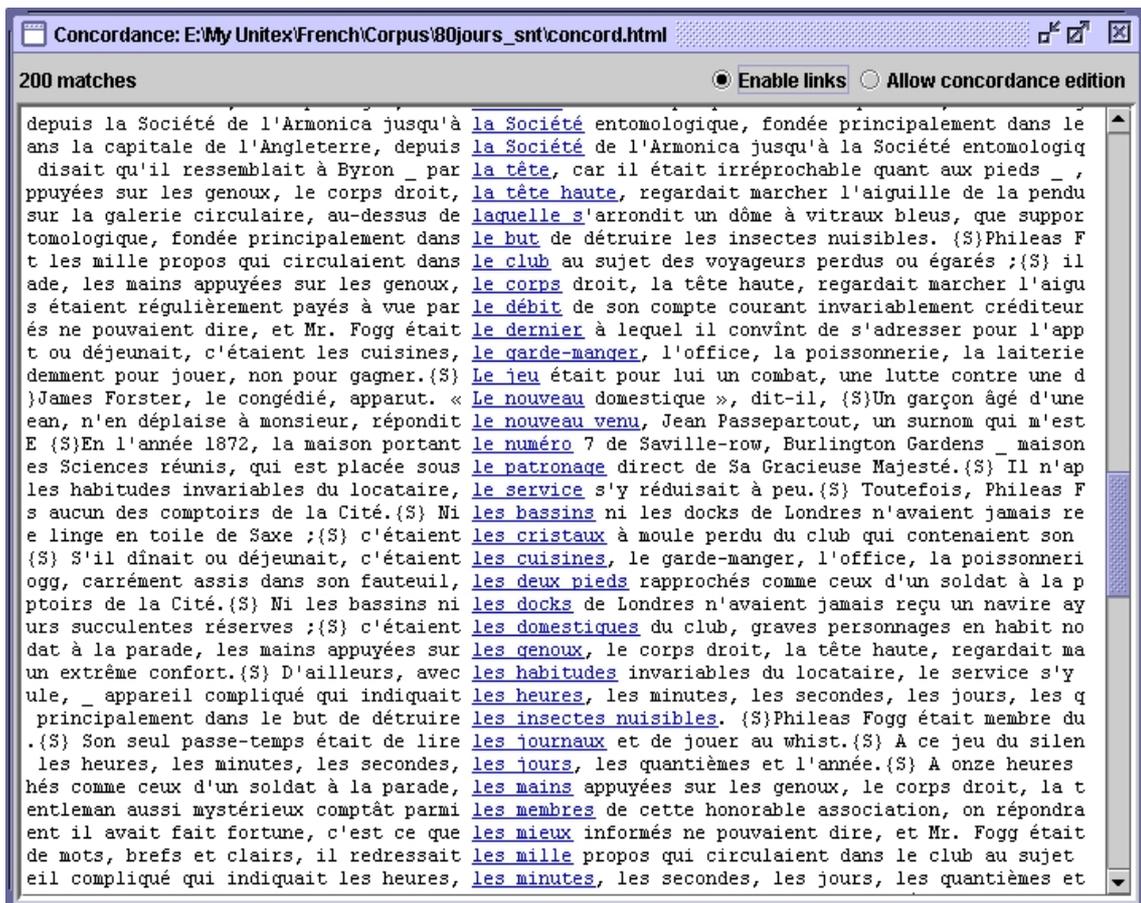


FIG. 4.8 – Exemple de concordance

Chapitre 5

Grammaires locales

Les grammaires locales sont un moyen puissant de représenter la plupart des phénomènes linguistiques. La première section présentera le formalisme sur lequel ces grammaires reposent. Nous verrons ensuite comment construire et présenter des grammaires avec Unitex.

5.1 Formalisme des grammaires locales

5.1.1 Grammaires algébriques

Les grammaires Unitex sont des variantes des grammaires algébriques, également appelées grammaires hors-contexte. Une grammaire algébrique est constituée de règles de réécriture. Voici une grammaire qui reconnaît n'importe quel nombre de caractères a :

$$\begin{aligned} S &\rightarrow aS \\ S &\rightarrow \varepsilon \end{aligned}$$

Les symboles figurant à gauche des règles sont appelés *symboles non-terminaux* car ils peuvent être réécrits. Les symboles qui ne peuvent pas être réécrits par des règles sont appelés *symboles terminaux*. Les membres droits des règles sont des suites de symboles non-terminaux et terminaux. Le symbole epsilon noté ε désigne le mot vide. Dans la grammaire ci-dessus, S est un symbole non-terminal et a un terminal. S peut se réécrire soit en un a suivi d'un S , soit en mot vide. L'opération de réécriture par l'application d'une règle est appelée *dérivation*. On dit qu'une grammaire reconnaît un mot s'il existe une suite de dérivations qui produit ce mot. Le non-terminal qui sert de point de départ à la première dérivation est appelé *axiome*.

La grammaire ci-dessus reconnaît ainsi le mot aa , car on peut obtenir ce mot depuis l'axiome S en effectuant les dérivations suivantes:

Dérivation 1: réécriture de l'axiome en aS

$$\underline{S} \rightarrow aS$$

Dérivation 2: réécriture du S du membre droit en aS

$$S \rightarrow a\underline{S} \rightarrow aaS$$

Dérivation 3: réécriture du S en ε

$$S \rightarrow aS \rightarrow aa\underline{S} \rightarrow aa$$

On appelle langage d'une grammaire l'ensemble des mots reconnus par celle-ci. Les langages reconnus par les grammaires algébriques sont appelés *langages algébriques*.

5.1.2 Grammaires algébriques étendues

Les grammaires algébriques étendues sont des grammaires algébriques où les membres droits des règles ne sont plus des suites de symboles mais des expressions rationnelles. Ainsi, la grammaire reconnaissant une suite quelconque de a peut se réécrire en une grammaire étendue d'une seule règle:

$$S \rightarrow a^*$$

Ces grammaires, également appelées *réseaux de transitions récursifs* (*RTN* en anglais) ou *diagrammes de syntaxe*, se prêtent à une représentation graphique conviviale. En effet, le membre droit d'une règle peut être représenté par un graphe dont le nom est le membre gauche de la règle.

Toutefois, les grammaires Unitex ne sont pas exactement des grammaires algébriques étendues, car elles intègrent la notion de *transduction*. Cette notion, empruntée aux automates à états finis, signifie qu'une grammaire peut produire des sorties. Dans un souci de clarté, nous utiliserons malgré tout les termes grammaire ou graphe. Quand une grammaire produira des sorties, nous utiliserons le terme *transducteur*, par extension de la définition d'un transducteur dans le domaine des automates à états finis.

5.2 Édition de graphes

5.2.1 Importation d'un graphe Intex

Pour pouvoir utiliser des graphes Intex dans Unitex, il faut les convertir en Unicode. Le procédé de conversion est le même que pour les textes (voir section 2.2). Si vous utilisez un traitement de texte pour effectuer la conversion, vérifiez que le graphe porte toujours l'extension `.grf` après la conversion, car il arrive que l'extension `.txt` soit rajoutée automatiquement. Si l'extension `.txt` a été ajoutée, supprimez-la.

ATTENTION: un graphe converti en Unicode qui a été utilisé avec Unitex ne peut plus être utilisé avec Intex.

Pour pouvoir l'utiliser à nouveau avec Intex, vous devez le convertir en texte ASCII, puis l'ouvrir avec un traitement de texte et remplacer la première ligne:

```
#Unigraph
```

par la ligne suivante:

```
#FSGraph 4.0
```

5.2.2 Création d'un graphe

Pour créer un graphe, cliquez sur "New" dans le menu "FSGraph". Vous voyez alors apparaître la fenêtre de la figure 5.2. Le symbole en forme de flèche est l'*état initial* du graphe. Le symbole rond contenant un carré est l'*état final* du graphe. La grammaire ne reconnaîtra que les expressions décrites par des chemins reliant l'état initial à l'état final.



FIG. 5.1 – Menu *FSGraph*

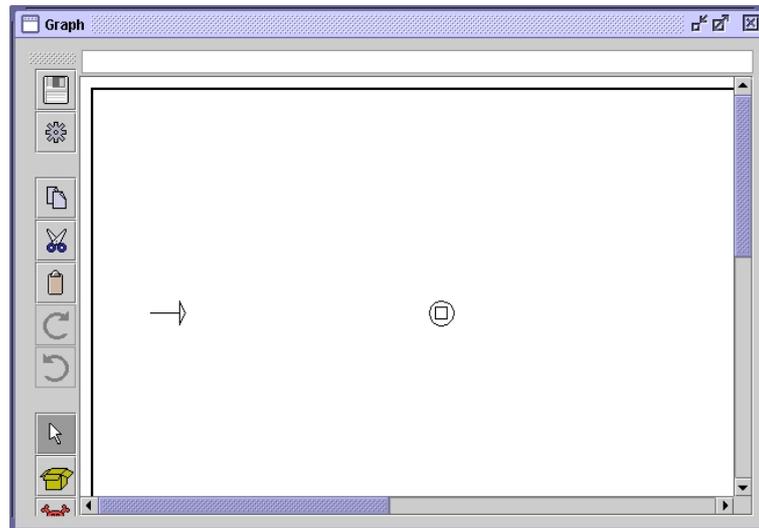


FIG. 5.2 – Graphe vierge

Pour créer une boîte, cliquez sur la fenêtre tout en appuyant sur la touche Ctrl. Vous verrez alors apparaître un carré bleu symbolisant la boîte vide créée (voir figure 5.3). Lors de la création d'une boîte, celle-ci est automatiquement sélectionnée. Vous voyez donc le

contenu de la boîte s'affiche dans la zone de texte située en haut de la fenêtre. La boîte créée contient le symbole $\langle E \rangle$ qui représente le mot vide epsilon. Remplacez ce symbole par le texte `le+la+l'+les` et validez en appuyant sur la touche Entrée. Vous venez de créer une boîte contenant quatre lignes (voir figure 5.4). En effet, le caractère `+` sert de séparateur. La boîte apparaît sous la forme de lignes de texte rouge car elle n'est pour l'instant reliée à aucune autre. On utilise souvent ce type de boîtes pour insérer des commentaires dans un graphe.

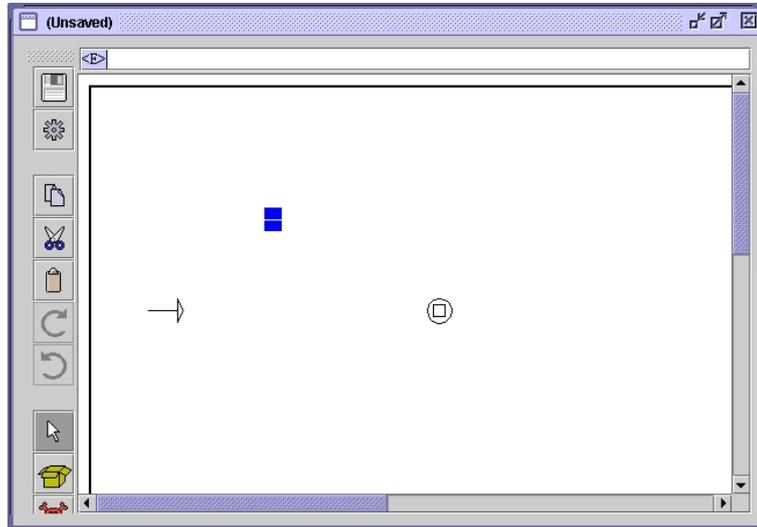


FIG. 5.3 – *Création d'une boîte*

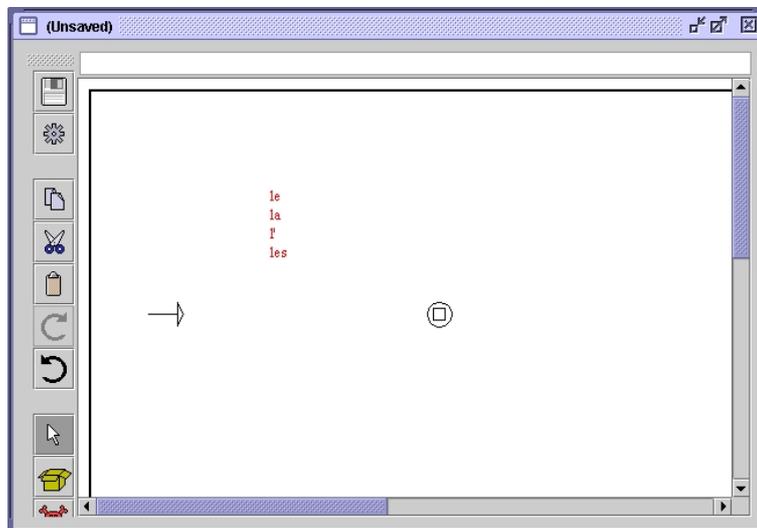


FIG. 5.4 – *Boîte contenant le+la+l'+les*

Pour relier une boîte à une autre, il faut cliquer sur la boîte de départ puis sur la boîte de destination. S'il y a déjà une transition entre les deux boîtes, celle-ci est enlevée. Il est

possible d'effectuer cette même opération en cliquant d'abord sur la boîte de destination, puis sur la boîte de départ tout en pressant sur la touche Shift. Dans notre exemple, une fois la boîte reliée à l'état initial et à l'état final du graphe, on obtient le graphe de la figure 5.5:

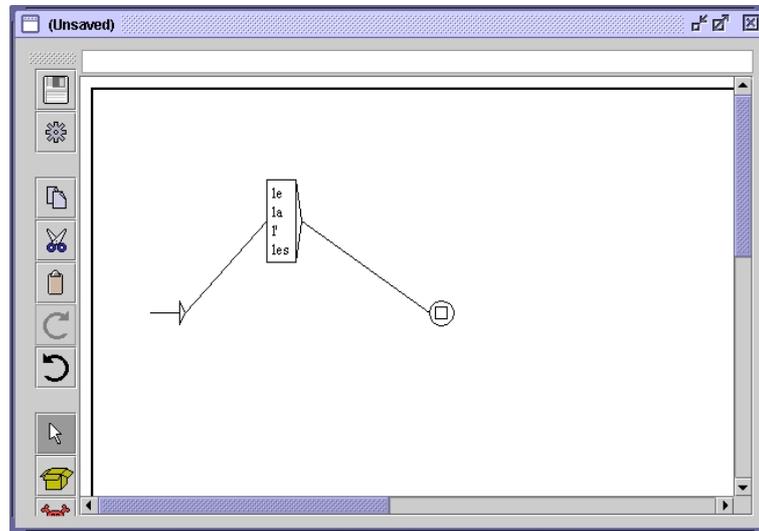


FIG. 5.5 – Graphe reconnaissant des déterminants

NOTE: si vous double-cliquez sur une boîte, vous relierez cette boîte à elle-même (voir figure 5.6). Pour annuler, double-cliquez une nouvelle fois sur la boîte.



FIG. 5.6 – Boîte reliée à elle-même

Cliquez sur "Save as..." dans le menu "FSGraph" pour sauver ce graphe. Par défaut, Unitex vous propose d'effectuer la sauvegarde dans le sous-répertoire **Graphs** de votre répertoire personnel. Vous pouvez voir si le graphe a été modifié depuis la dernière sauvegarde en regardant si le titre de la fenêtre contient la mention (**Unsaved**).

5.2.3 Sous-graphes

Pour faire appel à un sous-graphe, il faut indiquer son nom dans une boîte en le faisant précéder du caractère `:`. Si vous entrez le texte `alpha+:beta+gamma+:e:\grec\delta.grf` dans une boîte, vous obtiendrez une boîte similaire à celle de la figure 5.7:

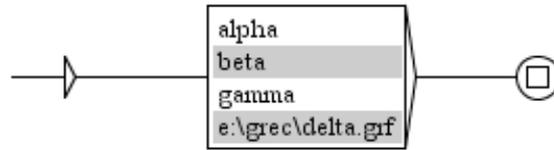


FIG. 5.7 – Graphe faisant appel aux sous-graphes *beta* et *delta*

Vous pouvez indiquer le nom complet du graphe (`e:\grec\delta.grf`) ou simplement le nom sans le chemin d'accès (`beta`); dans ce cas, le sous-graphe est supposé se trouver dans le même répertoire que le graphe qui y fait référence.

Les appels à des sous-graphes sont représentés dans les boîtes par des lignes grisées. Sous Windows, vous pouvez ouvrir un sous-graphe en cliquant sur la ligne grisée tout en appuyant sur la touche Alt. Sous Linux, la combinaison `<Alt+Click>` est interceptée par le système. Pour ouvrir un sous-graphe, cliquez sur son nom en pressant simultanément les boutons gauche et droit de la souris.

5.2.4 Manipulation des boîtes

Vous pouvez sélectionner plusieurs boîtes au moyen de la souris. Pour cela, cliquez et déplacez la souris sans relâcher le bouton. Lorsque vous relâchez le bouton, toutes les boîtes touchées par le rectangle de sélection seront sélectionnées et s'afficheront alors en blanc sur fond bleu :



FIG. 5.8 – Sélection de plusieurs boîtes

Lorsque des boîtes sont sélectionnées, vous pouvez les déplacer en cliquant et en déplaçant le curseur sans relâcher le bouton. Pour annuler la sélection, cliquez sur une zone vide du graphe; si vous cliquez sur une boîte, toutes les boîtes de la sélection seront reliées à celle-ci.

Vous pouvez effectuer un copier-coller sur plusieurs boîtes. Pour cela, sélectionnez-les et appuyez sur `<Ctrl+C>` ou cliquez sur "Copy" dans le menu "Edit". Votre sélection multiple est maintenant dans le presse-papiers d'Unitex. Vous pouvez alors coller cette sélection en pressant `<Ctrl+V>` ou en cliquant sur "Paste" dans le menu "Edit".

NOTE: vous pouvez coller une sélection multiple dans un autre graphe que celui dans lequel vous avez effectué la copie.

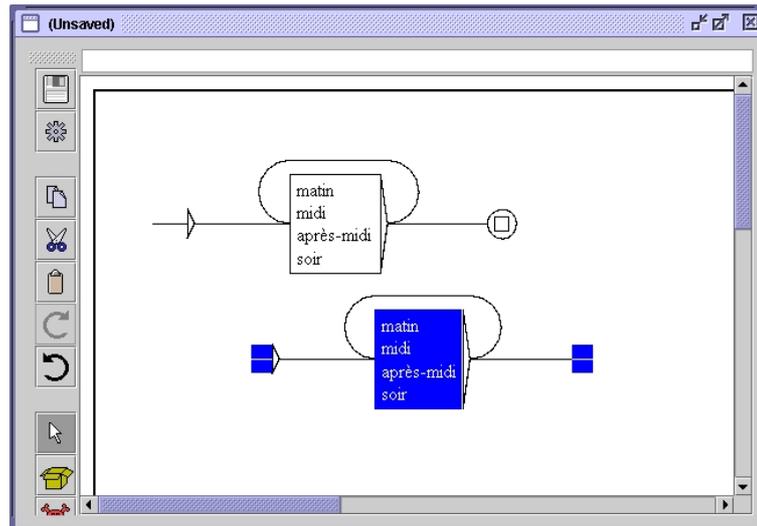


FIG. 5.9 – Copier-coller d'une sélection multiple

Pour supprimer des boîtes, sélectionnez-les et supprimez le texte qu'elles contiennent. Pour cela, supprimez le texte présent dans la zone de texte située en haut de la fenêtre et validez avec la touche Entrée. L'état initial et l'état final ne peuvent pas être supprimés.

5.2.5 Sortie

Il est possible d'associer une sortie à une boîte. Pour cela, utilisez le caractère spécial /. Tous les caractères situés à droite de celui-ci seront considérés comme faisant partie de la sortie. Ainsi, le texte `un+deux+trois/nombre` donne la boîte de la figure 5.10:

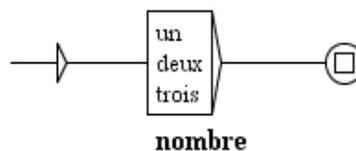


FIG. 5.10 – Exemple de sortie

La sortie associée à une boîte est représentée en texte gras sous celle-ci.

5.2.6 Utilisation des variables

Il est possible de sélectionner des parties du texte reconnu par une grammaire au moyen de variables. Pour associer une variable `var1` à une partie d'une grammaire, utilisez les symboles spéciaux `$var1(` (et `$var1)` pour définir respectivement le début et la fin de la zone à stocker.

Créez deux boîtes contenant l'une `$var1` (et l'autre `$var1`). Ces boîtes ne doivent rien contenir d'autre que le nom de la variable précédé de `$` et suivi d'une parenthèse. Reliez ensuite ces boîtes à la zone de la grammaire voulue. Dans le graphe de la figure 5.11, on reconnaît une séquence commençant par une majuscule après `Monsieur` ou `M.`, et on la stocke dans une variable nommée `var1`.

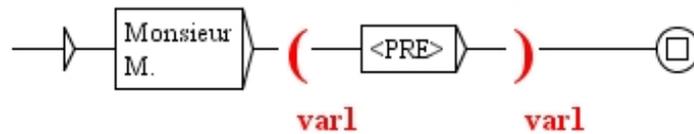


FIG. 5.11 – Utilisation d'une variable `var1`

Les noms de variables peuvent contenir des lettres non accentuées, minuscules ou majuscules, ainsi que des chiffres et le caractère `_` (underscore). Unitex fait la différence entre les lettres minuscules et majuscules.

Quand une variable a ainsi été définie, on peut l'utiliser dans les sorties en faisant précéder son nom du caractère `$`. La grammaire de la figure 5.12 reconnaît une date formée d'un mois et d'une année, et produit en sortie la même date, mais dans l'ordre année-mois.

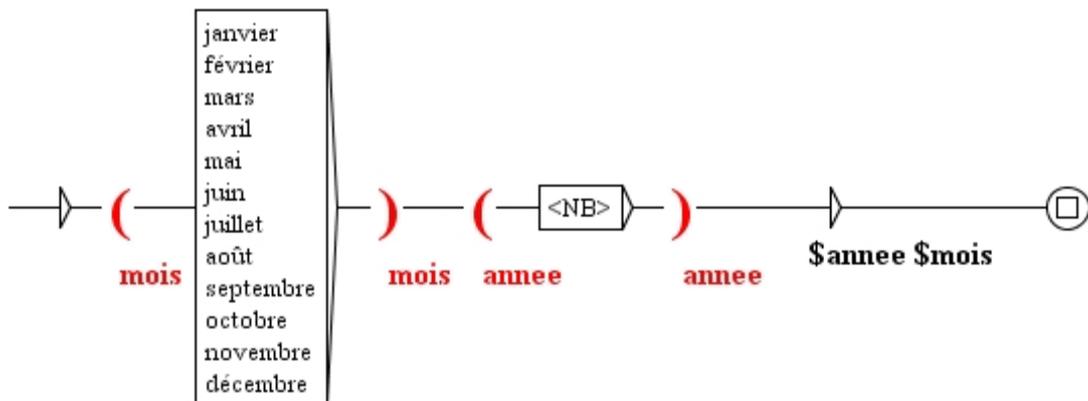


FIG. 5.12 – Inversion du mois et de l'année dans une date

5.2.7 Copie de listes

Il peut être pratique d'effectuer un copier-coller d'une liste de mots ou d'expressions depuis un éditeur de texte vers une boîte dans un graphe. Afin d'éviter de devoir copier manuellement chaque terme, Unitex propose un mécanisme de copie de listes. Pour l'utiliser, sélectionnez votre liste dans votre éditeur de texte et copiez-la au moyen de `<Ctrl+C>` ou de la fonction de

copie intégrée à votre éditeur. Créez ensuite une boîte dans votre graphe, et utilisez <Ctrl+V> ou la commande "Paste" du menu "Edit" pour la coller dans la boîte. Vous verrez alors apparaître la fenêtre de la figure 5.13:



FIG. 5.13 – Sélection de contexte pour la copie d'une liste

Cette fenêtre vous permet de définir les contextes gauche et droit qui seront ajoutés automatiquement à chaque terme de la liste. Par défaut, ces contextes sont vides. Si l'on applique les contextes < et .V> à la liste suivante:

manger
dormir
boire
jouer
lire

on obtient la boîte de la figure 5.14:

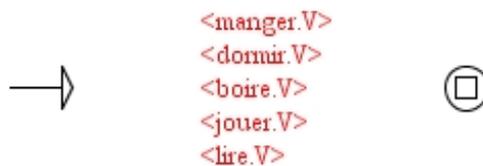


FIG. 5.14 – Boîte obtenue par copie d'une liste avec ajout de contextes

5.2.8 Symboles spéciaux

L'éditeur de graphes d'Unitex interprète de façon particulière les symboles suivants:

" + : / < > # \

Le tableau 5.1 résume la signification pour Unitex de ces symboles, ainsi que la ou les façons de reconnaître ces caractères dans des textes.

Caractère	Signification	Codage
"	les guillemets délimitent des séquences qui ne doivent ni être interprétées par Unitex, ni subir de variantes de casse	\"
+	le + sépare les différentes lignes des boîtes	"+"
:	le : sert à introduire un appel à un sous-graphe	":" ou \:
/	le / indique le début de la sortie dans une boîte	\/
<	le < indique le début d'un motif ou d'un méta	"<" ou \<
>	le > indique la fin d'un motif ou d'un méta	">" ou \>
#	le # sert à interdire la présence de l'espace	"#"
\	le \ sert à déspecialiser la plupart des caractères spéciaux	\\ \\

TAB. 5.1 – Codage des symboles spéciaux dans l'éditeur de graphes

5.2.9 Commandes de la barre d'icônes

La barre d'icônes présente à gauche des graphes contient des raccourcis vers certaines commandes et permet de manipuler les boîtes d'un graphe en utilisant des "outils". Cette barre d'icônes peut être déplacée en cliquant sur la zone "rugueuse". Elle peut même être dissociée du graphe et apparaître alors comme une fenêtre séparée (voir figure 5.15). Dans ce cas, le fait de fermer cette fenêtre remplace la barre d'icônes à sa position initiale. Chaque graphe possède sa propre barre d'icônes.



FIG. 5.15 – Barre d'icônes

Les deux premières icônes sont des raccourcis permettant de sauver et de compiler le graphe. Les trois suivantes correspondent aux opérations "Copier", "Couper" et "Coller". Les deux suivantes correspondent aux opérations "Redo" et "Undo" qui permettent de refaire ou défaire des opérations. La dernière icône en forme de clé est un raccourci vers la fenêtre de configuration de l'apparence du graphe.

Les 6 autres icônes correspondent à des commandes d'édition des boîtes. La première, en forme de flèche blanche, correspond au mode d'édition normal des boîtes. Les 5 autres correspondent à des outils. Pour utiliser un outil, cliquez sur l'icône correspondante: le curseur

de la souris changera alors de forme et les clics de la souris seront alors interprétés de façon particulière. Voici la description des outils, de gauche à droite :

- création de boîtes: crée une boîte vide à l'endroit du clic;
- suppression de boîtes: supprime la boîte sur laquelle vous cliquez;
- relier des boîtes à une autre boîte: cet outil permet de sélectionner une ou plusieurs boîtes, et de la ou les relier à une autre. À la différence du mode normal, la ou les transitions qui vont être créées sont affichées pendant le déplacement du pointeur de la souris;
- relier des boîtes à une autre boîte en sens inverse: cet outil effectue la même chose que le précédent, mais en reliant en sens inverse les boîtes sélectionnées à la boîte cliquée;
- ouvrir un sous-graphe: ouvre un sous-graphe lorsque vous cliquez sur la ligne grisée correspondante dans une boîte.

5.3 Options de présentation

5.3.1 Tri des lignes d'une boîte

Vous pouvez trier le contenu d'une boîte en la sélectionnant et en cliquant sur "Sort Node Label" dans le sous-menu "Tools" du menu "FSGraph". Ce tri ne fait pas appel au programme `SortTxt`. Il s'agit d'un tri basique qui trie les lignes de la boîte selon l'ordre des caractères dans le codage Unicode.

5.3.2 Zoom

Le sous-menu "Zoom" vous permet de choisir l'échelle à laquelle sera affiché le graphe.

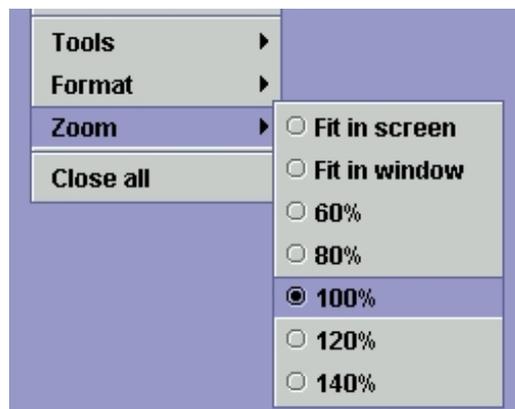


FIG. 5.16 – Sous-menu Zoom

L'option "Fit in screen" étire ou rétrécit le graphe pour lui donner la taille de l'écran. L'option "Fit in window" ajuste le graphe pour qu'il soit entièrement affiché dans la fenêtre.

5.3.3 Antialiasing

L'antialiasing est un effet de lissage qui permet d'éviter l'effet de pixellisation. Vous pouvez activer cet effet en cliquant sur "Antialiasing..." dans le sous-menu "Format". La figure 5.17 montre deux graphes affichés normalement (graphe du haut) et avec antialiasing (graphe du bas).

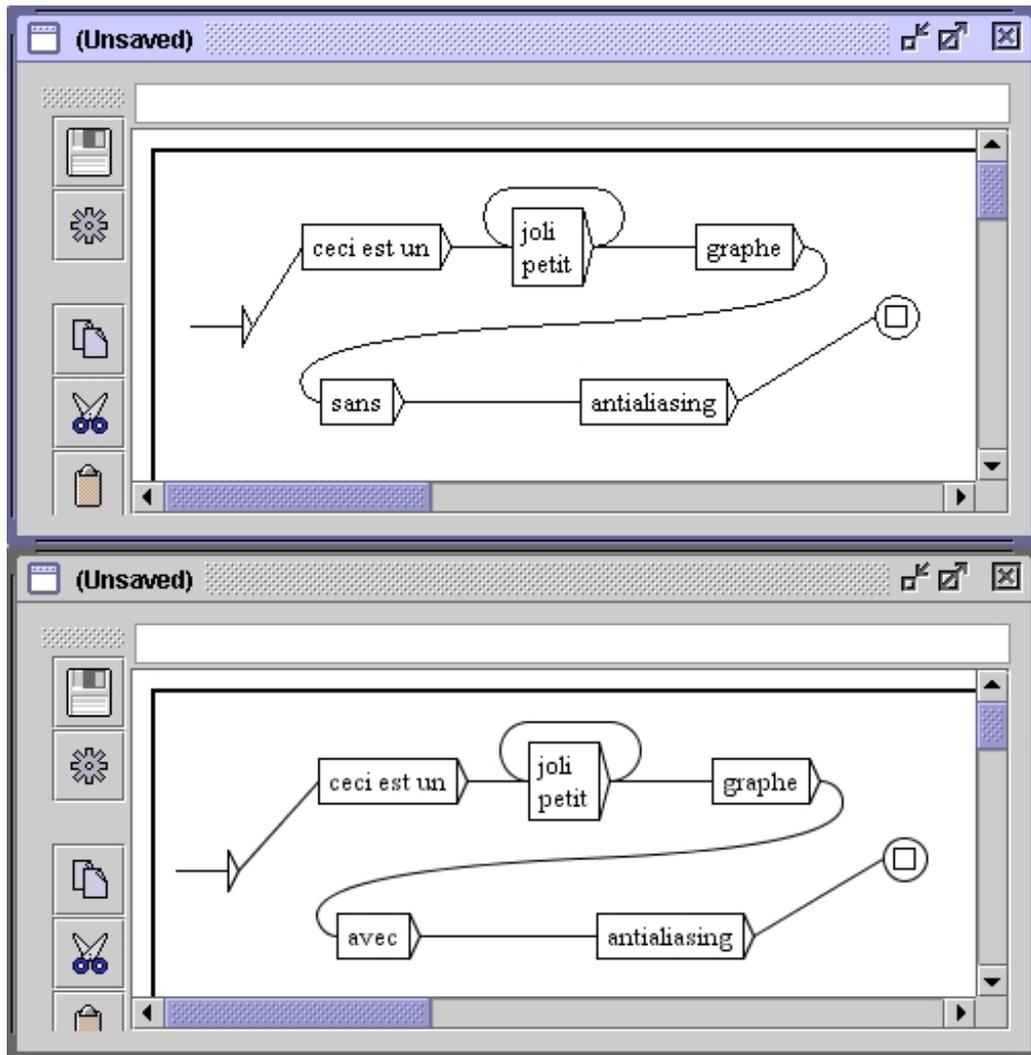


FIG. 5.17 – Exemple d'antialiasing

Cet effet ralentit l'exécution d'Unitex. Nous vous conseillons de ne pas l'utiliser si votre machine est peu puissante.

5.3.4 Alignement des boîtes

Afin d'obtenir des graphes harmonieux, il est utile de pouvoir aligner les boîtes, aussi bien horizontalement que verticalement. Pour cela, sélectionnez les boîtes à aligner et cliquez sur "Alignement..." dans le sous-menu "Format" du menu "FSGraph" ou appuyez sur <Ctrl+M>. Vous voyez alors apparaître la fenêtre de la figure 5.18.

Les possibilités d'alignement horizontal sont:

- Top: les boîtes sont alignées sur la boîte la plus haute;
- Center: les boîtes sont toutes centrées sur un même axe;
- Bottom: les boîtes sont alignées sur la boîte la plus basse.

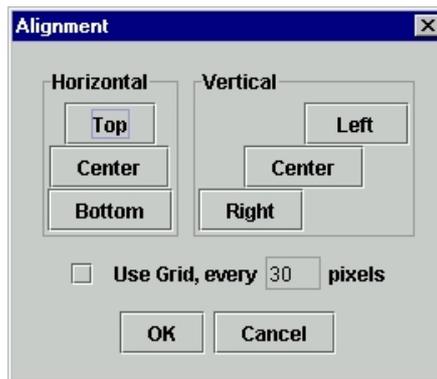


FIG. 5.18 – Fenêtre d'alignement

Les possibilités d'alignement vertical sont:

- Left: les boîtes sont alignées sur la boîte la plus à gauche;
- Center: les boîtes sont toutes centrées sur un même axe;
- Right: les boîtes sont alignées sur la boîte la plus à droite.

La figure 5.19 montre un exemple d'alignement. Le groupe de boîtes situé à droite est une copie des boîtes de gauche qui a été alignée verticalement à gauche.

L'option "Use Grid" de la fenêtre d'alignement permet d'afficher une grille en arrière-plan du graphe. Cela permet d'aligner approximativement les boîtes.

5.3.5 Présentation, polices et couleurs

Vous pouvez configurer l'aspect d'un graphe en appuyant sur <Ctrl+R> ou en cliquant sur "Presentation..." dans le sous-menu "Format" du menu "FSGraph", ce qui provoque l'affichage de la fenêtre de la figure 5.21.

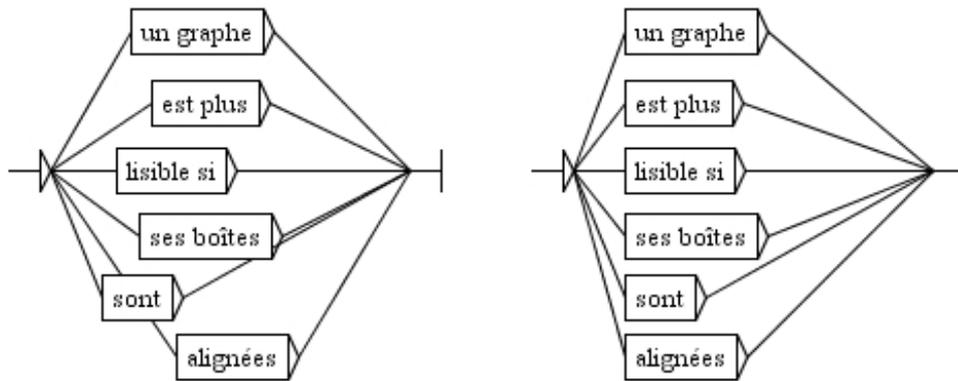


FIG. 5.19 – Exemple d'alignement vertical gauche

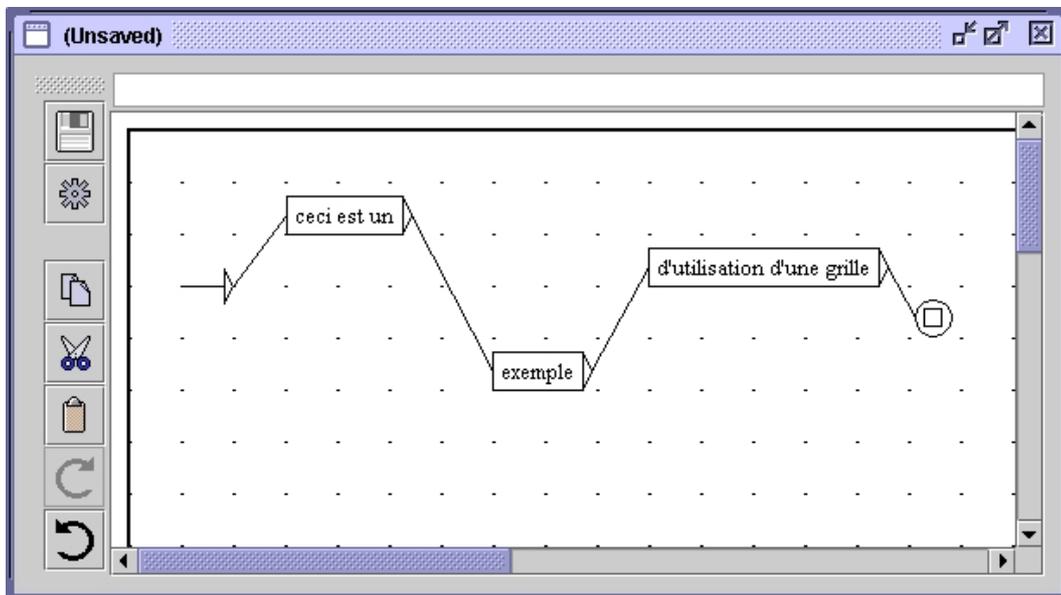


FIG. 5.20 – Exemple d'utilisation d'une grille

Les paramètres de polices sont:

- Input: police utilisée dans les boîtes, ainsi que dans la zone de texte où l'on édite le contenu des boîtes;
- Output: police utilisée pour afficher les sorties des boîtes.

Les paramètres de couleur sont:

- Background: couleur de fond;
- Foreground: couleur utilisé pour le texte et le dessin des boîtes;

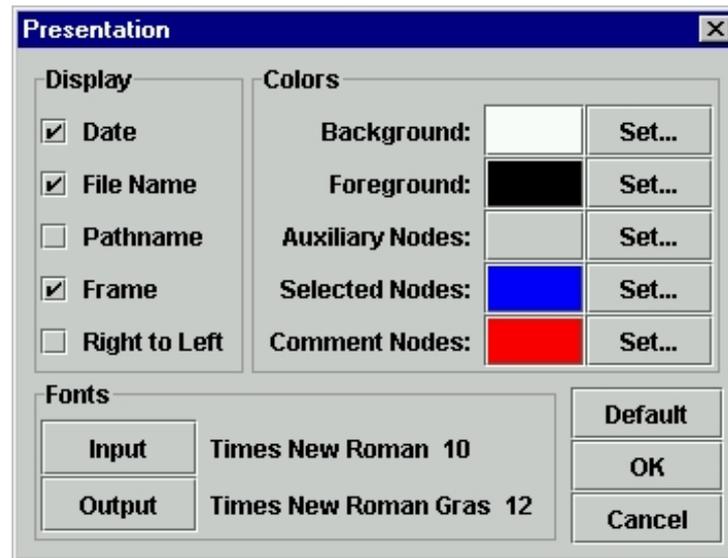


FIG. 5.21 – Configuration de l'aspect d'un graphe

- Auxiliary Nodes: couleur des boîtes faisant appel à des sous-graphes;
- Selected Nodes: couleur utilisée pour dessiner les boîtes quand elles sont sélectionnées;
- Comment Nodes: couleur utilisée pour dessiner les boîtes qui ne sont reliées à aucune autre.

Les autres paramètres sont:

- Date: affichage de la date courante dans le coin inférieur gauche du graphe;
- File Name: affichage du nom du graphe dans le coin inférieur gauche du graphe;
- Pathame: affichage du nom du graphe avec son chemin complet dans le coin inférieur gauche du graphe. Cette option n'a d'effet que si l'option "File Name" est sélectionnée;
- Frame: dessine un cadre autour du graphe;
- Right to Left: inverse le sens de lecture du graphe (voir exemple de la figure 5.22).

Vous pouvez rétablir les paramètres par défaut en cliquant sur le bouton "Default". Si vous cliquez sur le bouton "OK", seul le graphe courant sera modifié. Pour modifier les préférences par défaut d'une langue, cliquez sur "Preferences..." dans le menu "Info" et choisissez l'onglet "Graph Presentation". La fenêtre de configuration des préférences possède une option supplémentaire concernant l'antialiasing (voir figure 5.23). Cette option permet d'activer l'antialiasing par défaut pour tous les graphes de la langue courante. Il est préférable de ne pas activer cette option si votre machine n'est pas très puissante.

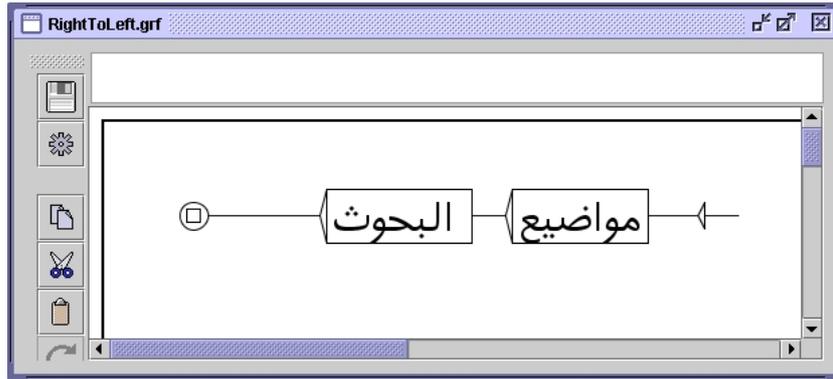


FIG. 5.22 – Graphe se lisant de droite à gauche

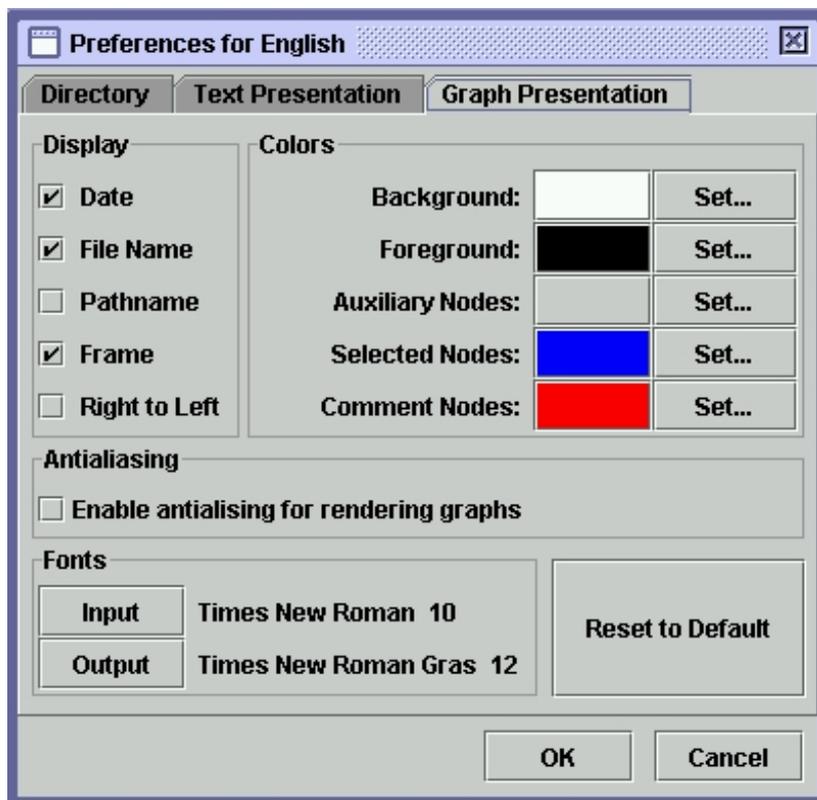


FIG. 5.23 – Configuration des préférences par défaut

5.4 Les graphes en dehors d'Unitex

5.4.1 Inclusion d'un graphe dans un document

Pour inclure un graphe dans un document, il faut en faire une image. Pour cela, activez l'antialiasing pour le graphe qui vous intéresse (ce n'est pas obligatoire, mais ça permet

d'obtenir une image plus lisse).

Sous Windows:

Appuyez ensuite sur la touche "Imprime écran" de votre clavier qui doit se trouver près de la touche F12. Lancez le programme `Paint` dans le menu "Accessoires" de Windows. Appuyez sur <Ctrl+V>. `Paint` devrait vous dire que l'image contenue dans le presse-papiers est trop grande et vous demander si vous voulez agrandir l'image. Cliquez sur "Oui". Vous pouvez maintenant éditer l'image de l'écran. Sélectionnez la zone qui vous intéresse. Pour cela, passez en mode sélection en cliquant sur le rectangle en pointillé qui se trouve dans le coin supérieur gauche de la fenêtre. Vous pouvez maintenant sélectionner une zone de l'image avec la souris. Une fois votre zone sélectionnée, appuyez sur <Ctrl+C>. Votre sélection est maintenant dans le presse-papier, il ne vous reste plus qu'à aller dans votre document et à appuyer sur <Ctrl+V> pour coller votre image.

Sous Linux:

Effectuez une capture d'écran (par exemple avec le programme `xv`). Retaillez ensuite votre image avec un éditeur graphique (par exemple `TheGimp`), et collez votre image dans votre document de la même façon que sous Windows.

5.4.2 Impression d'un graphe

Vous pouvez imprimer un graphe en cliquant sur "Print..." dans le menu "FSGraph" ou en appuyant sur <Ctrl+P>.

ATTENTION: vous devez vous assurer que le paramètre d'orientation de l'imprimante (portrait ou paysage) correspond bien à l'orientation de votre graphe.

Vous pouvez définir vos préférences d'impression en cliquant sur "Page Setup" dans le menu "FSGraph". Vous pouvez également imprimer tous les graphes qui sont ouverts en cliquant sur "Print All...".

Chapitre 6

Utilisation avancée des graphes

6.1 Les types de graphes

Unitex peut manipuler 4 types de graphes qui correspondent aux utilisations suivantes: flexion automatique de dictionnaires, prétraitement des textes, normalisation des automates de texte et recherche de motifs. Ces différents types de graphes ne sont pas interprétés de la même façon par Unitex. Certaines choses comme les sorties sont permises pour certains types et interdites pour d'autres. De plus, les symboles spéciaux ne sont pas les mêmes en fonction du type de graphe. Cette section présente donc chacun des types de graphes en détaillant leurs particularités.

6.1.1 Graphes de flexion

Un graphe de flexion décrit les variations morphologiques associées à une classe de mots, en associant à chaque variante des codes flexionnels. Les chemins d'un tel graphe décrivent les modifications à appliquer aux formes canoniques tandis que les sorties contiennent les informations flexionnelles qui seront produites.

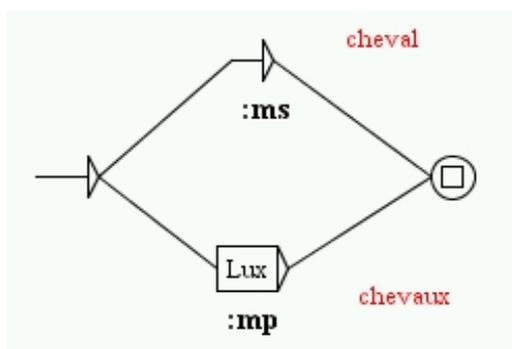


FIG. 6.1 – *Exemple de grammaire de flexion*

Les chemins peuvent contenir des opérateurs et des lettres. Les opérateurs possibles sont représentés par les caractères L et R. Les lettres sont tous les caractères qui ne sont pas des

opérateurs. Le seul symbole spécial autorisé est le mot vide <E>. Il n'est pas possible de faire référence aux dictionnaires dans un graphe de flexion. Il est également impossible de faire appel à des sous-graphes.

Les sorties sont concaténées pour produire une chaîne de caractères. Cette chaîne est ensuite concaténée à la ligne de dictionnaire produite (voir chapitre 3.4). Les sorties à variables n'ont pas de sens dans un graphe de flexion.

Le contenu d'un graphe de flexion est manipulé sans aucune variante de casse: les lettres minuscules restent minuscules, idem pour les majuscules. En outre, la liaison de deux boîtes est strictement équivalente à la concaténation de leurs contenus munie de la concaténation de leurs sorties (voir figure 6.2).

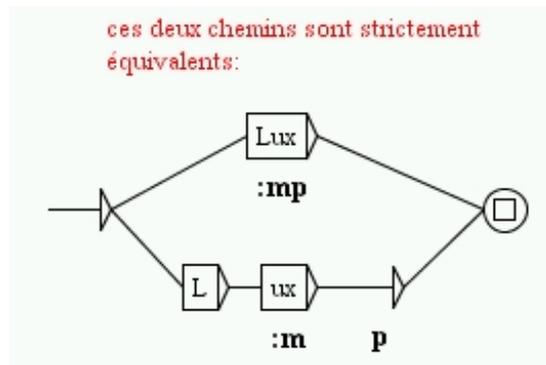


FIG. 6.2 – Deux chemins équivalents dans une grammaire de flexion

Les graphes de flexion doivent être compilés avant de pouvoir être utilisés par le programme de flexion.

6.1.2 Graphes de prétraitement

Les graphes de prétraitement sont destinés à être appliqués aux textes avant que ceux-ci soient découpés en unités lexicales. Ces graphes peuvent être utilisés pour insérer ou remplacer des séquences dans les textes. Les deux utilisations usuelles de ces graphes sont la normalisation de formes non ambiguës et le découpage en phrases.

L'interprétation de ces graphes dans Unitex est très proche de celle des graphes syntaxiques utilisés pour la recherche de motifs. Les différences sont les suivantes:

- on peut utiliser le symbole spécial <^> qui reconnaît un retour à la ligne;
- il est impossible de faire référence aux dictionnaires;
- il faut compiler ces graphes pour qu'ils puissent être utilisés lors des opérations de prétraitement.

Les figures 2.9 et 2.10 montrent des exemples de graphes de prétraitement.

6.1.3 Graphes de normalisation de l'automate du texte

Les graphes de normalisation de l'automate du texte permettent de normaliser des formes ambiguës. En effet, ils peuvent décrire plusieurs étiquettes pour une même forme. Ces étiquettes sont ensuite insérées dans l'automate du texte, explicitant ainsi les ambiguïtés. La figure 6.3 montre un extrait du graphe de normalisation utilisé pour le français.

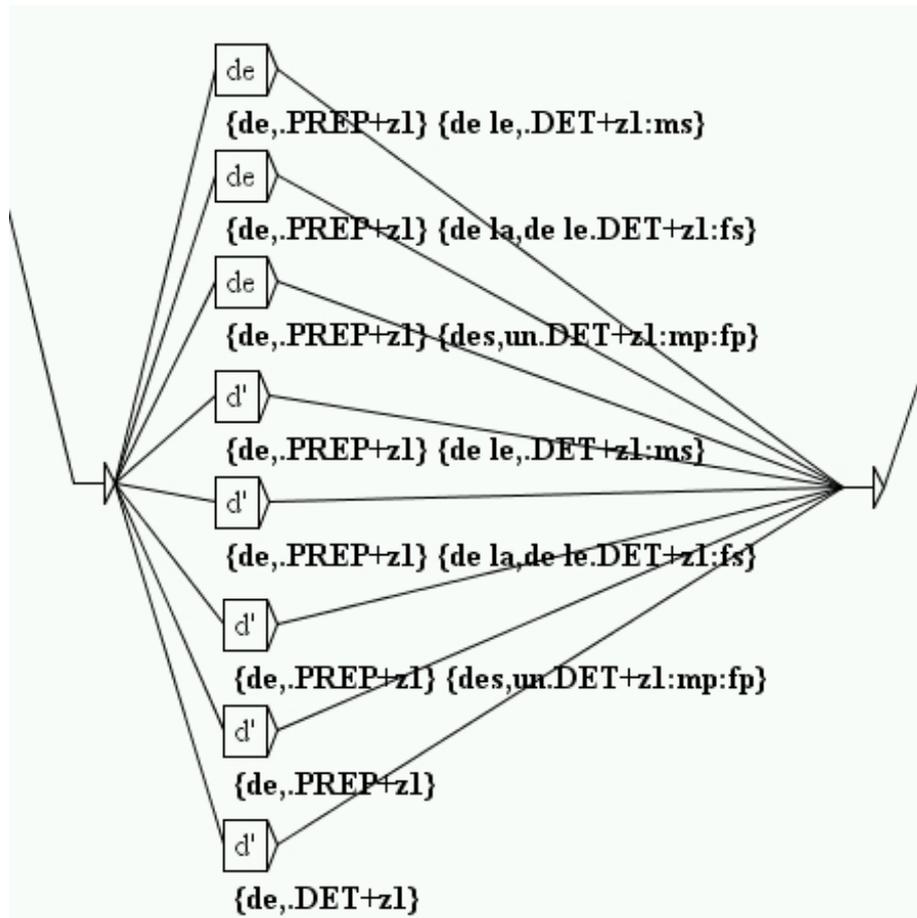


FIG. 6.3 – Extrait du graphe de normalisation utilisé pour le français

Les chemins décrivent les formes qui doivent être normalisées. Les variantes minuscules et majuscules sont prises en compte selon le principe suivant: les lettres majuscules dans le graphe ne reconnaissent que les lettres majuscules dans l'automate du texte; les lettres minuscules peuvent reconnaître les lettres minuscules et majuscules.

Les sorties représentent les séquences d'étiquettes qui seront insérées dans l'automate du texte. Ces étiquettes peuvent être des entrées de dictionnaires ou de simples chaînes de caractères. Les étiquettes représentant des entrées de dictionnaire doivent respecter le format des entrées d'un DELAF et être encadrées par les symboles { et }. Les sorties à variables n'ont pas de sens dans ce type de graphe.

Il est possible de faire appel à des sous-graphes. Il n'est pas possible de faire référence aux dictionnaires pour décrire les formes à normaliser. L'unique symbole spécial reconnu dans ce type de graphe est le mot vide <E>. Les graphes de normalisation de formes ambiguës doivent être compilés avant de pouvoir être utilisés.

6.1.4 Graphes syntaxiques

Les graphes syntaxiques, souvent appelés grammaires locales, permettent de décrire des motifs syntaxiques qui pourront ensuite être recherchés dans des textes. De tous les types de graphe, ceux-ci possèdent la plus grande puissance d'expressions, car ils permettent de faire référence aux dictionnaires.

Les variantes minuscules/majuscules sont autorisées selon le principe décrit plus haut. Il est toutefois possible de forcer le respect de la casse en encadrant une expression avec des guillemets. L'emploi des guillemets permet également de forcer le respect des espacements. En effet, Unitex considère par défaut qu'un espace est possible entre deux boîtes. Pour forcer la présence d'un espace, il faut le mettre entre guillemets. Pour interdire la présence d'un espace, il faut utiliser le symbole spécial #.

Les graphes syntaxiques peuvent faire appel à des sous-graphes (voir section 5.2.3). Ils gèrent également les sorties, y compris les sorties à variables. Les séquences produites sont interprétées comme des chaînes de caractères qui seront insérées dans les concordances, ou dans le texte si vous voulez modifier celui-ci (voir section 6.6.3).

Les symboles spéciaux supportés par les graphes syntaxiques sont les mêmes que ceux utilisables dans les expressions rationnelles (voir section 4.3.1).

Il n'est pas obligatoire de compiler les graphes syntaxiques avant de les utiliser pour la recherche de motifs. Si un graphe n'est pas compilé, le système le compilera automatiquement.

6.1.5 Grammaires ELAG

La syntaxe des grammaires de levée d'ambiguïtés est présentée à la section 7.3.1, page 100.

6.1.6 Graphes paramétrés

Les graphes paramétrés sont des méta-graphes permettant de générer une famille de graphes à partir d'une table de lexique-grammaire. Il est possible de construire des graphes paramétrés pour n'importe quel type de graphe. La construction et l'utilisation des graphes paramétrés seront développées dans le chapitre 8.

6.2 Compiler une grammaire

6.2.1 Compilation d'un graphe

La compilation est l'opération qui permet de passer du format `.grf` à un format plus facile à manipuler par les programmes d'Unitex. Pour compiler un graphe, vous devez l'ouvrir puis cliquer sur "Compile FST2" dans le sous-menu "Tools" du menu "FSGraph". Unitex lance

alors le programme `Grf2Fst2` dont vous pouvez suivre l'exécution dans une fenêtre (voir figure 6.4).

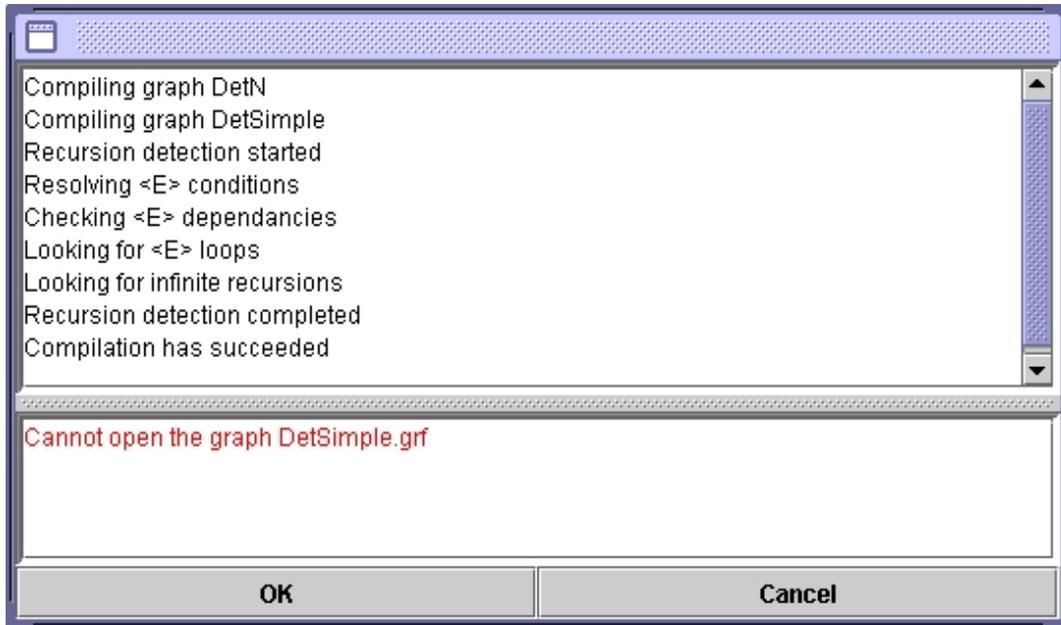


FIG. 6.4 – Fenêtre de compilation

Si le graphe fait appel à des sous-graphes, ceux-ci sont automatiquement compilés. Le résultat est un fichier `.fst2` qui rassemble tous les graphes qui composent la grammaire. La grammaire est alors prête à être utilisée par les différents programmes d'Unitex.

6.2.2 Approximation par un transducteur à états finis

Le format FST2 conserve l'architecture en sous-graphes des grammaires, ce qui les différencie des stricts transducteurs à états finis. Le programme `Flatten` permet de transformer une grammaire FST2 en un transducteur à états finis quand cela est possible, et d'en construire une approximation sinon. Cette fonction permet ainsi d'obtenir des objets plus simples à manipuler et sur lesquels peuvent s'appliquer tous les algorithmes classiques sur les automates.

Pour compiler et transformer ainsi une grammaire, sélectionnez la commande "Compile & Flatten FST2" dans le sous-menu "Tools" du menu "FSGraph". La fenêtre de la figure 6.5 vous permet de configurer l'opération d'approximation.

Le cadre "Flattening depth" permet de préciser le niveau d'imbrication des sous-graphes. Cette valeur représente la profondeur maximale au delà de laquelle les appels à des sous-graphes ne seront plus remplacés par les sous-graphes eux-mêmes.

Le cadre "Expected result grammar format" permet de déterminer le comportement du programme au delà de la limite indiquée. Si vous sélectionnez l'option "Finite State Trans-

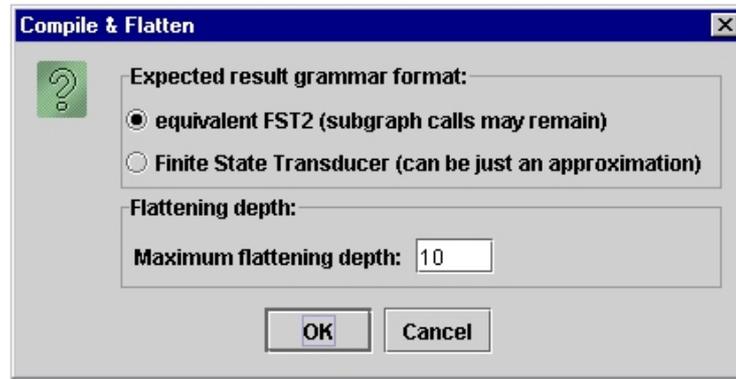


FIG. 6.5 – Configuration de l'approximation d'une grammaire

ducer", les appels aux sous-graphes seront ignorés au delà de la profondeur maximale. Cette option garantit ainsi l'obtention d'un transducteur à états finis, éventuellement non équivalent à la grammaire de départ. En revanche, l'option "equivalent FST2" indique au programme de laisser tels quels les appels aux sous-graphes au delà de la profondeur limite. Cette option garantit la stricte équivalence du résultat avec la grammaire d'origine, mais ne produit pas forcément un transducteur à états finis. Cette option peut être utilisée pour optimiser certaines grammaires.

Un message indique à la fin du processus d'approximation si le résultat est un transducteur à états finis ou une grammaire FST2, et dans le cas d'un transducteur, s'il est équivalent à la grammaire d'origine (voir figure 6.6).

6.2.3 Contraintes sur les grammaires

À l'exception des grammaires de flexion, une grammaire ne peut pas avoir de chemin vide. Cela signifie que le graphe principal d'une grammaire ne doit pas pouvoir reconnaître le mot vide, mais cela n'empêche pas un sous-graphe de cette grammaire de reconnaître epsilon.

Il n'est pas possible d'associer une sortie à un appel à un sous-graphe. De telles sorties sont ignorées par Unitex. Il faut donc utiliser une boîte vide située immédiatement à gauche de l'appel au sous-graphe pour porter la sortie (voir figure 6.7).

Les grammaires ne doivent pas non plus comporter de boucles infinies, car les programmes d'Unitex ne pourraient jamais terminer l'exploration de telles grammaires. Ces boucles peuvent être dues à des transitions étiquetées par le mot vide epsilon ou à des appels de sous-graphes récursifs.

Les boucles dues à des transitions par le mot vide peuvent avoir deux origines, dont la première est illustrée par la figure 6.8.

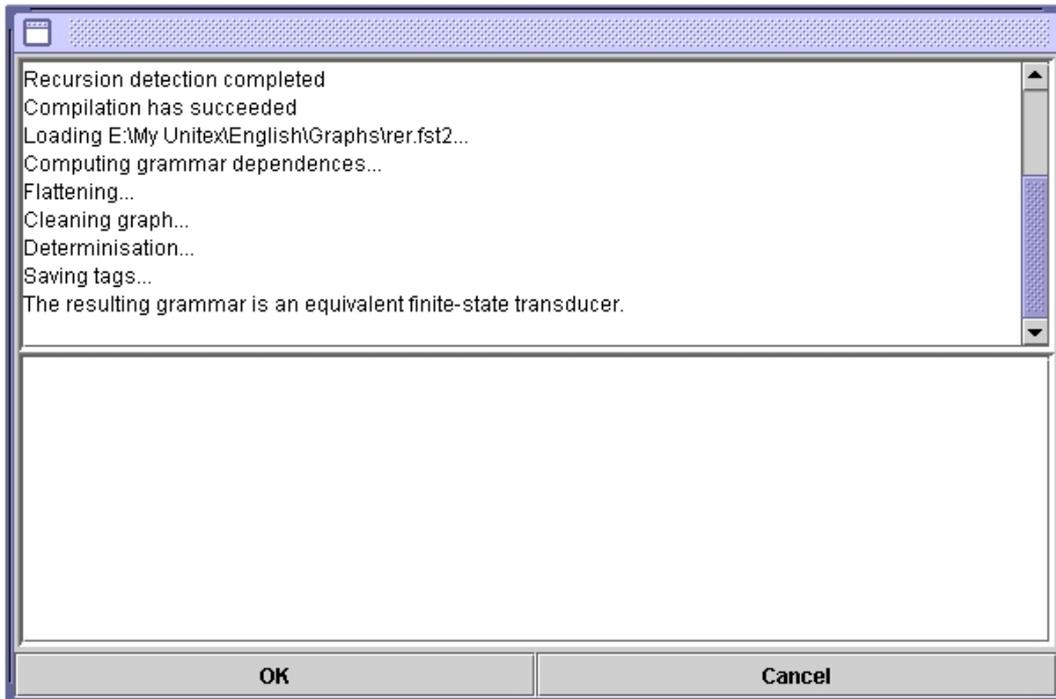


FIG. 6.6 – Résultat de l'approximation d'une grammaire

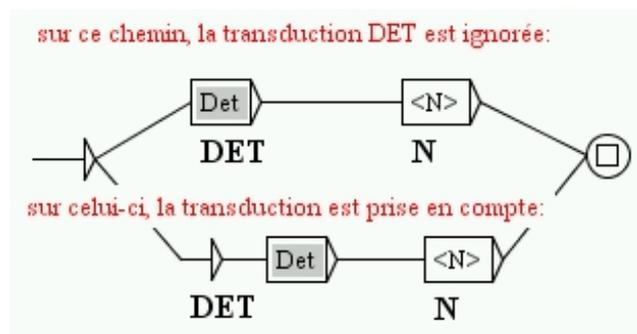


FIG. 6.7 – Comment associer une sortie à un appel de sous-graphe

Ce type de boucle est dû au fait qu'une transition par le mot vide ne peut pas être éliminée automatiquement par Unitex lorsqu'elle est munie d'une sortie. Ainsi, la transition par le mot vide de la figure 6.8 ne sera pas supprimée et provoquera une boucle infinie.

La seconde catégorie de boucle par epsilon concerne les appels à des sous-graphes pouvant reconnaître le mot vide. Ce cas de figure est illustré par la figure 6.9: si le sous-graphe Adj reconnaît epsilon, on a une boucle infinie qu'Unitex ne peut pas éliminer.

La troisième possibilité de boucle infinie concerne les appels récursifs à des sous-graphes.

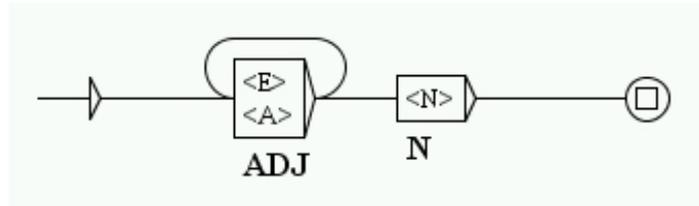


FIG. 6.8 – Boucle infinie due à une transition par le mot vide avec sortie

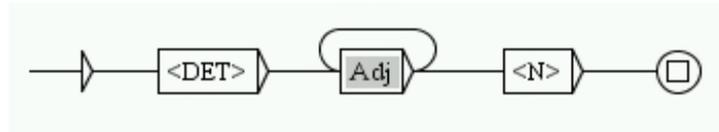


FIG. 6.9 – Boucle infinie due à un appel à un sous-graphe reconnaissant epsilon

Considérons les graphes `Det` et `DetCompose` de la figure 6.10. Chacun de ces graphes peut appeler l'autre *sans rien lire dans le texte*. Le fait qu'aucun des deux graphes ne comporte d'étiquette entre l'état initial et l'appel à l'autre graphe est capital. En effet, s'il y avait au moins une étiquette différente d'épsilon entre le début du graphe `Det` et l'appel à `DetCompose`, cela signifierait que les programmes d'Unitex explorant le graphe `Det` devraient lire le motif décrit par cette étiquette dans le texte avant d'appeler récursivement `DetCompose`. Dans ce cas, les programmes ne pourraient boucler indéfiniment que s'ils rencontraient une infinité de fois le motif dans le texte, ce qui ne peut pas arriver.

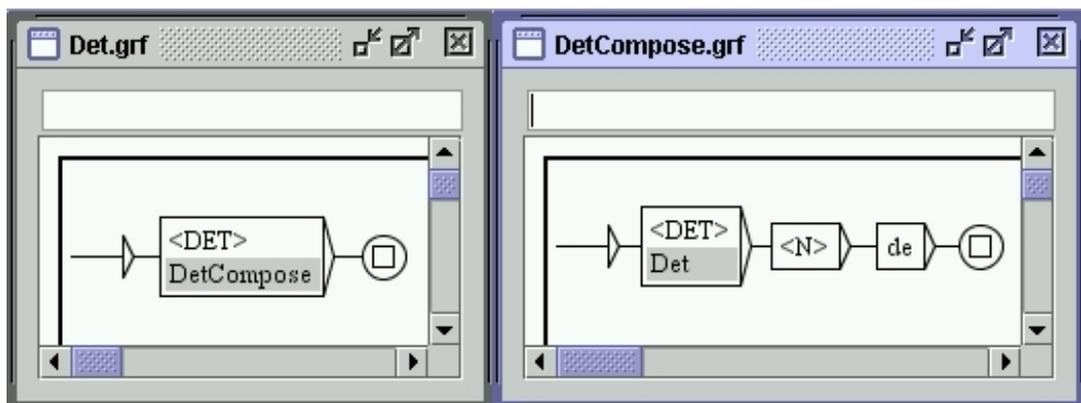


FIG. 6.10 – Boucle infinie due à deux graphes s'appelant mutuellement

6.2.4 Détection d'erreurs

Pour éviter aux programmes de se bloquer ou de planter, Unitex effectue automatiquement une détection d'erreurs lors de la compilation des graphes. Le compilateur de graphes vérifie que le graphe principal ne reconnaît pas le mot vide et recherche toutes les formes de boucles infinies. Si une erreur est trouvée, un message d'erreur apparaît dans la fenêtre de compilation. La figure 6.11 montre le message obtenu lorsqu'on tente de compiler le graphe `Det` de la figure 6.10.

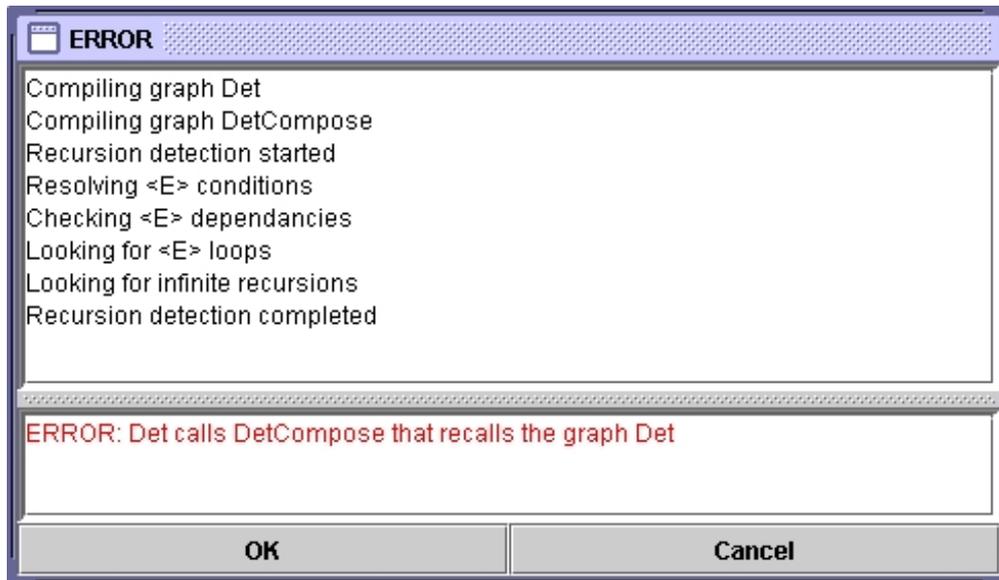


FIG. 6.11 – Message d'erreur obtenu en compilant le graphe `Det`

Si vous avez lancé une recherche de motifs en sélectionnant un graphe au format `.grf` et qu'Unitex y décèle une erreur, l'opération de recherche sera automatiquement interrompue.

6.3 Exploration des chemins d'une grammaire

Il est possible de générer les chemins reconnus par une grammaire, par exemple pour vérifier qu'elle engendre correctement les formes attendues. Pour cela, ouvrez le graphe principal de votre grammaire, et assurez que la fenêtre du graphe est bien la fenêtre active (la fenêtre active possède une barre de titre bleu, tandis que les fenêtres inactives ont une barre de titre grise). Allez ensuite dans le menu "FSGraph", puis dans le sous-menu "Tools", et cliquez sur "Explore graph paths". La fenêtre de la figure 6.12 apparaît alors.

Le cadre supérieur contient le nom du graphe principal de la grammaire à explorer. Les options suivantes concernent la gestion des sorties de la grammaire:

- "Ignore outputs": les sorties sont ignorées;
- "Separate inputs and outputs": les sorties sont affichées groupées après les entrées (a b c / A B C);

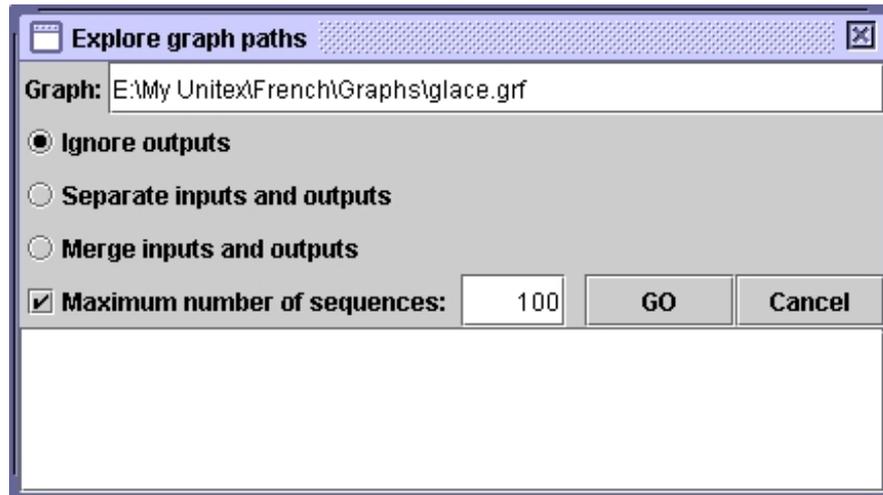


FIG. 6.12 – *Exploration des chemins d'une grammaire*

- "Merge inputs and outputs": chaque sortie est affichée immédiatement après l'entrée qui lui correspond (a/A b/B c/C).

Si l'option "Maximum number of sequences" est cochée, le nombre spécifié sera le nombre maximum de chemins générés. Si l'option n'est pas sélectionnée, tous les chemins seront générés.

Voici ce que l'on obtient pour le graphe de la figure 6.13 avec les paramètres par défaut (ignorer les sorties, limite = 100 chemins):

```

<NB> <boule> de glace à la pistache
<NB> <boule> de glace à la fraise
<NB> <boule> de glace à la vanille
<NB> <boule> de glace vanille
<NB> <boule> de glace fraise
<NB> <boule> de glace pistache
<NB> <boule> de pistache
<NB> <boule> de fraise
<NB> <boule> de vanille
glace à la pistache
glace à la fraise
glace à la vanille
glace vanille
glace fraise
glace pistache
  
```

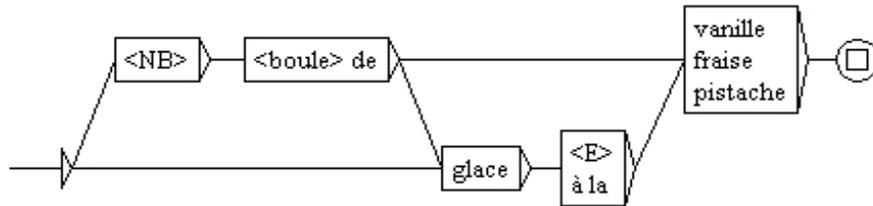


FIG. 6.13 – Exemple de graphe

6.4 Collection de graphes

Il peut arriver que l'on souhaite appliquer plusieurs grammaires situées dans un même répertoire. Pour cela, il est possible de construire automatiquement une grammaire à partir d'une arborescence de fichiers. Supposons par exemple que l'on ait l'arborescence suivante:

- *Dicos*:
 - *Banque*:
 - `carte.grf`
 - *Nourriture*:
 - `eau.grf`
 - `pain.grf`
 - `truc.grf`

Si l'on veut rassembler toutes ces grammaires en une seule, on peut le faire avec la commande "Build Graph Collection" dans le sous-menu "FSGraph > Tools". On configure cette opération au moyen de la fenêtre de la figure 6.14.



FIG. 6.14 – Construction d'une collection de graphes

Dans le champ "Source directory", sélectionnez le répertoire racine que vous voulez explorer (dans notre exemple, le répertoire *Dicos*). Dans le champ "Resulting GRF grammar", indiquez le nom de la grammaire produite. ATTENTION: ne placez pas la grammaire de sortie dans l'arborescence que vous voulez explorer, car dans ce cas, le programme va chercher lire et à écrire simultanément dans ce fichier, ce qui provoquera un plantage.

Lorsque vous cliquerez sur "OK", le programme recopiera les graphes dans le répertoire de la grammaire de sortie, et créera des sous-graphes correspondant aux différents sous-répertoires, comme on peut le voir sur la figure 6.15, qui montre le graphe de sortie engendré pour notre exemple. On peut constater qu'une boîte contient les appels à des sous-graphes correspondant à des sous-répertoires (ici les répertoires *Banque* et *Nourriture*), et que l'autre boîte fait appel à tous les graphes qui se trouvaient dans le répertoire (ici le graphe `truc.grf`).

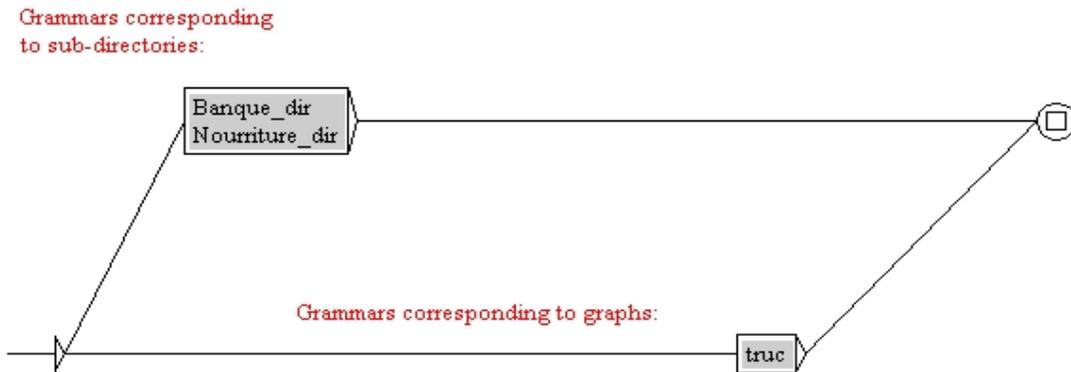


FIG. 6.15 – Graphe principal d'une collection de graphes

6.5 Règles d'application des transducteurs

Cette section décrit les règles d'application des transducteurs lors des opérations de pré-traitement et de recherche de motifs. Les graphes de flexion et de normalisation de formes ambiguës ne sont pas concernés par ce qui suit.

6.5.1 Insertion à gauche du motif reconnu

Lorsqu'un transducteur est appliqué en mode REPLACE, les sorties remplacent les séquences lues dans le texte. En mode MERGE, les sorties sont insérées à gauche des séquences reconnues. Considérons le transducteur de la figure 6.16.

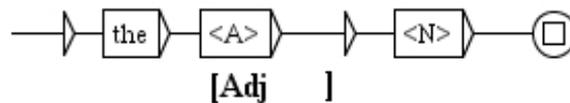


FIG. 6.16 – Exemple de transducteur

Si l'on applique ce transducteur au roman *Ivanhoe* de Sir Walter Scott en mode MERGE, on obtient la concordance suivante:

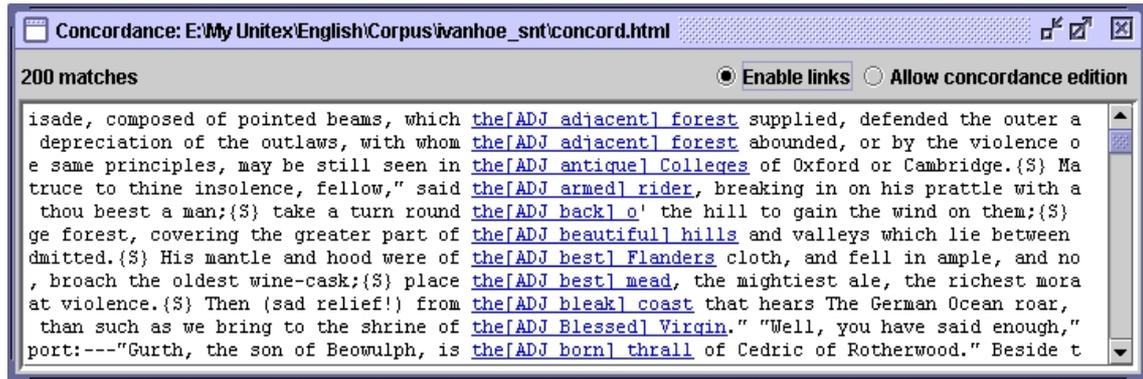


FIG. 6.17 – Concordance obtenue en mode MERGE avec le transducteur de la figure 6.16

6.5.2 Application en avançant

Pendant les opérations de prétraitement, le texte est modifié au fur et à mesure qu'il est parcouru. Afin d'éviter le risque de boucler indéfiniment, il ne faut pas que les séquences produites par un transducteur puissent être ré-analysées par celui-ci. Pour cette raison, quand une séquence a été introduite dans le texte, l'application du transducteur se poursuit après cette séquence.

Cette règle ne concerne que les transducteurs de prétraitement, car lors de l'application de graphes syntaxiques, les sorties ne modifient pas le texte parcouru mais un fichier de concordances distinct du texte.

6.5.3 Priorité à gauche

Lors de l'application d'une grammaire locale, les occurrences qui se chevauchent sont toutes indexées. Lors de la construction de la concordance, toutes ces occurrences sont présentées (voir figure 6.18).

```
red by the river Don, there extended in [ancient times] a large forest, covering the greater part
tered by the river Don, there extended [in ancient] times a large forest, covering the greater
he river Don, there extended in ancient [times a] large forest, covering the greater part of th
```

FIG. 6.18 – Occurrences se chevauchant dans une concordance

En revanche, si vous modifiez le texte au lieu de construire une concordance, il est nécessaire de choisir parmi ces occurrences lesquelles seront prises en compte. Pour cela, Unitex applique la règle de priorité suivante: la séquence la plus à gauche l'emporte.

Si l'on applique cette règle aux trois occurrences de la concordance précédente, l'occurrence [in ancient] est concurrente avec [ancient times]. C'est donc la première qui est retenue car c'est l'occurrence la plus à gauche, et [ancient times] est éliminée. L'occurrence suivante [times a] n'est donc plus en conflit avec [ancient times] et peut donc apparaître dans le résultat:

```
...Don, there extended [in ancient] [times a] large forest...
```

La règle de priorité à gauche s'applique uniquement lorsque le texte est modifié, soit lors du prétraitement, soit après l'application d'un graphe syntaxique (voir section 6.6.3).

6.5.4 Priorité aux séquences les plus longues

Lors de l'application d'un graphe syntaxique, il est possible de choisir si la priorité doit être donnée aux séquences les plus courtes ou les plus longues, ou si toutes les séquences doivent être retenues. Lors des opérations de prétraitement, la priorité est toujours donnée aux séquences les plus longues.

6.5.5 Sorties à variables

Comme nous l'avons vu à la section 5.2.6, il est possible d'utiliser des variables pour stocker le texte qui a été analysé par une grammaire. Ces variables peuvent être utilisées dans les graphes de prétraitement et dans les graphes syntaxiques.

Vous devez donner des noms aux variables que vous utilisez. Ces noms peuvent contenir les lettres comprises entre A et Z non accentuées minuscules ou majuscules, des chiffres et le caractère _ (underscore).

Pour définir le début (ou la fin) de la zone stockée dans une variable, vous devez créer une boîte contenant le nom de la variable encadré par les caractères \$ et ((\$ et) pour la fin d'une variable). Pour utiliser une variable dans une sortie, vous devez faire précéder son nom du caractère \$ (voir figure 6.19).

Les variables sont globales. Cela signifie que vous pouvez définir une variable dans un graphe et l'appeler dans un autre, comme l'illustrent les graphes de la figure 6.19:

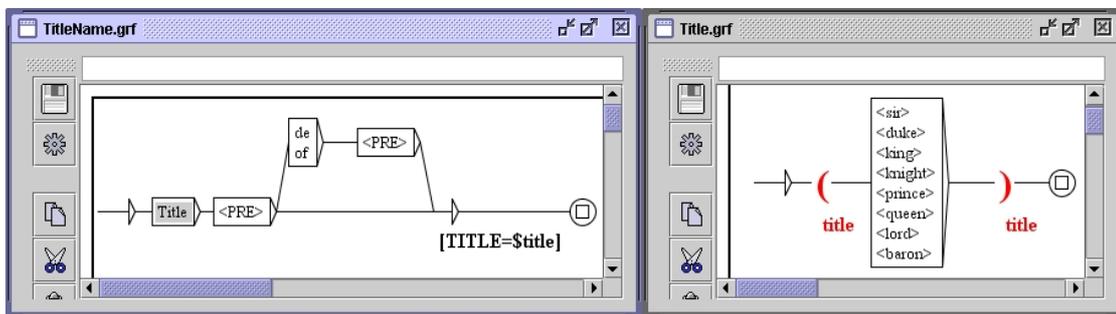


FIG. 6.19 – Définition d'une variable dans un sous-graphe

Si on applique le graphe `TitleName` en mode MERGE au texte *Ivanhoe*, on obtient la concordance suivante:

Les sorties à variables peuvent être utilisées pour déplacer des groupes de mots. En effet, l'application d'un transducteur en mode REPLACE n'écrit dans le texte que les séquences produites par des sorties. Pour inverser deux groupes de mots, il suffit donc de les stocker dans des variables et de produire une sortie avec ces variables dans l'ordre souhaité. Ainsi, le transducteur de la figure 6.21 appliqué en mode REPLACE au texte *Ivanhoe* donne la concordance de la figure 6.22.

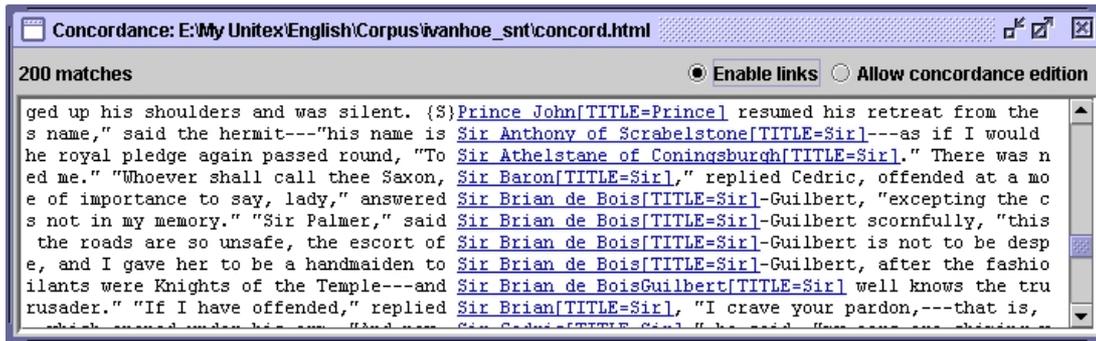
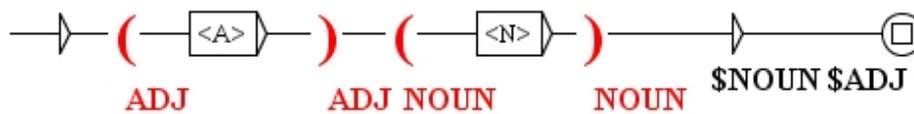
FIG. 6.20 – Concordance obtenue par l'application du graphe *TitleName*

FIG. 6.21 – Inversion de mots grâce à l'utilisation de deux variables

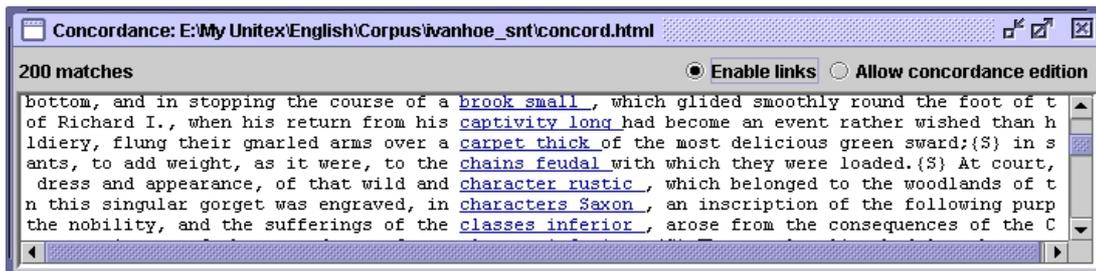


FIG. 6.22 – Résultat de l'application du transducteur de la figure 6.21

La présence d'un espace à droite de chaque occurrence dans la concordance de la figure 6.22 est due à l'insertion d'un espace après \$NOUN \$ADJ dans la sortie. Sans cet espace, le résultat de la sortie serait collé à son contexte droit (voir figure 6.23).

En effet, le programme `Locate` considère toujours la possibilité d'un espace facultatif entre deux boîtes. Dans le cas présent, le programme essaye de lire un espace entre la boîte marquant la fin de la variable `NOUN` et la boîte contenant la sortie. Si un espace est lu de la sorte en mode `REPLACE`, il est effacé car il fait partie du texte analysé par la grammaire. Pour éviter de perdre cet espace, il faut donc le réinsérer en le mettant dans une sortie.

Si le début ou la fin d'une variable est mal définie (fin d'une variable avant son début, absence du début ou de la fin d'une variable), celle-ci sera ignorée lors des sorties.

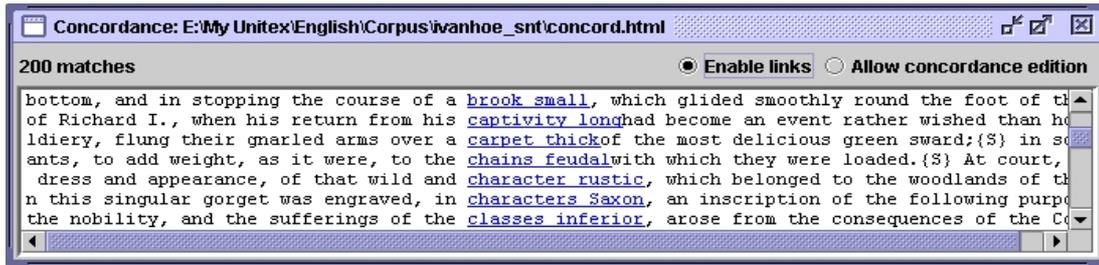


FIG. 6.23 – Problème d'espacement en mode REPLACE

Il n'y a aucune limitation du nombre de variables utilisables.

Les variables peuvent être imbriquées, voire même se chevaucher comme le montre la figure 6.24:

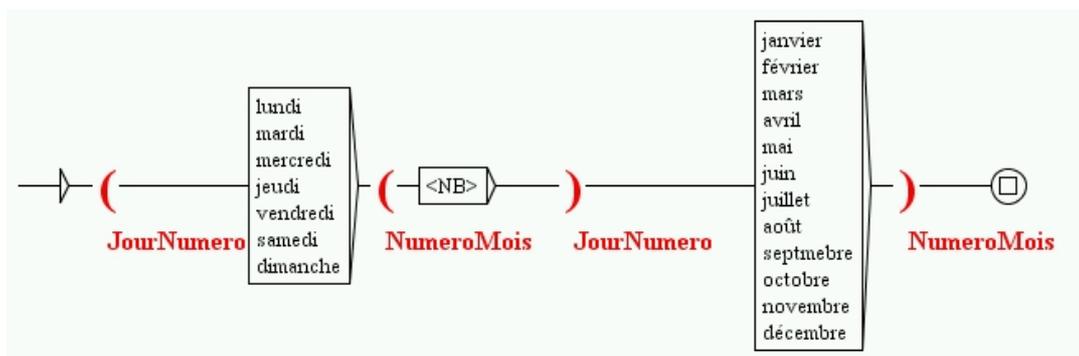


FIG. 6.24 – Chevauchement de variables

6.6 Application des graphes aux textes

Cette section concerne uniquement les graphes syntaxiques.

6.6.1 Configuration de la recherche

Pour appliquer un graphe à un texte, vous devez ouvrir le texte, puis cliquer sur "Locate Pattern..." dans le menu "Text" ou appuyer sur <Ctrl+L>. Vous pouvez alors configurer votre recherche grâce à la fenêtre de la figure 6.25.

Dans le cadre intitulé "Locate pattern in the form of", choisissez "Graph" et sélectionnez votre graphe en cliquant sur le bouton "Set". Vous pouvez choisir un graphe au format .grf (Unicode Graphs) ou un graphe compilé au format .fst2 (Unicode Compiled Graphs). Si

vosre graphe est au format `.grf`, Unitex le compilera automatiquement avant de lancer la recherche.

Le cadre "Index" permet de sélectionner le mode de reconnaissance:

- "Shortest matches": donne la priorité aux séquences les plus courtes;
- "Longest matches": donne la priorité aux séquences les plus longues. C'est le mode utilisé par défaut;
- "All matches": donne toutes les séquences reconnues.

Le cadre "Search limitation" permet de limiter ou non la recherche à un certain nombre d'occurrences. Par défaut, la recherche est limitée aux 200 premières occurrences.

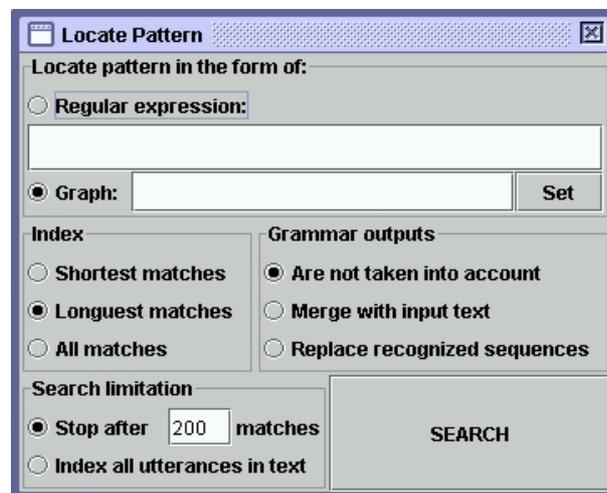


FIG. 6.25 – Fenêtre de recherche d'expressions

Le cadre "Grammar outputs" concerne le mode d'utilisation des sorties. Le mode "Merge with input text" permet d'insérer les séquences produites par les sorties. Le mode "Replace recognized sequences" permet de remplacer les séquences reconnues par les séquences produites. Le troisième mode ignore les sorties. Ce dernier mode est utilisé par défaut.

Une fois vos paramètres fixés, cliquez sur "SEARCH" pour lancer la recherche.

6.6.2 Concordance

Le résultat de la recherche est un fichier d'index contenant les positions de toutes les occurrences trouvées. La fenêtre de la figure 6.26 vous propose de construire une concordance ou de modifier le texte.

Pour afficher une concordance, vous devez cliquer sur le bouton "Build concordance". Vous pouvez paramétrer la taille des contextes gauche et droit en caractères. Vous pouvez également choisir le mode de tri qui sera appliqué aux lignes de la concordance grâce au menu "Sort According to". Pour plus de détails sur les paramètres de construction de la concordance, reportez-vous à la section 4.8.2

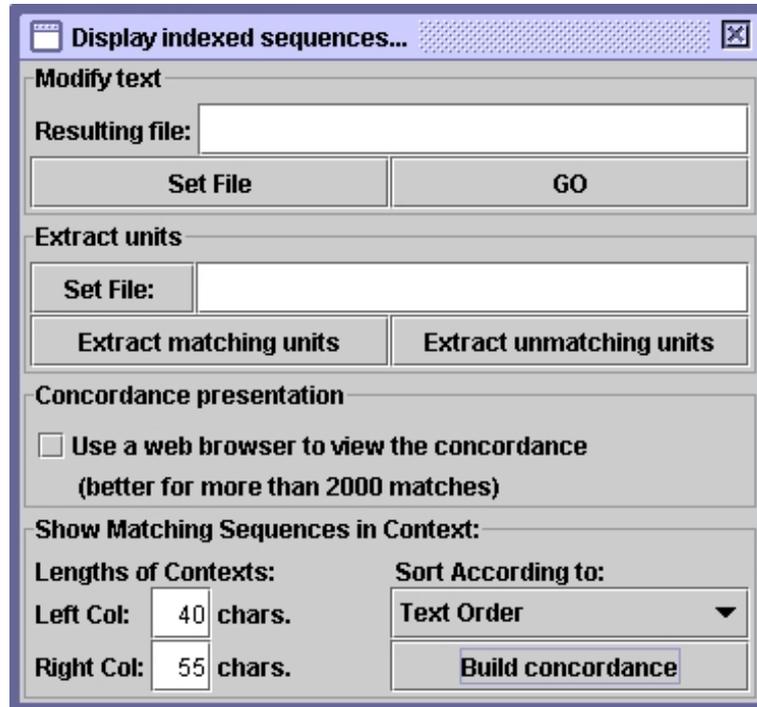


FIG. 6.26 – Configuration de l'affichage des occurrences trouvées

La concordance est produite sous la forme d'un fichier HTML. Vous pouvez paramétrer Unitex pour que les concordances soient lues à l'aide d'un navigateur Web (voir section 4.8.2).

Si vous affichez les concordances avec la fenêtre proposée par Unitex, vous pouvez accéder à la séquence reconnue dans le texte en cliquant sur l'occurrence. Si la fenêtre du texte n'est pas iconifiée et que le texte n'est pas trop long pour être affiché, vous verrez apparaître la séquence sélectionnée (voir figure 6.27).

De plus, si l'automate du texte a été construit et que la fenêtre correspondante n'est pas iconifiée, le fait de cliquer sur une occurrence sélectionne l'automate de la phrase qui contient cette occurrence.

6.6.3 Modification du texte

Vous pouvez choisir de modifier le texte au lieu de construire une concordance. Pour cela, sélectionnez un nom de fichier dans le cadre "Modify text" de la fenêtre de la figure 6.26. Ce fichier doit porter l'extension `.txt`.

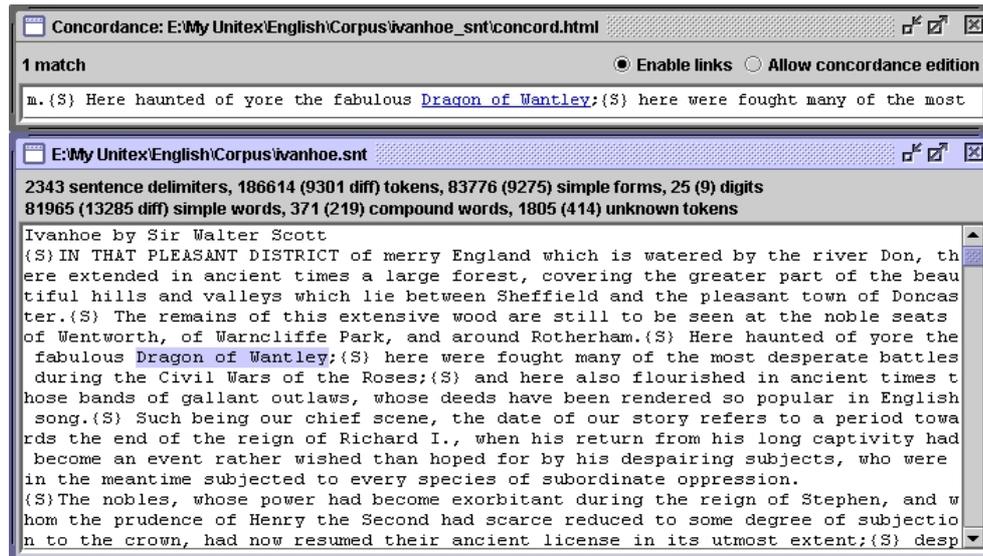


FIG. 6.27 – Sélection d'une occurrence dans le texte

Si vous souhaitez modifier le texte courant, il faut choisir le fichier `.txt` correspondant. Si vous choisissez un autre nom de fichier, le texte courant ne sera pas affecté. Cliquez sur le bouton "GO" pour lancer la modification du texte. Les règles de priorités appliquées lors de cette opérations sont détaillées à la section 3.6.2.

Une fois cette opération effectuée, le fichier résultant est une copie du texte dans laquelle les sorties ont été prises en compte. Les opérations de normalisation et de découpage en unités lexicales sont automatiquement appliquées à ce fichier texte. Les dictionnaires du texte existants ne sont pas modifiés. Ainsi, si vous avez choisi de modifier le texte courant, les modifications sont immédiatement effectives. Vous pouvez alors lancer de nouvelles recherches sur le texte.

ATTENTION: si vous avez choisi d'appliquer votre graphe en ignorant les sorties, toutes les occurrences seront effacées du texte.

6.6.4 Extraction des occurrences

Vous pouvez extraire toutes les phrases du texte qui contiennent ou non des occurrences. Pour cela, choisissez un nom de fichier de sortie grâce au bouton "Set File" dans le cadre "Extract units" (figure 6.26). Cliquez ensuite sur un des boutons "Extract matching units" ou "Extract unmatching units" selon que vous voulez extraire les phrases contenant les occurrences ou non.

Chapitre 7

Automate du texte

Les langues naturelles contiennent beaucoup d'ambiguïtés lexicales. L'automate du texte est un moyen efficace et visuel de représenter ces ambiguïtés. Chaque phrase du texte est représentée par un automate dont les chemins expriment toutes les interprétations possibles.

Ce chapitre présente les automates de texte, le détail de leur construction ainsi que les opérations qui peuvent leur être appliquées, en particulier la levée d'ambiguïtés au moyen du programme ELAG ([34]). Il n'est pour l'instant pas possible d'effectuer de recherche de motifs sur l'automate du texte.

7.1 Présentation

L'automate du texte permet d'exprimer toutes les interprétations lexicales possibles des mots. Ces différentes interprétations sont les différentes entrées présentes dans les dictionnaires du texte. La figure 7.1 montre l'automate de la quatrième phrase du texte *Ivanhoe*.

On peut voir sur la figure 7.1 que le mot **Here** possède ici trois interprétations (adjectif, adverbe et nom), **haunted** deux (adjectif et verbe), etc. Toutes les combinaisons possibles sont exprimées, car chaque interprétation de chaque mot est reliée à toutes les interprétations des mots suivant et précédent.

En cas de concurrence entre un mot composé et une séquence de mots simples, l'automate contient un chemin étiqueté par le mot composé, parallèle aux chemins exprimant les combinaisons de mots simples. Ceci est illustrée par la figure 7.2, où le mot composé **courts of law** est concurrent avec une combinaison de mots simples.

Par construction, l'automate du texte ne contient pas de boucle. On dit que l'automate du texte est *acyclique*.

NOTE: le terme automate du texte est un abus de langage. En effet, il y a en réalité un automate pour chaque phrase du texte. Cependant, la concaténation de tous ces automates correspondrait à l'automate de tout le texte. On utilise donc le terme automate du texte même si l'on ne manipule pas réellement cet objet pour des raisons pratiques.

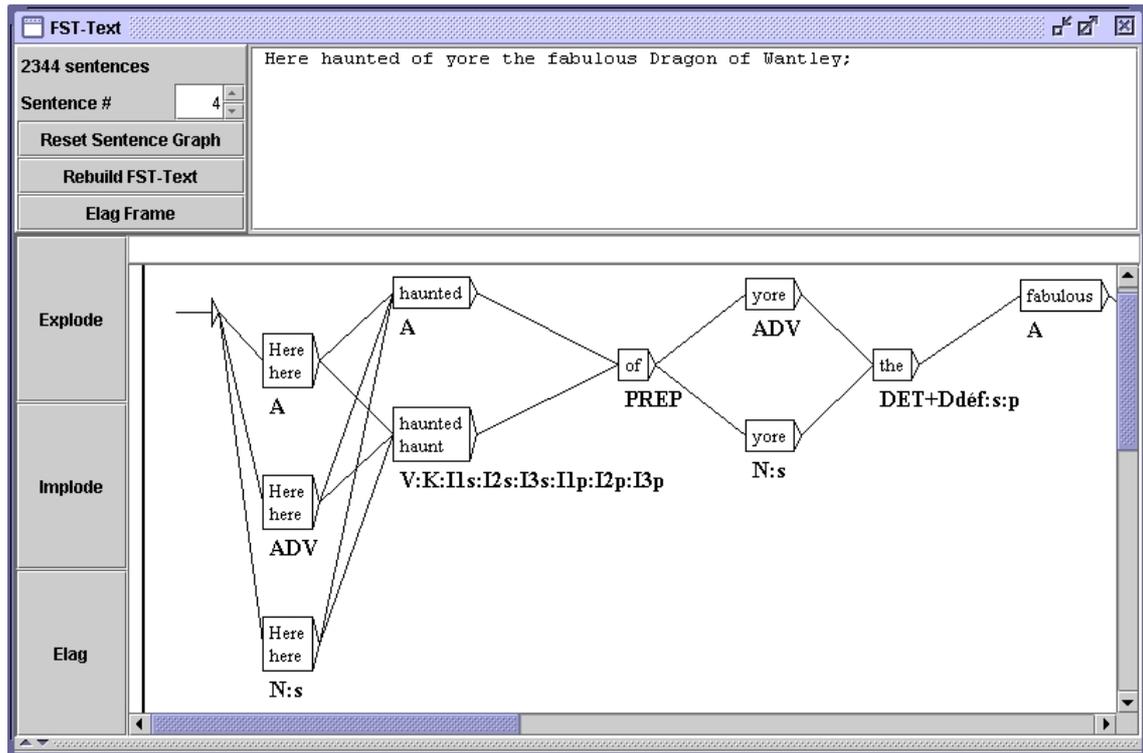


FIG. 7.1 – Exemple d'automate de phrase

7.2 Construction

Pour construire l'automate d'un texte, vous devez ouvrir ce texte, puis cliquer sur "Construct FST-Text..." dans le menu "Text". Il est recommandé d'avoir découpé le texte en phrases et de lui avoir appliqué les dictionnaires. Si vous n'avez pas découpé le texte en phrases, le programme de construction découpera arbitrairement le texte en séquences de 2000 unités lexicales au lieu de construire un automate par phrase. Si vous n'avez pas appliqué les dictionnaires, les automates de phrase que vous obtiendrez ne seront constitués que d'un seul chemin ne comportant que des mots inconnus.

7.2.1 Règles de construction de l'automate du texte

Les automates de phrase sont construits à partir des dictionnaires du texte. Le degré d'ambiguïté obtenu est donc directement lié à la finesse de description des dictionnaires utilisés. Sur l'automate de phrase de la figure 7.3, on peut voir que le mot **which** a été codé deux fois comme déterminant dans deux sous-catégories de la catégorie DET. Cette finesse de description ne sera d'aucune utilité si l'on ne s'intéresse qu'à la catégorie grammaticale de ce mot. Il faut donc adapter la finesse des dictionnaires à l'utilisation recherchée.

Pour chaque unité lexicale de la phrase, Unitex recherche toutes ses interprétations possibles dans le dictionnaire des mots simples du texte. On recherche ensuite toutes les suites

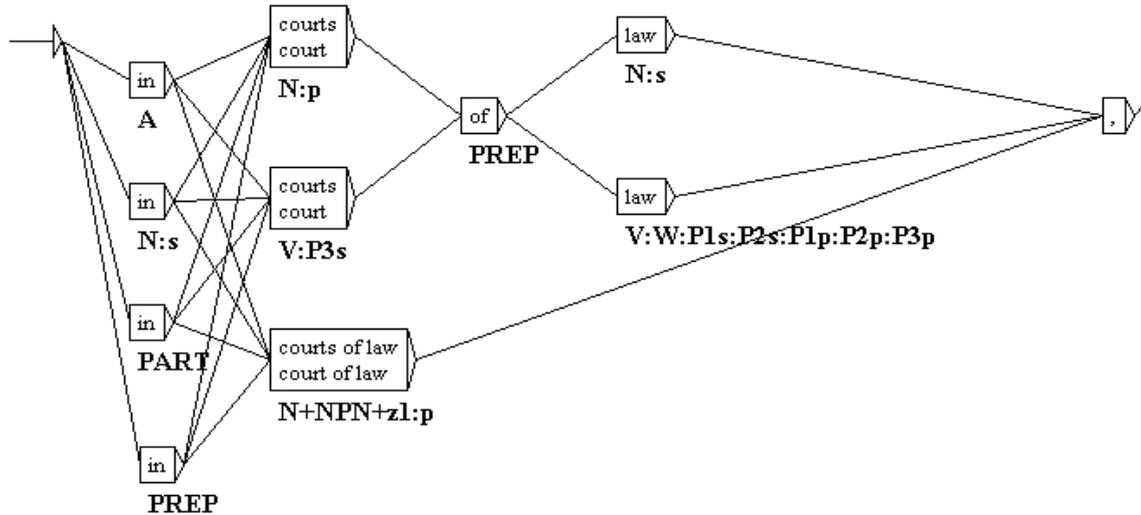


FIG. 7.2 – *Concurrence entre un mot composé et une combinaison de mots simples*

d'unités lexicales qui ont une interprétation dans le dictionnaire des mots composés du texte. Toutes les combinaisons de ces interprétations forment l'automate de la phrase.

NOTE: quand le texte contient des étiquettes lexicales (*i.e.* {aujourd'hui, .ADV}), ces étiquettes sont reproduites à l'identique dans l'automate, sans que le programme essaye de décomposer les séquences qu'elles représentent.

Dans chaque boîte, la 1^{ère} ligne contient la forme fléchée trouvée dans le texte, et la 2^{ème} ligne contient la forme canonique si elle est différente. Les autres informations sont codées sous la boîte (voir section 7.4.1).

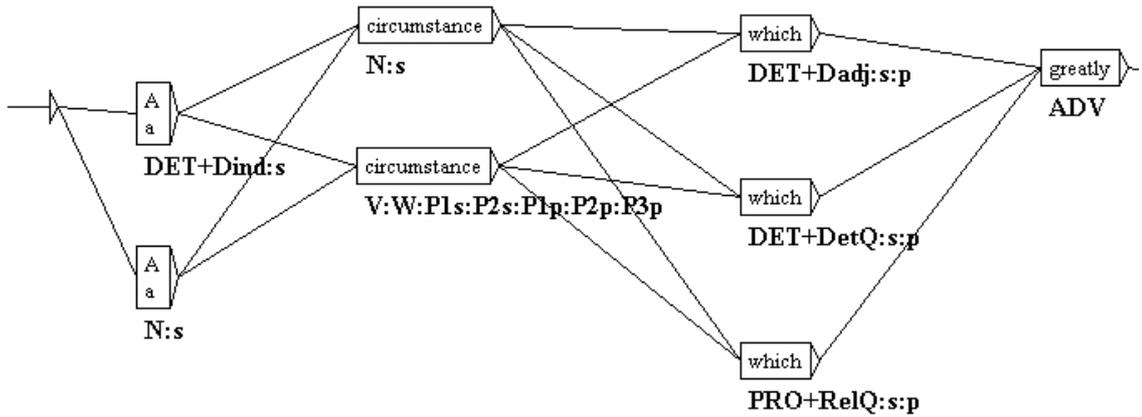
Les espaces séparant les unités lexicales ne sont pas retranscrits dans l'automate, à l'exception des espaces à l'intérieur de mots composés.

La casse des unités lexicales est conservée. Par exemple, si l'on trouve le mot **Here**, on conserve la majuscule (voir figure 7.1). Ce choix permet de ne pas perdre cette information lors du passage à l'automate du texte, ce qui pourra être utile pour des applications où la casse est importante, telle que la reconnaissance des noms propres.

7.2.2 Normalisation de formes ambiguës

Lors de la construction de l'automate, il est possible d'effectuer une normalisation de formes ambiguës en appliquant une grammaire de normalisation. Cette grammaire doit se nommer `Norm.fst2` et doit être placée dans votre répertoire personnel, dans le sous-répertoire `/Graphs/Normalization` de la langue voulue. Les grammaires de normalisation de formes ambiguës sont décrites à la section 6.1.3.

Si une séquence du texte est reconnue par la grammaire de normalisation, toutes les interprétations décrites par la grammaire sont insérées dans l'automate du texte. La figure

FIG. 7.3 – Double entrée pour *which* en tant que déterminant

7.4 montre l'extrait de la grammaire utilisée pour le français qui explicite l'ambiguïté de la séquence 1'.



FIG. 7.4 – Normalisation de la séquence 1'

Si l'on applique cette grammaire à une phrase française contenant la séquence 1', on obtient un automate de phrase similaire à celui de la figure 7.5.

Dans l'automate obtenu, on peut voir que les quatre règles de réécriture de la séquence 1' ont été appliquées, ce qui a ajouté quatre étiquettes dans l'automate. Ces étiquettes sont concurrentes avec les deux chemins préexistants pour la séquence 1'. La normalisation à la construction de l'automate du texte permet d'ajouter des chemins à l'automate, pas d'en supprimer. Lorsque la fonctionnalité de levée d'ambiguïtés sera disponible, elle permettra d'éliminer les chemins qui sont devenus superflus.

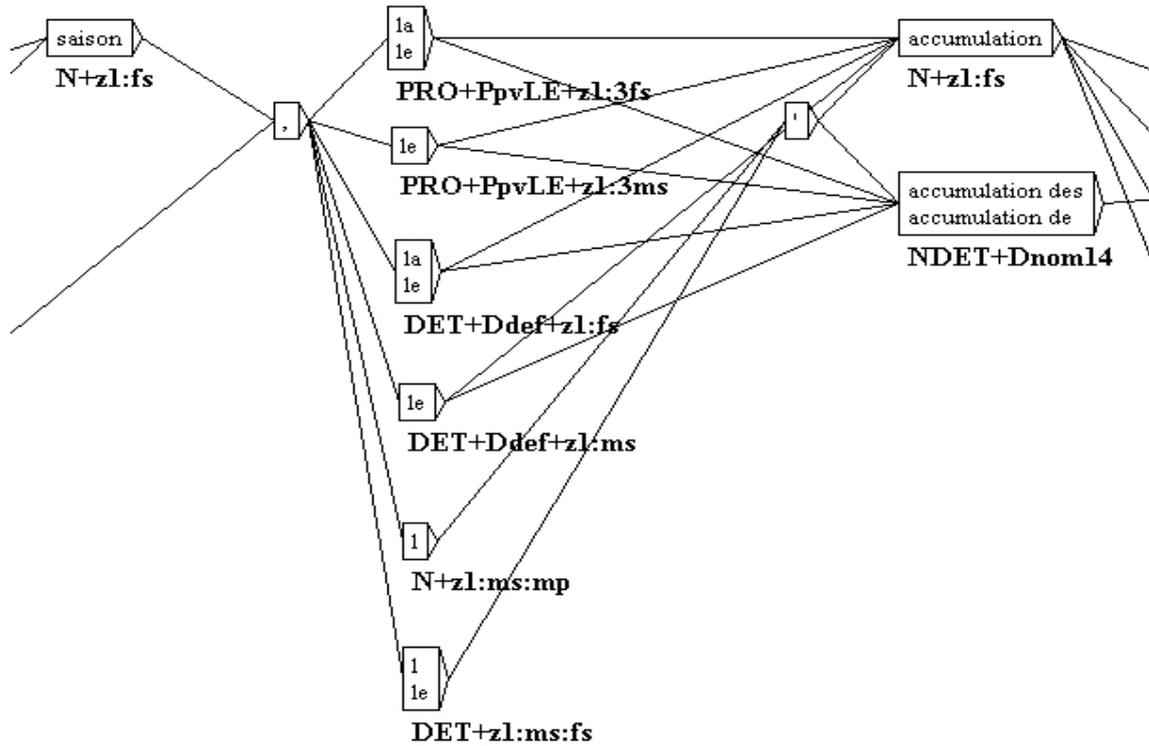


FIG. 7.5 – Automate normalisé avec la grammaire de la figure 7.4

7.2.3 Normalisation des pronoms clitiques en portugais

En portugais, les verbes au futur et au conditionnel peuvent être modifiés par l'insertion d'un ou deux pronoms clitiques entre le radical et le suffixe du verbe. Par exemple, la séquence *dir-me-ão* (*ils me diront*), correspond à la forme verbale complète *dirão*, associée au pronom *me*. En vue de pouvoir effectuer des manipulations sur cette forme réécrite, il est nécessaire de l'introduire dans l'automate du texte, en parallèle de la séquence d'origine. Ainsi, l'utilisateur pourra rechercher l'une ou l'autre forme selon ses besoins. Les figures 7.6 et 7.7 montrent l'automate d'une phrase avant et après normalisation des clitiques.

Le programme *Reconstrucao* permet de construire dynamiquement pour chaque texte une grammaire de normalisation de ces formes. La grammaire ainsi produite peut alors être utilisée pour normaliser l'automate du texte. La fenêtre de configuration de construction de l'automate propose l'option "Build clitic normalization grammar" (voir figure 7.10). Cette option lance automatiquement la construction de la grammaire de normalisation, qui est ensuite utilisée pour construire l'automate du texte, si vous avez sélectionné l'option "Apply the Normalization grammar".

7.2.4 Conservation des meilleurs chemins

Il peut arriver qu'un mot inconnu vienne parasiter l'automate du texte en étant concurrent avec une séquence complètement étiquetée. Ainsi, dans l'automate de phrase de la figure 7.8,

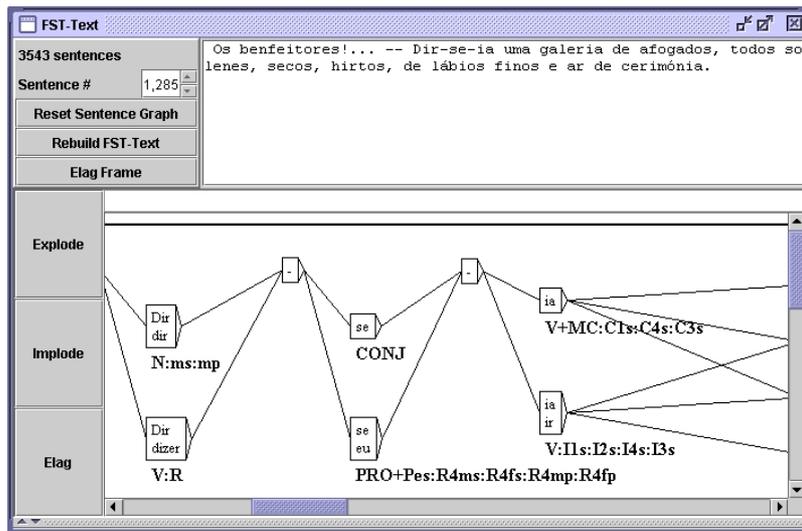


FIG. 7.6 – Automate de phrase non normalisé

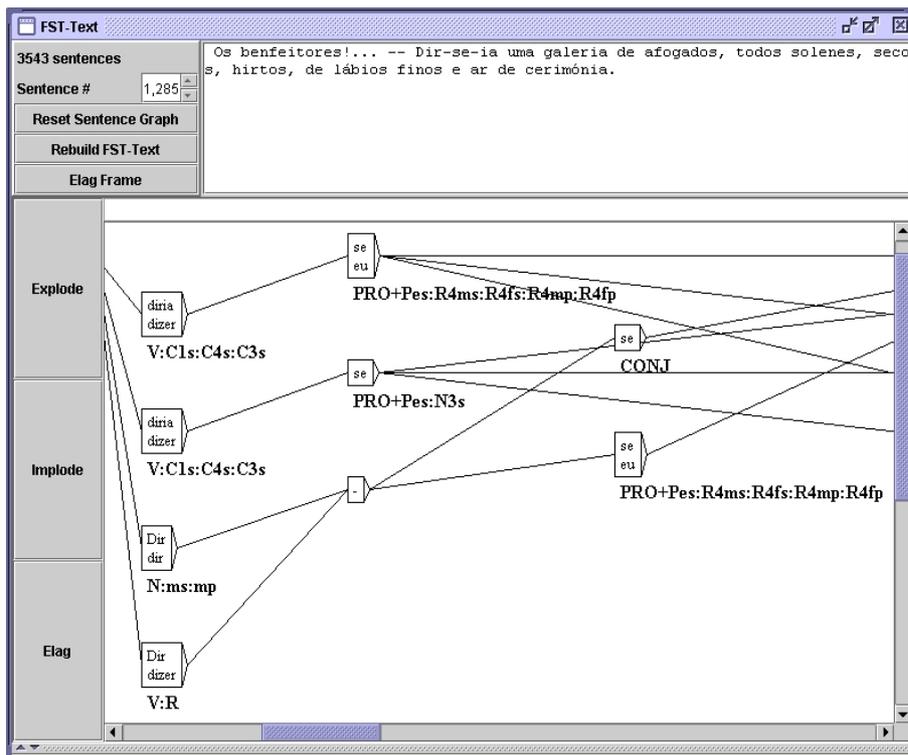


FIG. 7.7 – Automate de phrase normalisé

on peut voir que l'adverbe aujourd'hui est concurrencé par le mot inconnu aujourd, suivi d'une apostrophe et du participe passé du verbe huir.

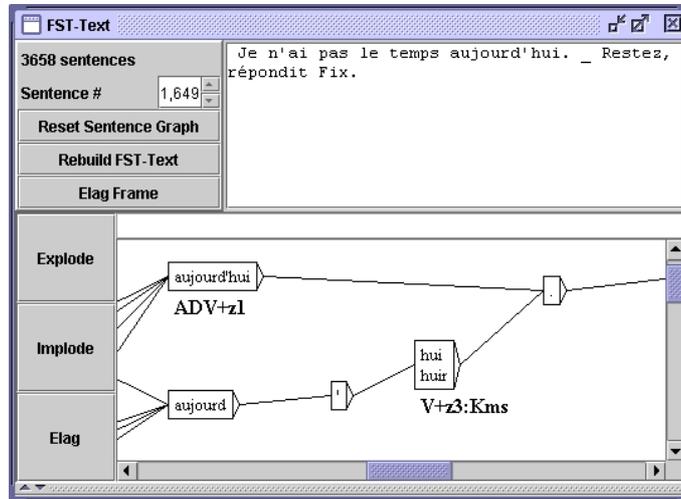


FIG. 7.8 – Ambiguïté due à une séquence contenant un mot inconnu

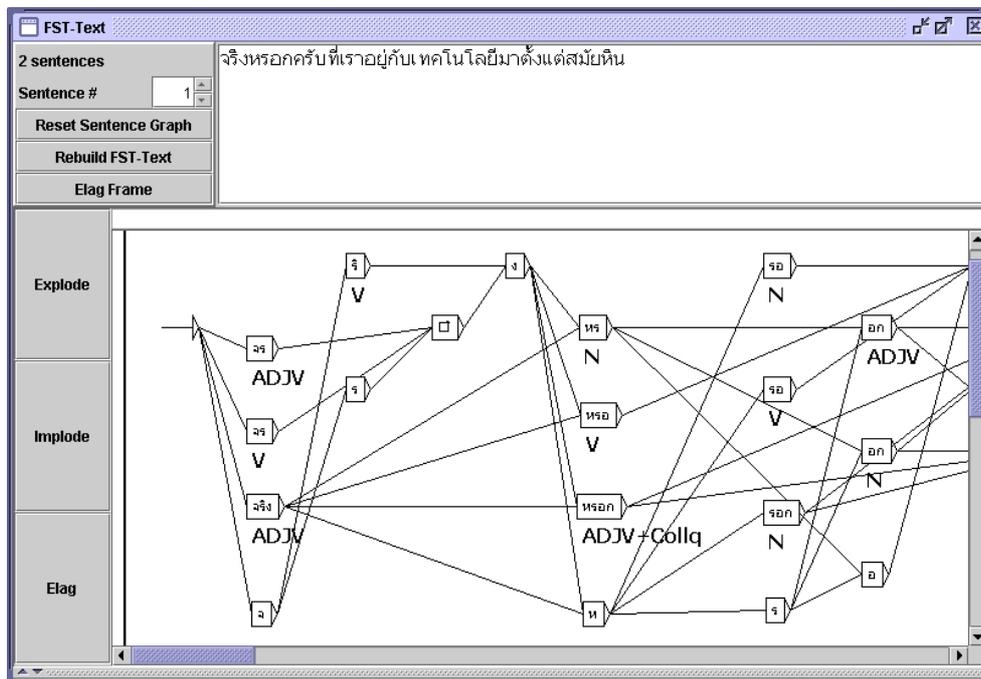


FIG. 7.9 – Automate d'une phrase thai

On trouve également ce phénomène dans le traitement de certaines langues asiatiques comme le thaï. Quand les mots ne sont pas délimités, il n'y a pas d'autre solution que d'envisager toutes les combinaisons possibles, ce qui entraîne la création de nombreux chemins comportant des mots inconnus qui s'entremêlent avec les chemins étiquetés. La figure 7.9 montre un exemple d'un tel automate de phrase en thaï.

Il est possible de supprimer ces chemins parasites. Pour cela, il faut sélectionner l'option "Clean Text FST" dans la fenêtre de configuration de la construction de l'automate du texte (voir figure 7.10). Cette option indique au programme de construction de l'automate qu'il doit nettoyer chaque automate de phrase.

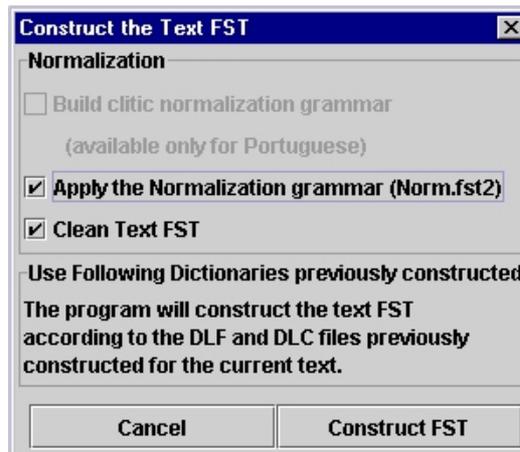


FIG. 7.10 – Configuration de la construction de l'automate du texte

Ce nettoyage s'effectue selon le principe suivant: si plusieurs chemins sont en concurrence dans l'automate, le programme garde ceux qui contiennent le moins de mots inconnus. Par exemple, la séquence *aujourd'hui* en tant qu'adverbe composé l'emporte sur la décomposition en *aujourd'hui* suivi d'une apostrophe et de *hui*, car *aujourd'* est un mot inconnu, ce qui fait une forme non étiquetée contre zéro dans le cas de l'adverbe composé.

La figure 7.11 montre l'automate de la figure 7.9 après nettoyage.

7.3 Levée d'ambiguïtés lexicales avec ELAG

Le programme ELAG permet d'appliquer des grammaires de levée d'ambiguïtés sur l'automate du texte. C'est un mécanisme puissant qui permet à chacun d'écrire ses propres règles de façon indépendante des règles déjà existantes. Cette section présente rapidement le formalisme des grammaires utilisées par ELAG ainsi que le fonctionnement du programme. Pour plus de détails, le lecteur pourra se reporter à [2] et [34].

7.3.1 Grammaires de levée d'ambiguïtés

Les grammaires manipulées par ELAG ont une syntaxe particulière. Elles comportent deux parties, que nous appellerons partie *si* et partie *alors*. La partie *si* d'une grammaire ELAG

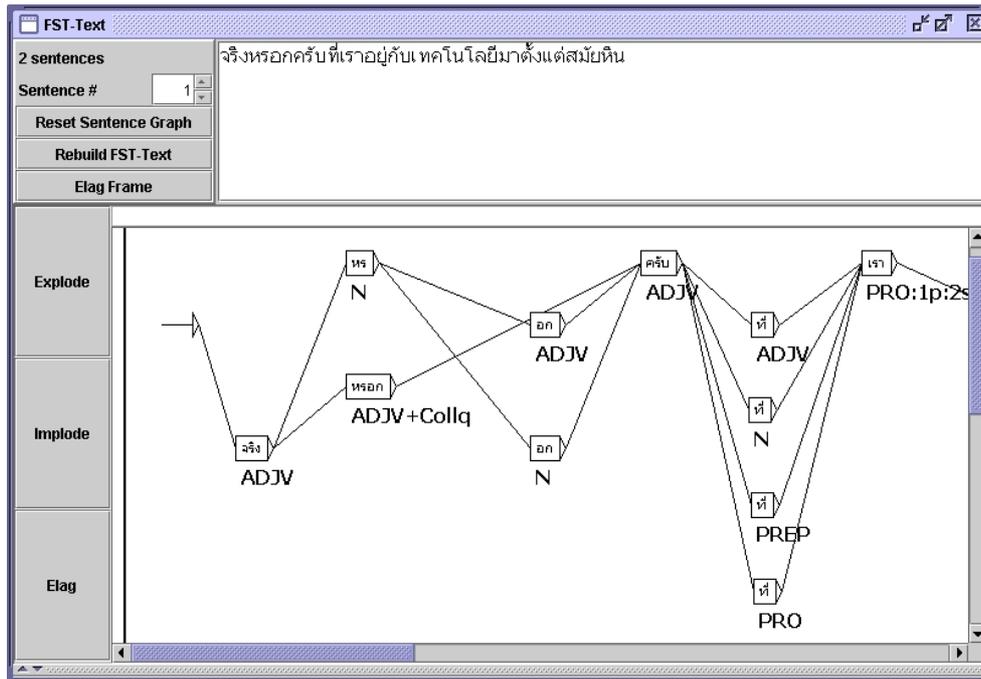


FIG. 7.11 – Automate de la figure 7.9 après nettoyage

se divise en deux zones délimitées par des boîtes contenant le symbole $\langle ! \rangle$. La partie *alors* est divisée de la même façon au moyen du symbole $\langle = \rangle$. La signification d'une grammaire est la suivante: dans l'automate du texte, si l'on trouve une séquence reconnue par la partie *si*, alors elle doit aussi être reconnue par la partie *alors* de la grammaire, faute de quoi elle sera retirée de l'automate du texte.

La figure 7.12 montre un exemple de grammaire. La partie *si* reconnaît un verbe à la deuxième personne du singulier suivi par un tiret et *tu*, soit en tant que pronom, soit en tant que participe passé du verbe *taire*. La partie *alors* impose que *tu* soit alors considéré comme pronom. La figure 7.13 montre le résultat de l'application de cette grammaire sur la phrase "*Feras-tu cela bientôt ?*". On peut voir sur l'automate du bas que le chemin correspondant à *tu* participe passé a été éliminé.

Point de synchronisation

Les parties *si* et *alors* d'une grammaire ELAG sont divisées en deux par le deuxième symbole $\langle ! \rangle$ dans la partie *si*, et par le deuxième symbole $\langle = \rangle$ dans la partie *alors*. Ces symboles forment un *point de synchronisation*. Cela permet d'écrire des règles dans lesquelles les contraintes *si* et *alors* ne sont pas nécessairement alignées, comme c'est par exemple le cas sur la figure 7.14. Cette grammaire s'interprète de la manière suivante: si on trouve un tiret suivi par *il*, *elle* ou *on*, alors ce tiret doit être précédé par un verbe, éventuellement suivi de *-t*. Ainsi, si l'on considère la phrase de la figure 7.15 commençant par *Est-il*, on peut voir que toutes les interprétations non verbales de *Est* ont été supprimées.

Si 'tu' se trouve après un verbe à la 2e personne du singulier
suivi par un tiret, alors c'est un pronom et non pas
le participe passé de 'taire'

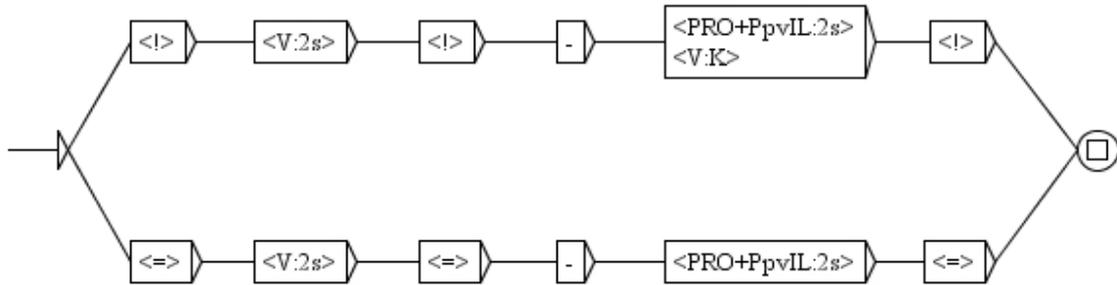


FIG. 7.12 – Exemple de grammaire ELAG

7.3.2 Compilation des grammaires ELAG

Avant de pouvoir être appliquée à un automate de texte, une grammaire ELAG doit être compilée en un fichier `.rul`. Cette opération s'effectue via la commande "Elag Rules", dans le menu "Text", qui fait apparaître la fenêtre de la figure 7.16.

Si le cadre à droite contient déjà des grammaires que vous ne souhaitez pas utiliser, vous pouvez les retirer au moyen du bouton `<<`. Sélectionnez ensuite votre grammaire dans l'explorateur de fichiers situé dans le cadre gauche, et cliquez sur le bouton `>>` pour l'ajouter à la liste du cadre droit. Cliquez alors sur le bouton *compile*. Ceci lancera le programme `ElagComp` qui va compiler la grammaire sélectionnée pour créer un fichier nommé *elag.rul*.

Si vous avez sélectionné votre grammaire dans le cadre droit, vous pouvez rechercher les motifs qu'elle reconnaît en cliquant sur le bouton *locate*. Cela ouvre la fenêtre "Locate Pattern" en spécifiant automatiquement un nom de graphe se terminant par `-conc.fst2`. Ce graphe correspond à la partie *si* de la grammaire. Vous pouvez ainsi obtenir les occurrences du texte sur lesquelles la grammaire s'appliquera.

NOTE: le fichier `-conc.fst2` utilisé pour localiser la partie *alors* d'une grammaire est généré lors de la compilation des grammaires ELAG au moyen du bouton *compile*. Il faut donc avoir d'abord compilé votre grammaire avant d'utiliser la fonction de recherche du bouton *locate*.

7.3.3 Levée d'ambiguïtés

Une fois que vous avez compilé votre grammaire en un fichier `elag.rul`, vous pouvez l'appliquer à l'automate du texte. Dans la fenêtre de l'automate du texte, cliquez sur le bouton *elag*. Une boîte de dialogue apparaîtra pour vous demander le nom du fichier `.rul` à utiliser (voir figure 7.17). Comme le fichier défaut est bien `elag.rul`, cliquez simplement sur "OK". Cela lancera le programme `Elag` qui va effectuer la levée d'ambiguïtés.

Une fois le programme terminé, vous pouvez consulter l'automate résultat en cliquant sur le bouton *Elag Frame*. Comme on le voit sur la figure 7.18, la fenêtre est séparée en deux:

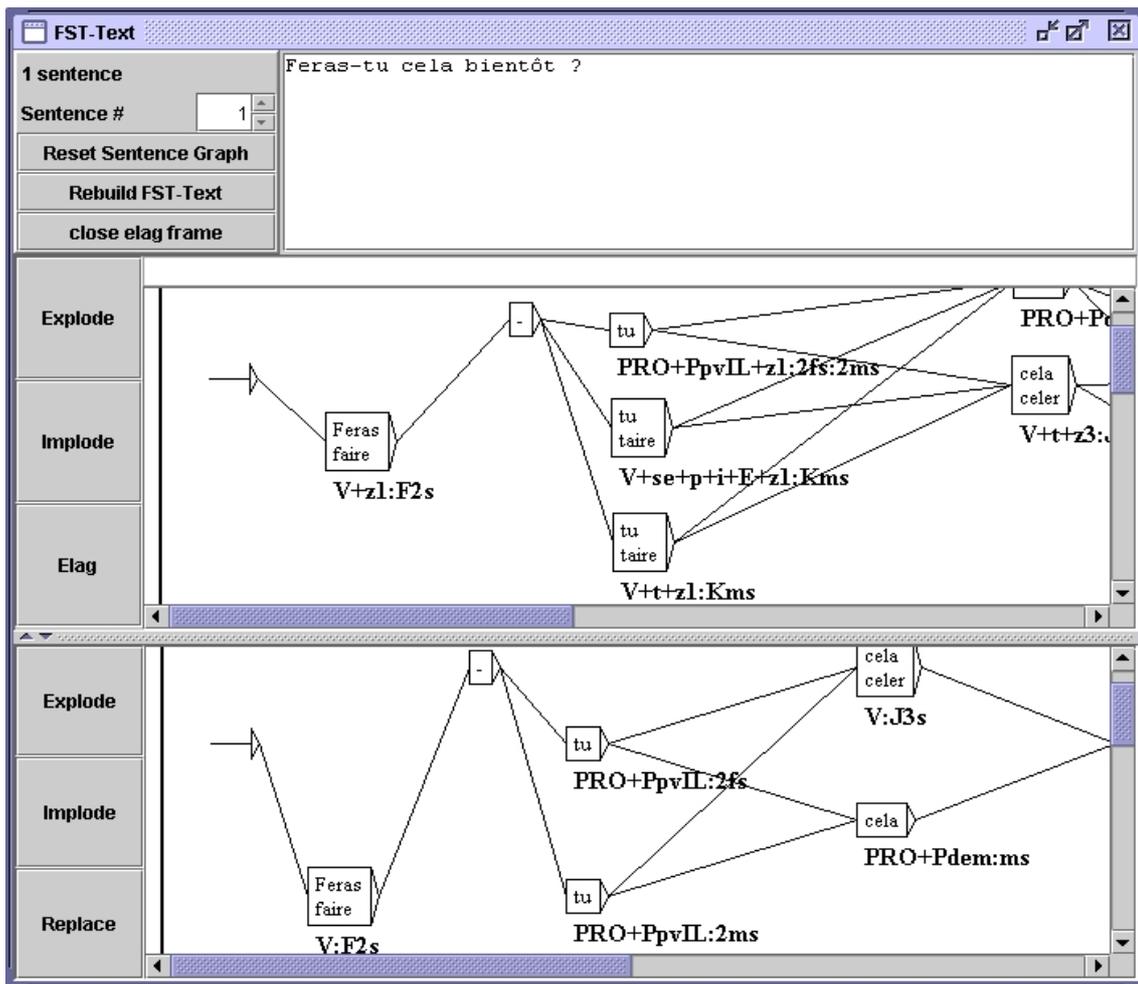


FIG. 7.13 – Résultat de l'application de la grammaire de la figure 7.12

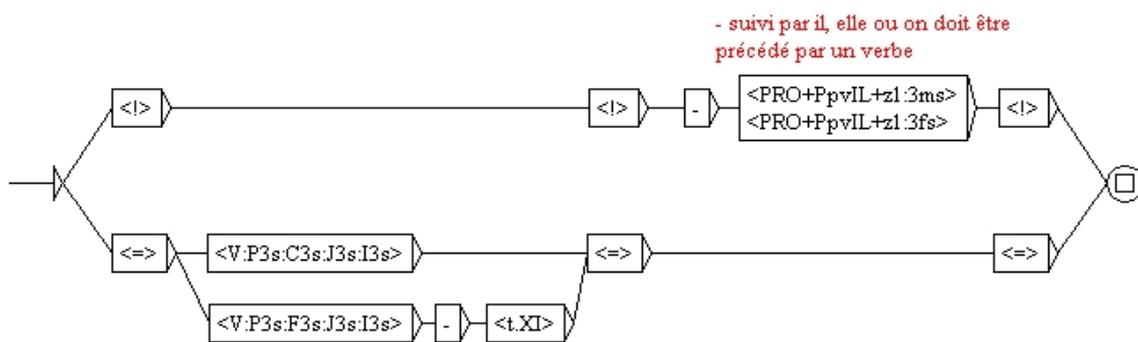


FIG. 7.14 – Utilisation du point de synchronisation

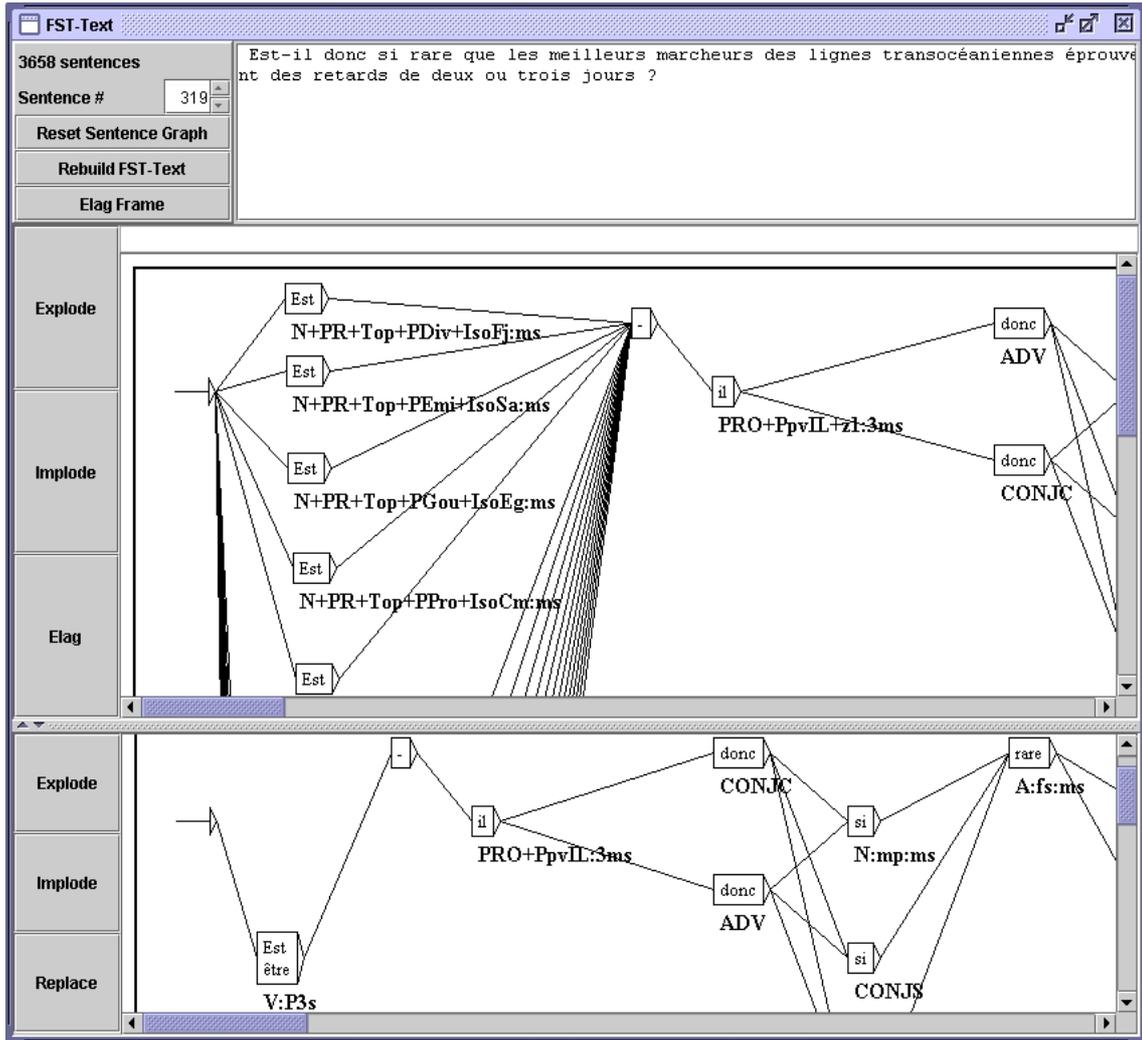


FIG. 7.15 – Résultat de l'application de la grammaire de la figure 7.14

l'automate d'origine est affiché en haut, et l'automate résultat en bas.

Ne soyez pas étonné si l'automate du bas semble plus compliqué. Cela s'explique par le fait que les entrées lexicales factorisées¹ ont été *explosées* de façon à traiter séparément chaque interprétation flexionnelle. Pour refactoriser ces entrées, cliquez sur le bouton *implode*. Un clic sur le bouton *explode* vous donne une vue explosée de l'automate du texte.

Si vous cliquez sur le bouton *replace*, l'automate résultat deviendra le nouvel automate du texte. Ainsi, si vous utilisez d'autres grammaires, elles s'appliqueront sur l'automate déjà partiellement désambiguïsé, ce qui permet de cumuler les effets de plusieurs grammaires.

1. Ce sont des entrées qui regroupent plusieurs interprétations flexionnelles différentes, comme par exemple: {se, .PR0+PpvLE:3ms:3fs:3mp:3fp}.

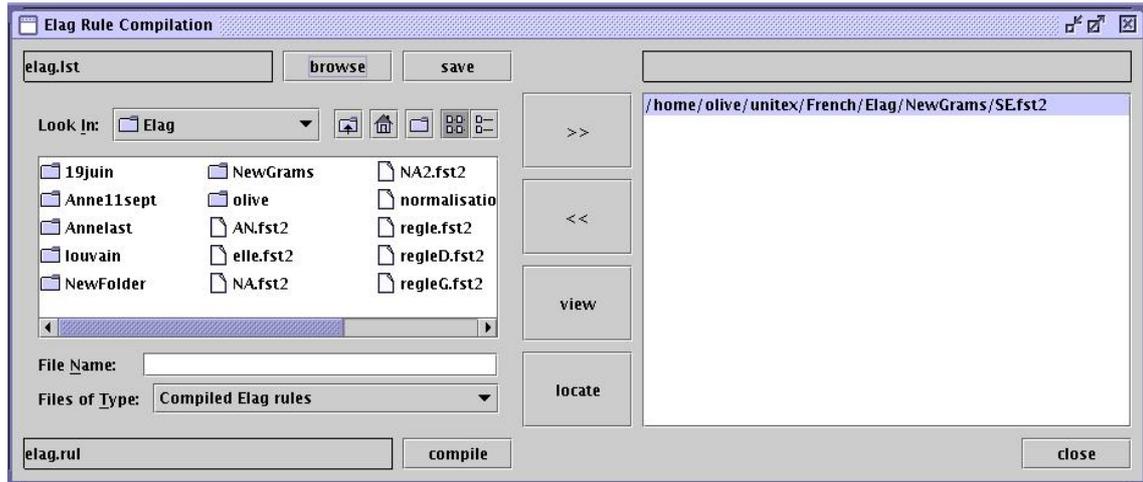


FIG. 7.16 – Fenêtre de compilation des grammaires ELAG

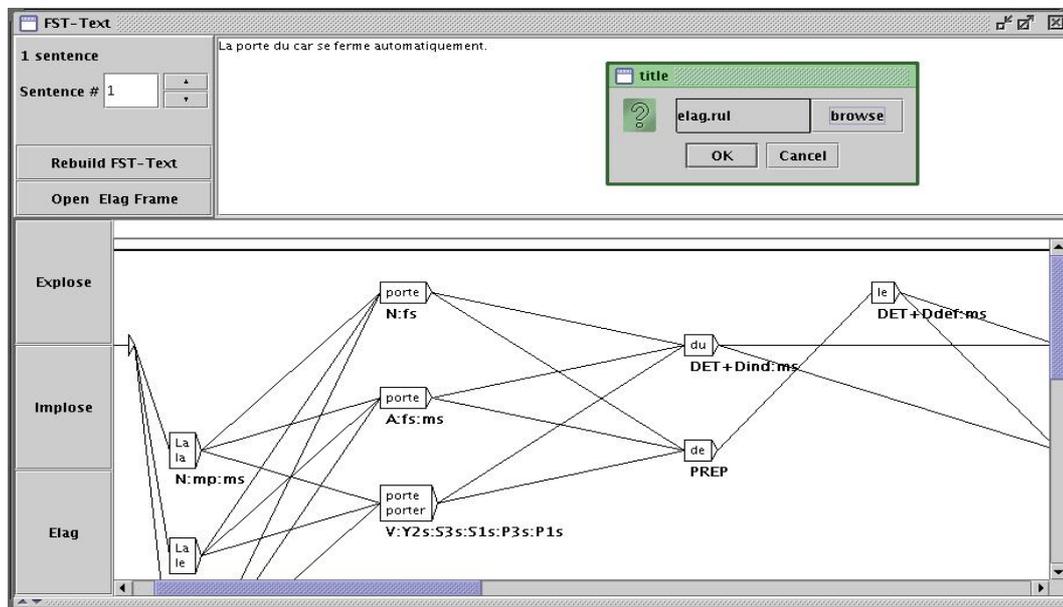


FIG. 7.17 – Fenêtre de l'automate du texte

7.3.4 Ensembles de grammaires

Il est possible de regrouper plusieurs grammaires ELAG en un ensemble de grammaires, afin de les appliquer en une seule fois. Les ensembles de grammaires ELAG sont décrits dans des fichiers `.lst`. Ils sont gérés depuis la fenêtre de compilation des grammaires ELAG (figure 7.16). Le label en haut à gauche indique le nom de l'ensemble courant, par défaut `elag.lst`. C'est le contenu de cet ensemble qui est affiché dans le cadre droit de la fenêtre.

Pour modifier le nom de l'ensemble, cliquez sur le bouton *browse*. Dans la boîte de dialogue

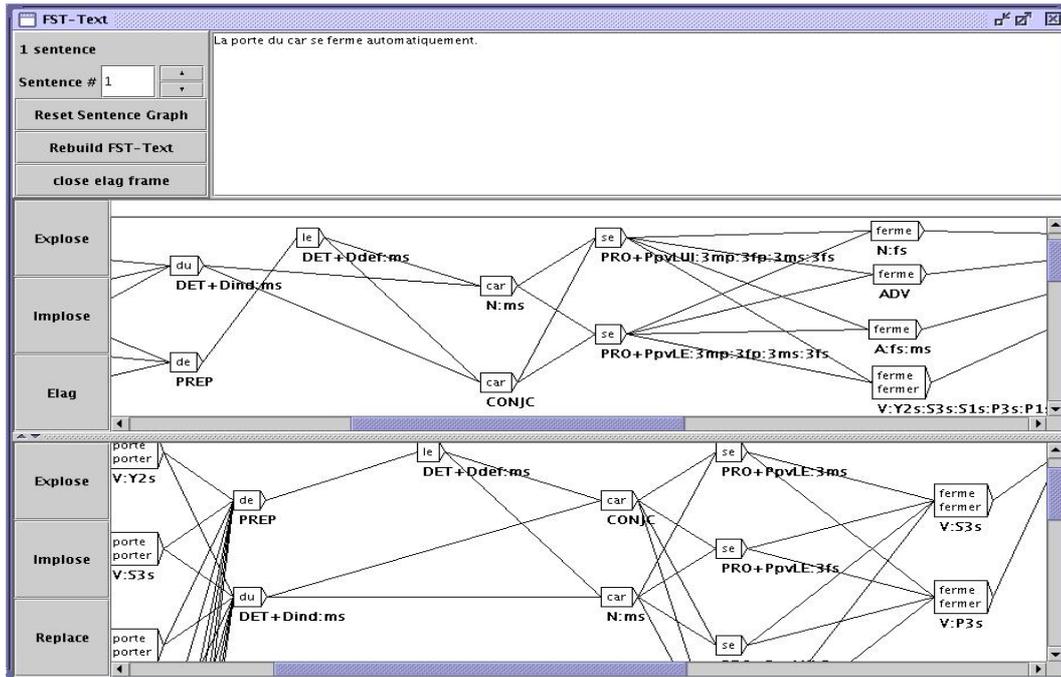


FIG. 7.18 – Fenêtre de l'automate du texte séparée en deux

qui apparaît alors, choisissez le nom du fichier `.lst` que vous voulez donner à votre ensemble.

Pour ajouter une grammaire à l'ensemble, sélectionnez-la dans l'explorateur de fichiers du cadre gauche, et cliquez sur le bouton `>>`.

Pour retirer une grammaire de l'ensemble, sélectionnez-la dans le cadre droit, et cliquez sur le bouton `<<`.

Une fois que vous avez sélectionné toutes vos grammaires, compilez-les en cliquant sur le bouton `compile`. Cela créera un fichier `.rul`, portant le nom indiqué en bas à droite (le nom du fichier est obtenu en remplaçant l'extension `.lst` par l'extension `.rul`).

Vous pouvez maintenant appliquer votre ensemble de grammaires. Comme expliqué plus haut, cliquez sur le bouton `elag` dans la fenêtre de l'automate du texte. Quand la boîte de dialogue vous demande le nom du fichier `.rul` à utiliser, cliquez sur le bouton `browse` et sélectionnez votre ensemble. L'automate résultat est identique à celui qui aurait été obtenu en appliquant successivement chacune des grammaires.

7.3.5 Fenêtre de processing d'ELAG

Lors de la désambiguïsation, le programme `Elag` est lancé dans une fenêtre de processing qui permet de voir les messages émis par le programme pendant son exécution.

Par exemple, lorsque l'automate du texte contient des symboles qui ne correspondent pas au jeu d'étiquettes d'ELAG (voir section suivante), un message indique la nature de l'erreur rencontrée. De même, lorsqu'une phrase est rejetée (toutes les analyses possibles ont été éliminées par les grammaires), un message indique le numéro de la phrase. Cela permet de

localiser rapidement la source des problèmes.

Evaluation de la levée d'ambiguïtés

L'évaluation du taux d'ambiguïté ne se base pas uniquement sur le nombre moyen d'interprétations par mot. Afin d'avoir une mesure plus représentative, le système prend également en compte les différentes combinaisons de mots.

Durant la levée d'ambiguïtés, le programme `Elag` calcule le nombre d'analyses possibles dans l'automate du texte avant et après modification (cela correspond au nombre de chemins possibles dans l'automate). En se basant sur cette valeur, le programme calcule l'ambiguïté moyenne par phrase et par mot. C'est cette dernière mesure qui est utilisée pour représenter le taux d'ambiguïtés du texte, car elle ne varie pas avec la taille du corpus, ni avec le nombre de phrases de celui-ci. La formule appliquée est :

$$\text{taux d'ambiguïtés} = \exp \frac{\log(\text{nombre de chemins})}{\text{longueur du texte}}$$

Le rapport entre le taux d'ambiguïtés avant et après l'application des grammaires donne une mesure de leur efficacité.

Toutes ces informations sont affichées dans le fenêtre de processing d'ELAG.

7.3.6 Description du jeu d'étiquettes

Les programmes `Elag` et `ElagComp` nécessitent une description formelle du jeu d'étiquettes des dictionnaires utilisés. Cette description consiste grosso modo en une énumération de toutes les catégories grammaticales présentes dans les dictionnaires, avec pour chacune d'elle, la liste des codes syntaxiques et flexionnels qui leur sont associées et une description de leurs possibles combinaisons. Ces informations sont décrites dans le fichier nommé `tagset.def`.

Fichier `tagset.def`

Voici un extrait du fichier `tagset.def` utilisé pour le français.

```
NAME français
```

```
POS ADV
```

```
.
```

```
POS PRO
```

```
inflex:
```

```
  pers   = 1 2 3
```

```
  genre  = m f
```

```
  nombre = s p
```

```
discr:
```

```
  subcat = Pind Pdem PpvIL PpvLUI PpvLE Ton PpvPR PronQ Dnom Pposs1s...
```

complete:

```
Pind    <genre> <nombre>
Pdem    <genre> <nombre>
Pposs1s <genre> <nombre>
Pposs1p <genre> <nombre>
Pposs2s <genre> <nombre>
Pposs2p <genre> <nombre>
Pposs3s <genre> <nombre>
Pposs3p <genre> <nombre>
PpvIL   <genre> <nombre> <pers>
PpvLE   <genre> <nombre> <pers>
PpvLUI  <genre> <nombre> <pers>    #
Ton     <genre> <nombre> <pers>    # lui, elle, moi
PpvPR   <genre> <nombre> <pers>    # en y
PronQ   <genre> <nombre> <pers>    # où qui que quoi
Dnom    <genre> <nombre> <pers>    # rien
```

.

POS A ## adjectifs

inflex:

genre = m f

nombre = s p

cat:

gauche = g

droite = d

complete:

<genre> <nombre>

_ # pour {de bonne humeur,.A}, {au bord des larmes,.A} par exemple

.

POS V

inflex:

temps = C F I J K P S T W Y G X

pers = 1 2 3

genre = m f

nombre = s p

complete:

W

G

C <pers> <nombre>

F <pers> <nombre>

I <pers> <nombre>

J <pers> <nombre>

P <pers> <nombre>

```

S <pers> <nombre>
T <pers> <nombre>
X 1 s # eussé dussé puissé fussé (-je)
Y 1 p
Y 2 <nombre>
K <genre> <nombre>
.

```

Le symbole # indique que le reste de la ligne est en commentaire. Un commentaire peut apparaître à n'importe quel endroit dans le fichier. Le fichier commence toujours par le mot **NAME**, suivi par un identifiant (**français**, dans l'exemple). La suite du fichier est constituée de sections **POS** (pour **Part-Of-Speech**), une pour chaque catégorie grammaticale. Chaque section décrit la structure des étiquettes des entrées lexicales appartenant à la catégorie grammaticale concernée. Chaque section se compose de 4 parties qui sont toutes optionnelles :

- **inflex**: cette partie énumère les codes flexionnels relatifs à la catégorie grammaticale. Par exemple, les codes **1,2,3** qui dénotent la personne de l'entrée sont des codes pertinents aux pronoms mais non aux adjectifs. Chaque ligne décrit un attribut flexionnel (genre, temps, etc), et est composée du nom de l'attribut, suivi du signe = et des valeurs qu'il peut prendre; Par exemple, la ligne suivante déclare un attribut *pers* pouvant prendre les valeurs 1, 2 ou 3 :

```
pers = 1 2 3
```

- **cat**:

cette partie déclare les attributs syntaxiques et sémantiques qui peuvent être attribués aux entrées appartenant à la catégorie grammaticale concernée. Chaque ligne décrit un attribut et les valeurs qu'il peut prendre. Les codes déclarés pour un même attribut doivent être exclusifs les uns des autres. Autrement dit, une entrée ne peut pas prendre plus d'une valeur pour un même attribut. En revanche, il peut exister des étiquettes ne prenant aucune valeur pour un attribut donné. Par exemple, pour définir l'attribut *niveau_de_langue* pouvant prendre les valeurs **z1**, **z2** et **z3**, on écrira la ligne suivante :

```
niveau_de_langue = z1 z2 z3
```

- **discr**:

cette partie est constituée de la déclaration d'un unique attribut. La syntaxe est la même que dans la partie **cat** et l'attribut décrit ici ne doit pas y être répété. Cette partie permet de diviser la catégorie grammaticale en sous-catégories *discriminantes* dans lesquelles les entrées ont des attributs flexionnels similaires. Pour les pronoms par exemple, une indication de personne est attribuée aux entrées appartenant à la sous-catégorie des pronoms personnels mais non aux pronoms relatifs. Ces dépendances sont décrites dans la partie **complete**;

- **complete**:

Dans cette partie est explicité l'étiquetage morphologique des mots appartenant à la catégorie grammaticale courante. Chaque ligne décrit une combinaison valide de codes

flexionnels en fonction de leur sous catégorie discriminante (si une telle catégorie a été déclarée). Lorsqu'un nom d'attribut apparaît entre angles (< et >), cela signifie que n'importe quelle valeur de cet attribut peut convenir. Il est également possible de déclarer qu'une entrée ne prend aucun trait flexionnel au moyen d'une ligne ne contenant que le caractère _ (underscore). Ainsi par exemple, si nous considérons les lignes suivantes extraites de la section concernant la description des verbes :

```
W
K <genre> <nombre>
```

Elles permettent de déclarer que les verbes à l'infinitif (dénnoté par le code W) n'ont pas d'autres traits flexionnels positionnés tandis que les formes à participe passé (code K) sont également attribuées d'un genre et d'un nombre.

Description des codes flexionnels

La principale fonction de la partie `discr` est de diviser les étiquettes en sous-catégories ayant un comportement morphologique similaire. Ces sous-catégories sont ensuite utilisées pour faciliter l'écriture de la partie `complete`. Pour la lisibilité des grammaires ELAG, il est souhaitable que les éléments d'une même sous-catégorie aient tous le même comportement flexionnel; dans ce cas la partie `complete` est composée d'une seule ligne par sous-catégorie.

Considérons par exemple les lignes suivantes, extraites de la description des pronoms:

```
Pdem <genre> <nombre>
PpvIl <genre> <nombre> <pers>
PpvPr
```

Ces lignes signifient:

- tous les pronoms démonstratifs (<PRO+Pdem>) ont des indications de genre et de nombre, et aucune autre;
- les pronoms personnels nominatifs (<PRO+PpvIl>) sont étiquetés morphologiquement par une personne, un genre et nombre;
- les pronoms prépositionnels (*en, y*) n'ont aucun trait flexionnel.

Toutes les combinaisons des traits flexionnels et discriminants qui apparaissent dans les dictionnaires doivent être décrits dans le fichier `tagset.def`, faute de quoi les entrées correspondantes seront rejetées par ELAG.

Dans le cas où des mots d'une même sous-catégorie diffèrent par leurs traits flexionnels, il est nécessaire d'écrire plusieurs lignes dans la partie `complete`. L'inconvénient de cette méthode de description, est qu'il devient difficile de faire la distinction entre de tels mots dans une grammaire ELAG.

Si l'on considère la description donnée précédemment en exemple, certains adjectifs du français prennent un genre et un nombre, alors que d'autres n'ont aucun trait flexionnel. C'est par exemple le cas de séquences figées comme *de bonne humeur*, qui ont un comportement syntaxique très proche de celui des adjectifs. De telles séquences ont ainsi été intégrées dans le dictionnaire du français en tant qu'adjectifs invariables et donc sans trait flexionnel. Le problème est que si l'on veut faire référence exclusivement à ce type d'adjectifs dans une

grammaire de désambiguïsation, le symbole `<A>` ne convient pas, puisqu'il donnera tous les adjectifs.

Pour contourner cette difficulté, il est possible de nier un attribut flexionnel, en écrivant le caractère `@` juste avant une des valeurs possibles pour cet attribut. Ainsi, le symbole `<A:@m@p>` reconnaît tous les adjectifs qui n'ont ni genre ni nombre. A l'aide de cet opérateur, il est maintenant possible d'écrire des grammaires comme celles de la figure 7.19, qui imposent l'accord en genre et en nombre entre un nom et l'adjectif qui le suit ². Cette grammaire conservera l'analyse correcte de phrases comme: *Les personnes de bonne humeur m'insupportent*.

Il est toutefois recommandé de limiter l'usage de l'opérateur `@`, car cela nuit à la lisibilité des grammaires. Il est préférable de distinguer les étiquettes qui acceptent différentes combinaisons flexionnelles au moyen de sous-catégories discriminantes définies dans la partie `discr`.

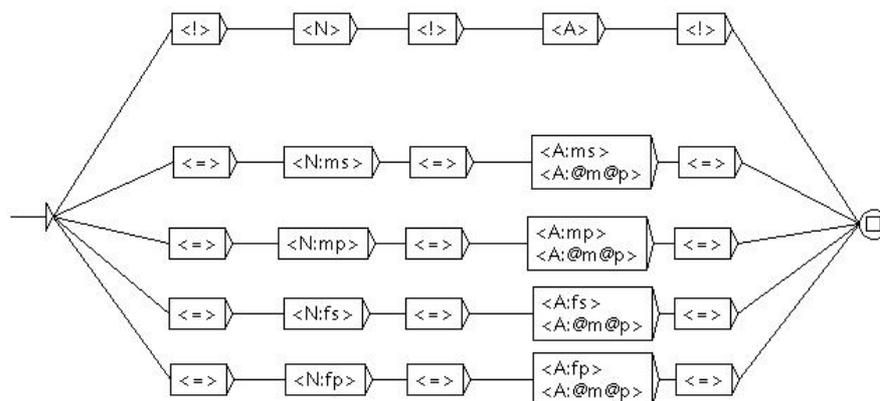


FIG. 7.19 – Grammaire ELAG vérifiant l'accord en genre et en nombre entre un nom et l'adjectif qui le suit

Codes optionnels

Les codes syntaxiques et sémantiques optionnels sont déclarés dans la partie `cat`. Ils peuvent être utilisés dans les grammaires ELAG comme les autres codes. La différence est que ces codes n'interviennent pas pour décider si une étiquette doit être rejetée ou non. Ce sont des codes facultatifs, qui sont indépendants des autres codes, comme par exemple l'attribut de niveau de langue (`z1`, `z2` ou `z3`). De la même manière que pour les codes flexionnels, il est également possible de nier un attribut flexionnel en écrivant le caractère `!` juste avant le nom de l'attribut. Ainsi, avec notre fichier d'exemple, le symbole `<A!gauche:f>` reconnaît tous les adjectifs au féminin qui ne possèdent pas le code `g` ³.

Tous les codes qui ne sont pas déclarés dans le fichier `tagset.def` sont ignorés par ELAG.

2. Cette grammaire n'est pas complètement correcte, car elle élimine par exemple l'analyse correcte de la phrase: *J'ai reçu des coups de fil de ma mère hallucinants*.

3. Ce code indique que l'adjectif doit apparaître à gauche du nom auquel il se rapporte, comme c'est le cas pour *bel*.

Si une entrée de dictionnaire contient un tel code, ELAG produira un avertissement et retirera le code de l'entrée. En conséquence, si deux entrées concurrentes ne différaient dans l'automate du texte d'origine que par des codes non déclarés, ces entrées deviendront indistinguables par le programmes et seront donc unifiées en une seule entrée dans l'automate résultat. Ainsi, le jeu d'étiquettes décrit dans le fichier `tagset.def` peut suffire à réduire l'ambiguïté, en factorisant des mots qui ne diffèrent que par des codes non déclarés et ceci indépendamment des grammaires appliquées.

Par exemple, dans la version la plus complète du dictionnaire du français, chaque emploi distinct d'un verbe est caractérisé par une référence vers la table du lexique-grammaire qui le caractérise. Nous avons considéré jusqu'à présent que ces informations relèvent plus de la syntaxe que de l'analyse lexicale et nous ne les avons donc pas intégré dans la description du jeu d'étiquettes. Celle-ci sont donc automatiquement éliminées lors du chargement de l'automate du texte, ce qui réduit sont taux d'ambiguïtés.

Afin de bien distinguer les effets liés au jeu d'étiquettes de ceux de des grammaires ELAG, il est conseillé de procéder à une étape préalable de normalisation de l'automate du texte avant de lui appliquer les grammaires de désambiguïsation. Cette normalisation s'effectue en appliquant à l'automate du texte une grammaire n'imposant aucune contrainte, comme celle de la figure 7.20. Notez que cette grammaire est normalement présente dans la distribution d'Unitex et pré-compilée dans le fichier `norm.rul`.

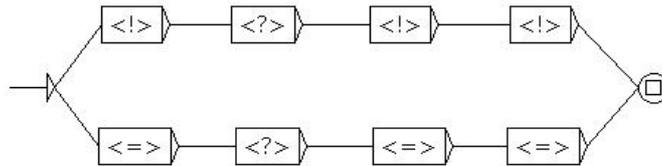


FIG. 7.20 – Grammaire ELAG n'exprimant aucune contrainte

Le résultat de l'application de cette grammaire est que l'automate d'origine est nettoyé de tous les codes qui ne sont soit pas décrits dans le fichier `tagset.def`, soit non conformes à cette description (à cause de catégories grammaticales inconnues ou de combinaisons invalides de traits flexionnels). En remplaçant alors l'automate du texte par l'automate ainsi normalisé, on peut être sûr que les modifications ultérieures de l'automate seront uniquement dues aux effets des grammaires ELAG.

7.3.7 Optimiser les grammaires

La compilation des grammaires effectuée par le programme `ElagComp` consiste à construire un automate dont le langage est l'ensemble des séquences d'entrées lexicales (ou interprétation lexicale d'une phrase) qui ne sont pas rejetées par les grammaires. Cette tâche est complexe et peut prendre beaucoup de temps, il est toutefois possible de l'accélérer sensiblement en observant certains principes lors de l'écriture des grammaires.

Limiter le nombre de branches *alors*

Il est recommandé de réduire au minimum le nombre de parties *alors* d'une grammaire. Cela peut réduire considérablement le temps de compilation des grammaires. Le plus souvent, une grammaire possédant beaucoup de parties *alors* peut être réécrite avec une ou deux parties *alors*, sans perte de lisibilité. C'est par exemple le cas de la grammaire de la figure 7.21, qui impose une contrainte entre un verbe et le pronom qui le suit.

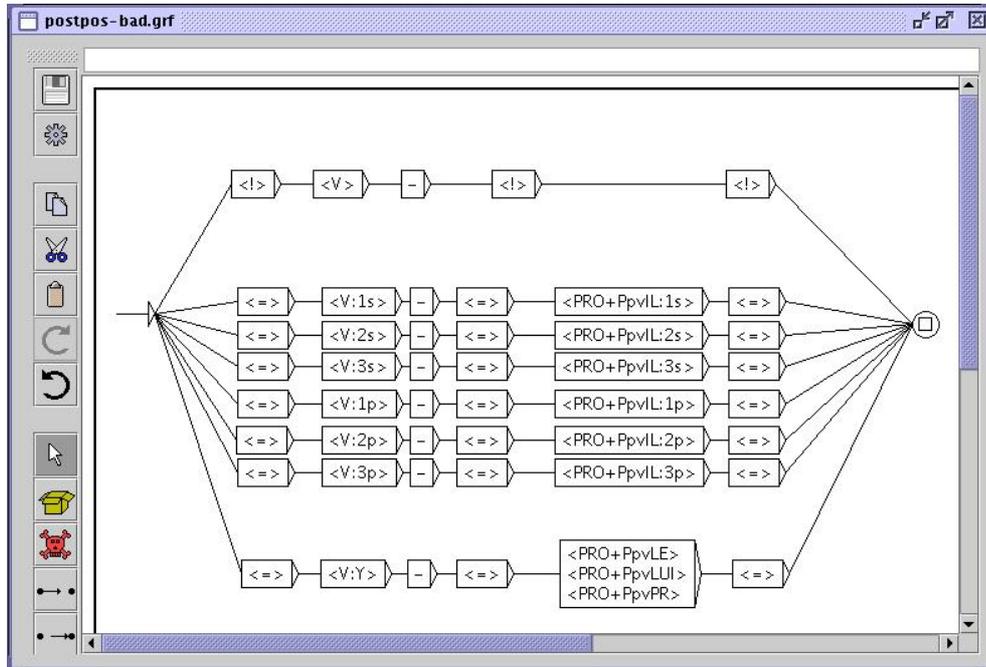


FIG. 7.21 – Grammaire ELAG vérifiant l'accord entre verbe et pronom

Comme on peut le voir sur la figure 7.22, on peut écrire une grammaire équivalente en factorisant toutes les parties *alors* en une seule. Les deux grammaires auront exactement le même effet sur l'automate du texte, mais la seconde sera compilée beaucoup plus rapidement.

Utilisation des symboles lexicaux

Il vaut mieux n'utiliser les lemmes que lorsque c'est absolument nécessaire. Cela est particulièrement vrai pour les mots grammaticaux, lorsque leurs sous-catégories portent presque autant d'information que les lemmes eux-mêmes. Si vous utilisez malgré tout un lemme dans un symbole, il est recommandé de préciser le plus possible ses traits syntaxiques, sémantiques et flexionnels.

Par exemple, avec les dictionnaires fournis pour le français, il est préférable de remplacer des symboles comme <je.PRO:1s>, <je.PRO+PpvIL:1s> et <je.PRO> par le symbole <PRO+PpvIL:1s>. En effet, tous ces symboles sont identiques dans la mesure où ils ne peuvent reconnaître que l'unique entrée de dictionnaire {je, .PRO+PpvIL:1ms:1fs}. Cependant, comme le programme ne peut pas déduire automatiquement cette information, si l'on ne précise

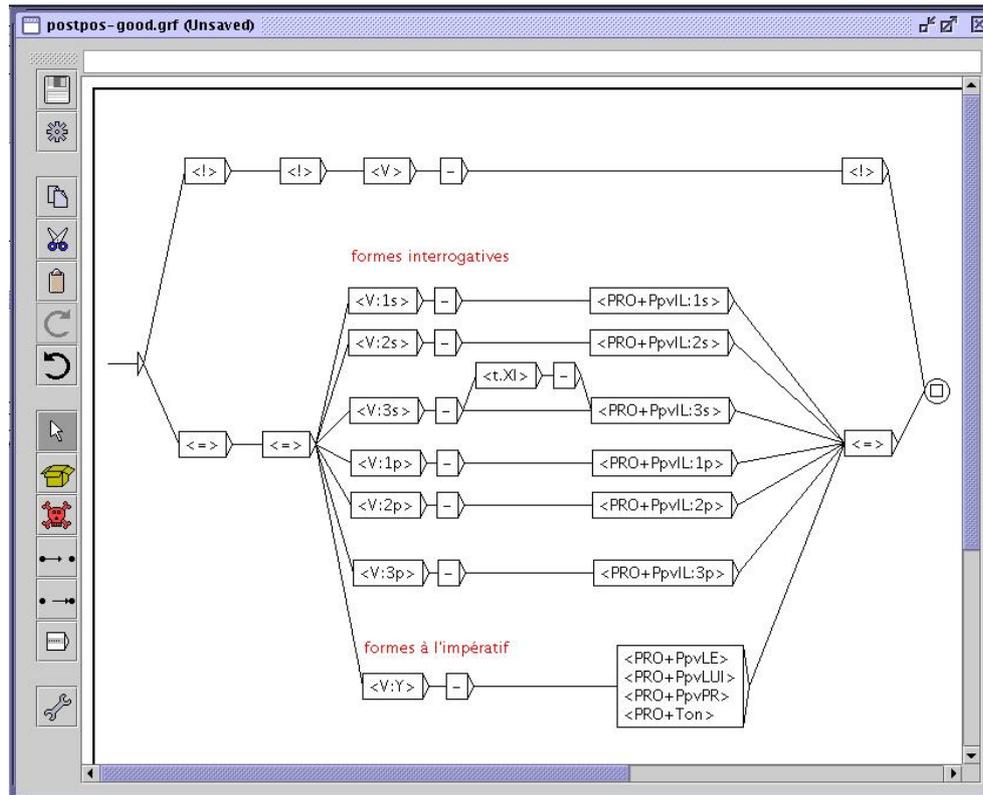


FIG. 7.22 – Grammaire ELAG optimisée vérifiant l'accord entre verbe et pronom

pas tous ces traits, le programme considerera en vain des étiquettes non existantes telles <je.PRO:3p>, <je.PRO+PronQ>, etc.

7.4 Manipulation de l'automate du texte

7.4.1 Affichage des automates de phrases

Comme nous l'avons vu précédemment, l'automate d'un texte est en réalité l'ensemble des automates des phrases de ce texte. Cette structure peut être représentée grâce au format `.fst2`, utilisé pour représenter les grammaires compilées.

Cependant, ce format ne permet pas d'afficher directement les automates de phrases. Il faut donc utiliser un programme (`Fst2Grf`) pour convertir un automate de phrase en un graphe pour qu'il puisse être affiché. Ce programme est appelé automatiquement quand vous sélectionnez une phrase pour générer le fichier `.grf` correspondant.

Les fichiers `.grf` générés ne sont pas interprétés de la même manière que les fichiers `.grf` qui représentent des graphes construits par l'utilisateur. En effet, dans un graphe normal, les lignes d'une boîte sont séparées par le symbole `+`. Dans un graphe de phrase, chaque boîte est soit une unité lexicale sans étiquette, soit une entrée de dictionnaire encadrée par des

accolades. Si la boîte ne contient qu'une unité sans étiquette, celle-ci apparaît seule dans la boîte. Si la boîte contient une entrée de dictionnaire, la forme fléchée est affichée, suivie de sa forme canonique si celle-ci est différente. Les informations grammaticales et flexionnelles sont affichées sous la boîte, comme dans les transductions.

La figure 7.23 montre le graphe obtenu pour la première phrase d'*Ivanhoe*. Les mots *Ivanhoe*, *Walter* et *Scott* sont considérés comme des mots inconnus. Le mot *by* correspond à deux entrées dans le dictionnaire. Le mot *Sir* correspond également à deux entrées du dictionnaire, mais comme la forme canonique de ces entrées est *sir*, elle est affichée puisqu'elle diffère de la forme fléchée par une minuscule.

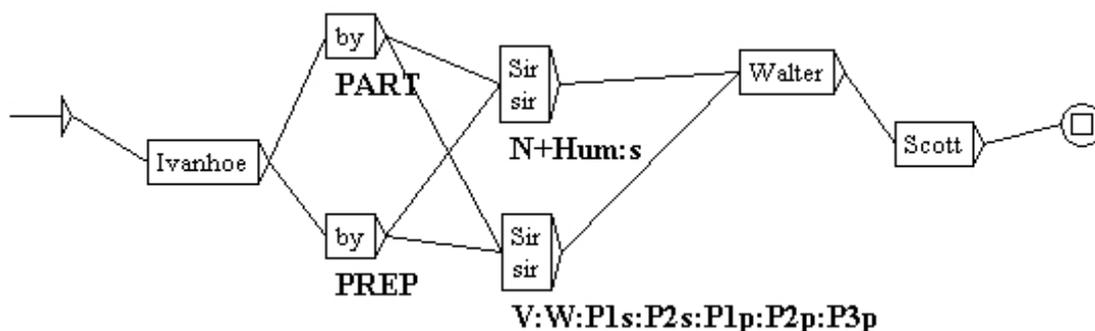


FIG. 7.23 – Automate de la première phrase d'*Ivanhoe*

7.4.2 Modifier manuellement l'automate du texte

Il est possible de modifier manuellement les automates de phrase, sauf ceux qui apparaissent dans le cadre réservé à ELAG (cadre du bas). Vous pouvez ajouter ou supprimer des boîtes ou des transitions. Lorsqu'un graphe est modifié, il est sauvegardé dans le répertoire du texte sous le nom `sentenceN.grf`, où *N* représente le numéro de la phrase.

Lorsque vous sélectionnez une phrase, si un graphe modifié existe pour cette phrase, celui-ci est affiché. Vous pouvez alors réinitialiser l'automate de cette phrase en cliquant sur le bouton "Reset Sentence Graph" (voir figure 7.24).

Lors de la construction de l'automate d'un texte, tous les graphes de phrase modifiés présents dans le répertoire du texte sont effacés.

NOTE: vous pouvez reconstruire l'automate du texte en prenant en compte vos modifications manuelles. Pour cela, cliquez sur le bouton "Rebuild FST-Text". Toutes les phrases pour lesquelles des modifications ont été faites sont alors remplacées dans l'automate du texte par leur version modifiée. Le nouvel automate du texte est ensuite rechargé automatiquement.

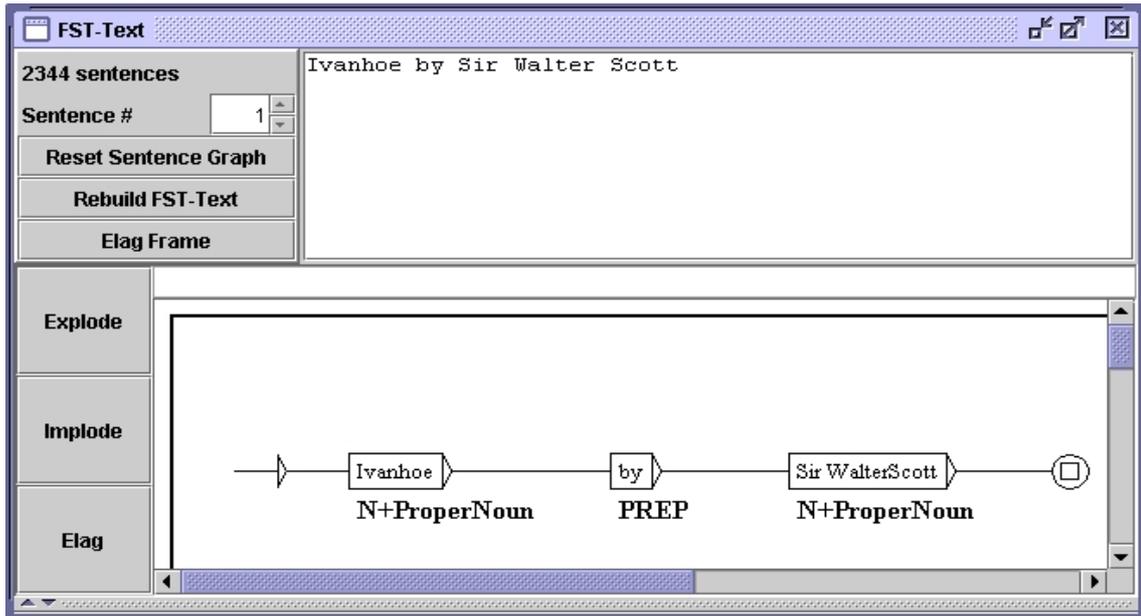


FIG. 7.24 – Automate de phrase modifié

7.4.3 Paramètres de présentation

Les automates de phrase sont soumis aux mêmes options de présentation que les graphes. Ils partagent les mêmes couleurs et polices de caractères, ainsi que l'utilisation de l'effet d'antialiasing. Pour configurer l'apparence des automates de phrase, vous devez modifier la configuration générale en cliquant sur "Preferences..." dans le menu "Info". Pour plus de détails, reportez-vous à la section 5.3.5.

Vous pouvez également imprimer un automate de phrase en cliquant sur "Print..." dans le menu "FSGraph" ou en appuyant sur <Ctrl+P>. Assurez-vous que le paramètre d'orientation de l'imprimante est bien réglé sur le mode paysage. Pour régler ce paramètre, cliquez sur "Page Setup" dans le menu "FSGraph".

Chapitre 8

Lexique-grammaire

Les tables de lexique-grammaire sont un moyen compact de représenter les propriétés syntaxiques des éléments d'une langue. Il est possible de construire automatiquement des grammaires locales à partir de ces tables, grâce à un mécanisme de graphes paramétrés.

La première partie de ce chapitre présente le formalisme de ces tables. La seconde partie décrit les graphes paramétrés et le mécanisme de génération automatique de graphes à partir d'une table de lexique-grammaire.

8.1 Les tables de lexique-grammaire

Le lexique-grammaire est une méthodologie qui a été développée par Maurice Gross et son équipe du LADL ([5], [6], [24], [26]) sur le principe suivant: chaque verbe a des propriétés syntaxiques quasiment uniques. De ce fait, ces propriétés doivent être systématiquement décrites, car il est impossible de prévoir le comportement précis d'un verbe. Ces descriptions systématiques sont représentées au moyen de matrices où les lignes correspondent aux verbes, et les colonnes aux propriétés syntaxiques. Les propriétés considérées sont des propriétés formelles telles que le nombre et la nature des compléments admis par le verbe et les différentes transformations que ce verbe peut subir (passivation, nominalisation, extraposition, etc.). Les matrices, plus souvent appelées tables, sont binaires: un signe + apparaît à l'intersection d'une ligne et d'une colonne d'une propriété si le verbe vérifie la propriété, un signe - sinon.

Ce type de description a également été appliqué aux adjectifs ([32], [36]), aux noms prédictifs ([19], [20]), aux adverbes ([25], [38]) ainsi qu'aux expressions figées, et ce dans plusieurs langues ([9], [15], [16], [40], [41], [43], [46], [48], [50]).

La figure 8.1 montre un exemple de table de lexique-grammaire. Cette table concerne les verbes admettant un complément numérique.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	N0 = Npc				Aux	32NM	N1 V	N1 = Nhum	N1 = N-hum	N1 = le fait Qu P	N1 = V-n	N1 = Dnum Nmes	Ppv = le	N1 V NO	N0 V Adj	N0 V Dnum V-n	N0 V à N1	N1 V NO (<E>+t) N1	N1 = V-n	Exemple
2	-	+	-	-	avoir	accepter	-	-	+	-	-	-	+	-	-	-	-	-	-	Ce salon§accepte§vingt personnes
3	-	+	-	-	avoir	accueillir	-	-	+	-	-	-	+	-	-	-	-	-	-	Ce salon§accueille§vingt personnes
4	-	+	-	-	avoir	accuser	-	-	+	-	-	+	+	-	-	-	-	-	-	Max§accuse§80 kilos
5	-	+	-	-	avoir	accuser	-	-	+	+	-	-	-	-	-	-	-	-	-	Max§accuse§ses trente ans
6	-	+	-	-	avoir	admettre	-	-	+	-	-	-	+	-	-	-	-	-	-	On§admet§50 personnes dans cette salle
7	-	+	-	-	avoir	affecter	-	-	+	-	-	-	-	-	-	-	-	-	-	Ces cristaux§affectent§une forme géométrique
8	-	+	-	-	avoir	afficher	-	-	+	-	-	+	-	-	-	-	-	-	-	Les valeurs ont§affiché§un repli
9	-	+	-	-	avoir	aimer	-	-	+	+	-	+	-	-	-	-	-	-	-	La plante§aime§l'eau
10	-	+	-	-	avoir	approcher	+	-	+	-	-	+	-	-	-	-	-	-	-	Cette maison§approche§les deux millions
11	-	+	-	-	avoir	arpenter	-	-	-	-	-	+	+	-	-	+	+	-	-	Ce terrain§arpen§30 arpents
12	-	+	-	-	avoir	atteindre	-	-	+	-	-	+	+	-	-	-	-	-	-	Max§atteint§80 kilos
13	+	+	+	-	avoir	avoir	-	-	+	-	-	+	+	-	-	-	-	-	-	Max§a§(une soeur+une voiture+des sous)
14	-	+	-	-	avoir	avoisiner	-	-	+	-	-	+	+	-	-	-	-	-	-	Ce sac§avoisine§les 20 kg.
15	-	+	-	-	avoir	battre	+	-	+	-	-	+	-	-	-	-	-	-	-	La montre§bat§les secondes
16	-	+	-	-	avoir	cache	-	-	+	-	-	-	-	-	-	-	-	-	-	Son calme§cache§(son+une grande)angoisse
17	-	+	-	-	avoir	caler	-	-	+	-	-	+	+	-	+	-	-	-	-	Ce bateau§cale§80 cm

FIG. 8.1 – Table de lexique-grammaire 32NM

8.2 Conversion d'une table en graphes

8.2.1 Principe des graphes paramétrés

La conversion d'une table en graphes s'effectue au moyen du mécanisme des graphes paramétrés. Le principe est le suivant: on construit un graphe qui décrit des constructions possibles. Ce graphe fait référence aux colonnes de la table grâce à des variables. On génère ensuite pour chaque ligne de la table une copie de ce graphe dans laquelle les variables sont remplacées en fonction du contenu des cellules situées à l'intersection des colonnes correspondantes et de la ligne traitée. Si une cellule de la table contient le signe +, la variable correspondante est remplacée par <E>. Si la cellule contient le signe -, la boîte contenant la variable correspondante est supprimée, ce qui détruit du même coup les chemins passant par cette boîte. Dans tous les autres cas, la variable est remplacée par le contenu de la cellule.

8.2.2 Format de la table

Les tables de lexique-grammaire sont généralement codées à l'aide d'un tableur comme OpenOffice.org Calc ([47]). Pour pouvoir être utilisées par Unitex, les tables doivent être codées en texte Unicode selon la convention suivante: les colonnes doivent être séparées par des tabulations et les lignes par des retours à la ligne.

Pour convertir une table avec OpenOffice.org Calc, sauvegardez-la au format texte (extension .csv). Le programme vous propose ensuite de paramétrer la sauvegarde au moyen d'une fenêtre comme celle de la figure 8.2. Choisissez le codage "Unicode", sélectionnez la tabulation

comme séparateur de colonnes, et ne précisez pas de délimiteur de texte.



FIG. 8.2 – Configuration de la sauvegarde d'une table avec OpenOffice.org Calc

Lors de la génération des graphes, Unitex saute la première ligne, considérée comme donnant les en-têtes des colonnes. Vous devez donc vous assurer que les en-têtes des colonnes occupent exactement une ligne. S'il n'y a pas de ligne d'en-tête, la première ligne de la table sera ignorée, et s'il y a plusieurs lignes d'en-tête, elles seront interprétées à partir de la deuxième comme des lignes de la table.

8.2.3 Les graphes paramétrés

Les graphes paramétrés sont des graphes dans lesquels apparaissent des variables faisant référence aux colonnes d'une table de lexique-grammaire. On utilise généralement ce mécanisme avec des graphes syntaxiques, mais rien n'empêcherait de construire des graphes paramétrés de flexion, de prétraitement ou de normalisation.

Les variables qui font référence aux colonnes sont formées du caractère @ (arrobas) suivi d'un nom de colonne en lettres majuscules (les colonnes sont numérotées en partant de A).

Exemple: @C fait référence à la troisième colonne de la table

Lorsqu'une variable doit être remplacée par un + ou un -, le signe - correspond à la suppression du chemin passant par cette variable. Il est possible d'effectuer l'opération contraire en faisant précéder le caractère @ d'un point d'exclamation. Dans ce cas, c'est lorsque la variable renvoie à un signe + que le chemin est supprimé. Si la variable ne renvoie ni à un signe + ni à un signe -, elle est remplacée par le contenu de la cellule.

Il existe également une variable spéciale @% qui est remplacée par le numéro de la ligne dans la table. Le fait que sa valeur soit différente pour chaque ligne permet de l'utiliser pour caractériser facilement une ligne. Cette variable n'est pas affectée par la présence d'un point d'exclamation à sa gauche.

La figure 8.3 montre un exemple de graphe paramétré conçu pour être appliqué à la table de lexique-grammaire 31H présentée sur la figure 8.4.

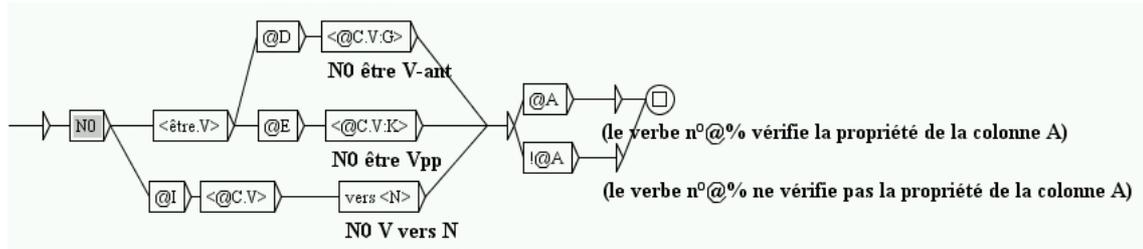


FIG. 8.3 – Exemple de graphe paramétré

Table31H.xls															
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
		31H												Exemple	
1	N0 =: V- _n	Aux		N0 est V-ant	N0 est Vpp	N0pc lui V	N0 V de N0pc	Nhum V sur ce point	N0 V vers N	il V N0 W	idée Loc esprit	Nhum Loc Nabs	N0 =: N-hum		
2	-	avoir	abandonner	-	-	-	-	-	-	-	-	-	-	Paul a§abandonné§	
3	-	avoir	abuser	-	-	-	-	+	-	-	-	-	-	Max§abuse§	
4	-	avoir	acquiescer	-	-	-	+	+	-	-	-	-	-	Max a§acquiescé§(E+de la tête)	
5	-	avoir	adouber	-	-	-	-	-	-	-	-	-	-	Paul§adoube§ échecs	
6	-	avoir	agioter	-	-	-	-	-	-	+	-	-	-	Max§agioté§sur les changes	
7	-	avoir	agoniser	+	-	-	-	-	+	-	-	+	-	Max§agonise§	
8	-	avoir	archaïser	+	-	-	-	+	-	-	-	-	-	Cet auteur§archaïse§volontiers	
9	-	avoir	arquer	-	-	-	+	+	-	-	-	-	-	Max a§arqué§toute la journée	
10	-	être	arriver	-	+	-	-	-	+	-	+	-	-	Max est§arrivé§	
11	-	avoir	atermoyer	-	-	-	-	+	-	+	+	-	-	Max§atermoie§	
12	+	avoir	badauder	-	-	-	-	-	+	-	+	-	-	Max§badaude§	

FIG. 8.4 – Table de lexique-grammaire 31H

8.2.4 Génération automatique de graphes

Pour pouvoir générer des graphes à partir d'un graphe paramétré et d'une table, il faut tout d'abord ouvrir la table en cliquant sur "Open..." dans le menu "Lexicon-Grammar" (voir figure 8.5). La table doit avoir été préalablement convertie en texte Unicode.

La table sélectionnée est alors affichée dans une fenêtre (voir figure 8.6).

Pour générer automatiquement des graphes à partir d'un graphe paramétré, cliquez sur "Compile to GRF..." dans le menu "Lexicon-Grammar". La fenêtre de la figure 8.7 apparaît alors.

Dans le cadre "Reference Graph (in GRF format)", indiquez le nom du graphe paramétré à utiliser. Dans le cadre "Resulting GRF grammar", indiquez le nom du graphe principal qui sera généré. Ce graphe principal est un graphe faisant appel à tous les graphes qui auront



FIG. 8.5 – Menu "Lexicon-Grammar"

NO = V-n	Aux	31H	NO est V-ant	NO est Vpp	NOpc lui V	NO V de NO...	Nhum V sur...	NO V vers N	il V NO W
-	avoir	abando...	-	-	-	-	-	-	-
-	avoir	abuser	-	-	-	-	+	-	-
-	avoir	acquie...	-	-	-	+	+	-	-
-	avoir	adouber	-	-	-	-	-	-	-
-	avoir	agioter	-	-	-	-	-	-	+
-	avoir	agoniser	+	-	-	-	-	-	+
-	avoir	archaiser	+	-	-	-	+	-	-
-	avoir	arquer	-	-	-	+	-	+	-
-	être	arriver	-	+	-	-	-	-	+
-	avoir	atermoyer	-	-	-	-	+	-	+
+	avoir	badauder	-	-	-	-	-	+	-
-	avoir	baissier	-	-	-	-	+	-	-
-	avoir	bambocher	-	-	-	-	-	-	+
-	avoir	bander	-	-	-	+	-	-	+
-	avoir	barouder	-	-	-	-	-	-	+
-	avoir	batifoler	+	-	-	-	-	-	+
-	avoir	bécher	-	-	-	-	+	-	-
+	avoir	bétifier	+	-	-	-	+	-	+
+	avoir	bigler	-	-	+	+	-	+	+
-	avoir	boiter	-	-	-	+	-	-	+
-	avoir	boitiller	+	-	-	+	-	+	+
+	avoir	bouffo...	-	-	-	-	-	-	+

FIG. 8.6 – Affichage d'une table

été générés. En lançant une recherche dans un texte avec ce graphe, vous appliquerez ainsi simultanément tous les graphes générés.

Le cadre "Name of produced subgraphs" permet de préciser le nom des graphes qui seront générés. Afin d'être certain que tous les graphes auront des noms distincts, il est conseillé d'utiliser la variable @%; cette variable sera remplacée pour chaque entrée par le numéro de celle-ci, garantissant ainsi que tous les graphes auront un nom différent. Par exemple, si l'on remplit ce cadre avec le nom `TestGraph_@%.grf`, le graphe généré à partir de la 16^{ème} ligne sera nommé `TestGraph_0016.grf`.

Les figures 8.8 et 8.9 montrent deux graphes générés en appliquant le graphe paramétré de la figure 8.3 à la table 31H. La figure 8.10 montre le graphe principal obtenu.

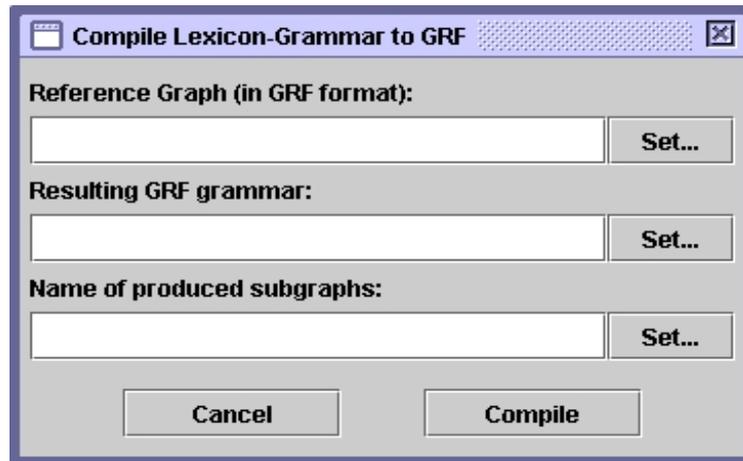
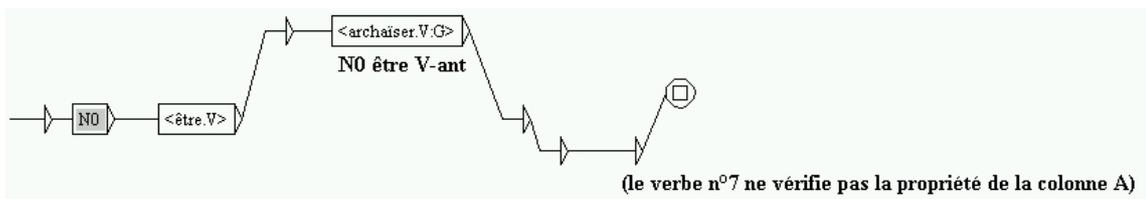
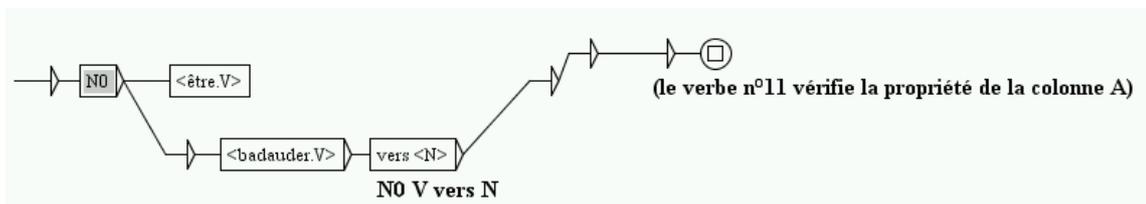
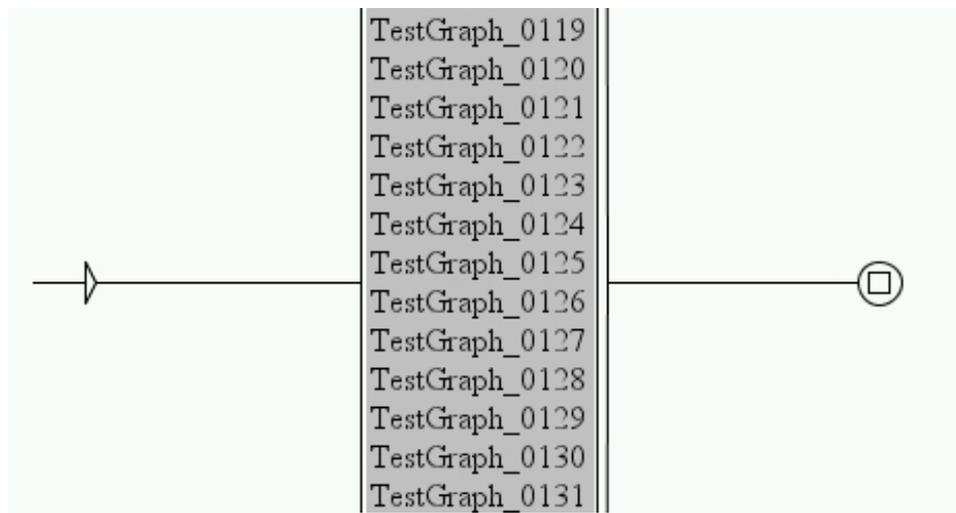


FIG. 8.7 – Configuration de la génération automatique de graphes

FIG. 8.8 – Graphe généré pour le verbe *archaïser*FIG. 8.9 – Graphe généré pour le verbe *badauder*

FIG. 8.10 – *Grappe principale appelant tous les graphes générés*

Chapitre 9

Utilisation des programmes externes

Ce chapitre présente l'utilisation des différents programmes qui composent Unitex. Ces programmes, qui se trouvent dans le répertoire `Unitex/App`, sont appelés automatiquement par l'interface. Vous pouvez voir les commandes qui ont été exécutées en cliquant sur "Console" dans le menu "Info". Vous pouvez également voir les options des différents programmes en les sélectionnant dans le sous-menu "Help on commands" du menu "Info".

IMPORTANT: plusieurs programmes utilisent le répertoire du texte (`mon_texte_snt`). Ce répertoire est créé par l'interface graphique après la normalisation du texte. Si vous travaillez en ligne de commande, vous devez créer ce répertoire vous-même après l'exécution du programme `Normalize`.

IMPORTANT (2): lorsqu'un paramètre contient des espaces, vous devez l'entourer de guillemets pour qu'il ne soit pas considéré comme plusieurs paramètres.

9.1 CheckDic

`CheckDic dictionnaire type`

Ce programme effectue la vérification du format d'un dictionnaire de type DELAS ou DELAF. Le paramètre `dictionnaire` correspond au nom du dictionnaire à vérifier. Le paramètre `type` peut prendre la valeur DELAS ou DELAF selon que l'on souhaite vérifier un dictionnaire de l'un ou l'autre de ces formats.

Le programme teste la syntaxe des lignes du dictionnaire. Il dresse également la liste des caractères présents dans les formes fléchies et canoniques, la liste des codes grammaticaux et syntaxiques ainsi que la liste des codes flexionnels utilisés. Les résultats de la vérification sont stockés dans un fichier nommé `CHECK_DIC.TXT`.

9.2 Compress

`Compress dictionnaire [-flip]`

Ce programme prend en paramètre un dictionnaire DELAF et le compresse. La compression d'un dictionnaire `dico.dic` produit deux fichiers:

- `dico.bin`: fichier binaire contenant l'automate minimal des formes fléchies du dictionnaire;
- `dico.inf`: fichier texte contenant des formes comprimées permettant de reconstruire les lignes du dictionnaire à partir des formes fléchies contenues dans l'automate.

Pour plus de détails sur les formats de ces fichiers, voir chapitre 10. Le paramètre optionnel `-flip` indique que les formes fléchies et canoniques seront inversées dans le dictionnaire comprimé. Cette option est utilisée pour construire le dictionnaire inversé nécessaire au programme `Reconstrucao`.

9.3 Concord

```
Concord index font fontsize left right order mode alph [-thai]
```

Ce programme prend en paramètre un fichier d'index de concordance produit par le programme `Locate` et produit une concordance. Il peut également produire une version du texte modifiée prenant en compte les transductions associées aux occurrences. Voici la description des paramètres:

- `index`: nom du fichier de concordance. Vous devez indiquer le chemin d'accès complet à ce fichier, car Unitex s'en sert pour déterminer sur quel texte la concordance doit être calculée.
- `font`: nom de la police de caractères à utiliser si la concordance doit être produite au format HTML. Si la concordance n'est pas au format HTML, ce paramètre est ignoré.
- `fontsize`: taille de la police si la concordance est au format HTML. Comme le paramètre `font`, celui-ci est ignoré si la concordance n'est pas au format HTML.
- `left`: nombre de caractères du contexte gauche des occurrences. En mode `thai`, il s'agit du nombre de caractères non diacritiques.
- `right`: nombre de caractères du contexte droit (non diacritiques, dans le cas du `thai`). Si l'occurrence a une longueur inférieure à cette valeur, la ligne de concordance est complétée pour que le contexte droit ait une longueur égale à `right`. Si l'occurrence a une longueur de plus de `right` caractères, elle est néanmoins affichée en entier.
- `order`: indique le mode de tri à utiliser pour ordonner les lignes de la concordance. Les valeurs possibles sont:
 - `TO`: ordre dans lequel les occurrences apparaissent dans le texte;
 - `LC`: contexte gauche, occurrence;
 - `LR`: contexte gauche, contexte droit;
 - `CL`: occurrence, contexte gauche;
 - `CR`: occurrence, contexte droit;
 - `RL`: contexte droit, contexte gauche;
 - `RC`: contexte droit, occurrence.

- **NULL**: ne précise aucun ordre de tri. Cette option doit être utilisée lorsque l'on souhaite modifier le texte au lieu de construire une concordance.

Pour plus de détails sur ces mode de tri, voir la section 4.8.2.

- **mode**: indique sous quel format la concordance doit être produite. Les 4 modes possibles sont:
 - **html**: produit une concordance au format HTML codée en UTF-8;
 - **text**: produit une concordance au format texte unicode;
 - **glossanet**: produit une concordance pour GlossaNet au format HTML. Le fichier HTML produit est codé en UTF-8;
 - **nom_de_fichier**: indique au programme qu'il doit produire une version modifiée du texte et la sauver dans un fichier nommé **nom_de_fichier** (voir section 6.6.3).
- **alph**: fichier alphabet utilisé pour le tri. La valeur **NULL** indique l'absence de fichier d'alphabet.
- **-thai**: ce paramètre est facultatif. Il indique au programme qu'il manipule du texte thaï. Cette option est nécessaire au bon fonctionnement du programme sur des textes en thaï.

Le résultat de l'application de ce programme est un fichier **concord.txt** si la concordance a été construite en mode texte, un fichier **concord.html** pour les modes **html** et **glossanet**, et un fichier texte dont le nom a été défini par l'utilisateur si le programme a construit une version modifiée du texte.

En mode **html**, l'occurrence est codée comme un lien. La référence associée à ce lien est de la forme ``. X et Y représentent les positions de début et de fin de l'occurrence en caractères dans le fichier **nom_du_texte.snt**. Z représente le numéro de la phrase dans laquelle apparaît l'occurrence.

9.4 Convert

```
Convert src [dest] mode text_1 [text_2 text_3 ...]
```

Ce programme permet de changer le codage de fichiers texte. Le paramètre **src** indique le codage d'entrée. Le paramètre optionnel **dest** indique le codage de sortie. Par défaut, le codage de sortie est **LITTLE-ENDIAN**. Les valeurs possibles pour ces paramètres sont les suivantes:

```
FRENCH
ENGLISH
GREEK
THAI
CZECH
GERMAN
SPANISH
PORTUGUESE
ITALIAN
```

NORWEGIAN

LATIN (page de codes latine par défaut)

windows-1252: page de codes Microsoft Windows 1252 - Latin I (Europe de l'ouest & USA)

windows-1250: page de codes Microsoft Windows 1250 - Europe centrale

windows-1257: page de codes Microsoft Windows 1257 - Baltique

windows-1251: page de codes Microsoft Windows 1251 - Cyrillique

windows-1254: page de codes Microsoft Windows 1254 - Turc

windows-1258: page de codes Microsoft Windows 1258 - Viet Nam

iso-8859-1 : page de codes ISO 8859-1 - Latin 1 (Europe de l'ouest & USA)

iso-8859-15: page de codes ISO 8859-15 - Latin 9 (Europe de l'ouest & USA)

iso-8859-2 : page de codes ISO 8859-2 - Latin 2 (Europe de l'est et centrale)

iso-8859-3 : page de codes ISO 859-3 - Latin 3 (Europe du sud)

iso-8859-4 : page de codes ISO 859-4 - Latin 4 (Europe du nord)

iso-8859-5 : page de codes ISO 8859-5 - Cyrillique

iso-8859-7 : page de codes ISO 8859-7 - Grec

iso-8859-9 : page de codes ISO 8859-9 - Latin 5 (Turc)

iso-8859-10: page de codes ISO 8859-10 - Latin 6 (Nordique)

next-step : page de codes NextStep

LITTLE-ENDIAN

BIG-ENDIAN

NOTE: il y a un mode supplémentaire pour le paramètre **dest** avec la valeur **UTF-8**, qui indique au programme qu'il doit convertir les fichiers Unicode Little-Endian en fichiers UTF-8.

Le paramètre **mode** spécifie comment gérer les noms des fichiers source et destination. Les valeurs possibles sont les suivantes:

-r : la conversion écrase les fichiers source

-ps=AFX : les fichiers source sont renommés avec le préfixe AFX (**toto.txt** ⇒ **AFXtoto.txt**)

-pd=AFX : les fichiers destination sont renommés avec le préfixe AFX

-ss=SFX : les fichiers source sont renommés avec le suffixe SFX (**toto.txt** ⇒ **totoSFX.txt**)

-sd=SFX : les fichiers destination sont renommés avec le suffixe SFX

Les paramètres **text_i** sont les noms des fichiers à convertir.

9.5 Dico

Dico **texte** **alphabet** **dic_1** [**dic_2** ...]

Ce programme applique des dictionnaires à un texte. Le texte doit avoir été découpé en unités lexicales par le programme **Tokenize**. Les dictionnaires doivent avoir été compressés avec le programme **Compress**.

texte représente le chemin d'accès complet au fichier texte, sans omettre l'extension **.snt**.

dic_i représente le chemin d'accès complet à un dictionnaire. Le dictionnaire doit porter l'extension **.bin**. Il est possible de donner des priorités aux dictionnaires. Pour plus de détails, voir section 3.6.1.

Le programme `Dico` produit les 4 fichiers suivants et les sauve dans le répertoire du texte:

- `dlf`: dictionnaire des mots simples du texte;
- `dlc`: dictionnaire des mots composés du texte;
- `err`: liste des mots inconnus du texte;
- `stat_dic.n`: fichier contenant les nombres de mots simples, composés et inconnus du texte.

NOTE: les fichiers `dlf`, `dlc` et `err` ne sont pas triés. Utilisez le programme `SortTxt` pour le faire.

9.6 Elag

```
Elag txtauto -l lang -g rules -o output [-d dir]
```

Ce programme prend un automate de texte `txtauto` et lui applique des règles de levée d'ambiguïtés. Les paramètres sont les suivants:

- `txtauto`: l'automate du texte au format `.fst2`
- `lang`: le fichier de configuration ELAG pour la langue considérée
- `rules`: le fichier de règles compilées au format `.rul`
- `output`: l'automate du texte de sortie
- `dir`: ce paramètre optionnel indique le répertoire dans lequel se trouvent les règles ELAG

9.7 ElagComp

```
ElagComp [ -r ruleslist | -g grammar ] -l lang [ -o output ] [ -d rulesdir ]
```

Ce programme compile une grammaire ELAG dont le nom est `grammar`, ou toutes les grammaires spécifiées dans le fichier `ruleslist`. Le résultat est stocké dans un fichier `output` qui pourra être utilisé par le programme `Elag`.

- `ruleslist`: fichier listant des grammaires ELAG
- `lang`: le fichier de configuration ELAG pour la langue considérée
- `output`: (optionnel) nom du fichier de sortie. Par défaut, le fichier de sortie est identique à `ruleslist`, sauf pour l'extension qui est `.rul`
- `rulesdir`: ce paramètre optionnel indique le répertoire dans lequel se trouvent les règles ELAG

9.8 Evamb

```
Evamb [ -imp | -exp ] [-o] fstname [ -n sentenceno ]
```

Ce programme calcule un taux d'ambiguïté moyen sur tout l'automate du texte `fstname`, ou juste sur la phrase spécifiée par `sentenceno`. Si le paramètre `-imp` est spécifié, le programme effectue le calcul sur une forme dite *compacte* de l'automate dans laquelle les ambiguïtés flexionnelles ne sont pas prises en compte. Si c'est le paramètre `-exp` qui est spécifié, toutes les ambiguïtés flexionnelles sont considérées; on parle alors de la forme *développée* de l'automate du texte. Ainsi, l'entrée `aimable, .A:ms:fs` ne comptera qu'une seule fois avec `-imp`, et deux fois avec `-exp`. L'automate du texte n'est pas modifié par ce programme.

9.9 ExploseFst2

```
ExploseFst2 txtauto -o out
```

Ce programme calcule et stocke dans `out`, la forme *développée* de l'automate de texte `txtauto`.

9.10 Extract

```
Extract yes/no texte concordance resultat
```

Ce programme prend en paramètre un texte et un fichier de concordance. Si le premier paramètre vaut `yes`, le programme extrait de ce texte toutes les phrases qui contiennent au moins une des occurrences de la concordance. Si ce paramètre vaut `no`, le programme extrait toutes les phrases qui ne contiennent aucune des occurrences.

Le paramètre `texte` doit représenter le chemin d'accès complet au fichier texte, sans omettre l'extension `.snt`.

Le paramètre `concordance` doit représenter le chemin d'accès complet au fichier de concordance, sans omettre l'extension `.ind`.

Le paramètre `resultat` représente le nom du fichier dans lequel seront sauvées les phrases extraites.

Le fichier `resultat` est un fichier texte contenant toutes les phrases extraites, à raison d'une phrase par ligne.

9.11 Flatten

```
Flatten fst2 type [depth]
```

Ce programme prend en paramètre une grammaire quelconque, et essaye de la transformer en un transducteur à états finis. Le paramètre `fst2` désigne la grammaire à transformer. Le paramètre `type` indique le type de grammaire attendue en résultat. Si ce paramètre vaut `FST`, la grammaire sera "dépliée" jusqu'à la profondeur maximum et sera tronquée s'il reste

des appels à des sous-graphes. Le résultat sera une grammaire au format `.fst2` ne contenant qu'un seul transducteur à états finis. Si le paramètre vaut `RTN`, les appels aux sous-graphes qui pourraient rester après tranformation sont laissés tels quels. Le résultat est donc un transducteur à états finis dans les cas favorables, et une grammaire optimisée strictement équivalente à la grammaire d'origine sinon. Le paramètre optionnel `depth` indique la profondeur maximum d'imbrication des sous-graphes qui sera gérée par le programme. La valeur par défaut est 10.

9.12 Fst2Grf

```
Fst2Grf automate_du_texte phrase
```

Ce programme extrait de l'automate d'un texte l'automate d'une phrase au format `.grf`. Le paramètre `automate_du_texte` représente le chemin d'accès complet à l'automate du texte duquel on veut extraire une phrase. Ce fichier s'appelle `text.fst2` et se trouve dans le répertoire du texte. Le paramètre `phrase` indique le numéro de la phrase à extraire.

Ce programme produit les 2 fichiers suivants et les sauve dans le répertoire du texte:

- `currentence.grf`: graphe représentant l'automate de la phrase;
- `currentence.txt`: fichier texte contenant cette phrase.

9.13 Fst2List

```
Fst2List [-o out] [-p s/f/d] [-[a/t] s/m] [-f s/a] [-s "L,[R]"] [-s0 "Str"] [-v]
        [-r[s/l/x] "L,[R]"] [-l line#] [-i subname]* [-c SS=0xxxx]* fname
```

Ce programme prend un fichier `.fst2` et produit la liste des séquences reconnues par cette grammaire. Les paramètres sont les suivants:

- `fname`: nom de la grammaire, avec l'extension `.fst2`;
- `-o out`: précise le nom du fichier de sortie. Par défaut, ce fichier se nomme `lst.txt`;
- `-[a/t] s/m`: précise si l'on tient compte (`a`) ou non (`t`) des éventuelles sorties de la grammaire. `s` indique qu'il n'y a qu'un seul état initial, tandis que `m` indique qu'il y en a plusieurs (ce mode est utile en coréen). Par défaut, ce paramètre vaut `-a s`;
- `-l line#`: nombre maximum de lignes à écrire dans le fichier de sortie;
- `-i subname`: indique que l'on doit arrêter l'exploration récursive lorsque l'on rencontre le graphe `sname`. Ce paramètre peut être utilisé plusieurs fois, afin de spécifier plusieurs graphes d'arrêt;
- `-p s/f/d`: `s` produit l'affichage des chemins de chaque sous-graphe de la grammaire; `f` (défaut) affiche les chemins de la grammaire globale; `d` affiche les chemins en ajoutant des indications sur les imbrications d'appels de sous-graphes;
- `-c SS=0xXXXX`: remplace le symbole `SS` quand il apparaît entre entre angles par un caractère unicode (`0xXXXX`);

- **-s "L[,R]"** : spécifie les délimiteurs gauche (L) et droit (R) qui entoureront les items. Par défaut, ces délimiteurs sont nuls;
- **-s0 "Str"** : si l'on tient compte des sorties de la grammaire, ce paramètre spécifie la séquence **Str** qui séparera une entrée de sa sortie. Par défaut, il n'y a pas de séparateur;
- **-f a/s** : si l'on tient compte des sorties de la grammaire, ce paramètre spécifie le format des lignes générées: **in0 in1 out0 out1 (s)** ou bien **in0 out0 in1 out1 (a)**. La valeur par défaut est **s**;
- **-v** : ce paramètre produit l'affichage de messages d'informations;
- **-r[s/l/x] "L[,R]"** : ce paramètre spécifie comment les cycles doivent être présentés. L et R désignent des délimiteurs. Si l'on considère le graphe de la figure 9.1, voici les résultats que l'on obtient si l'on pose L="[" et R="]":

```
-rs "[,]" il fait [très|C0]beau
           C0::très (C0 désigne un label généré par le programme)
-rl "[,]" il fait très [Loc0]
           Loc0très
           Loc0beau (Loc0 désigne un label généré par le programme)
-rx "[,]" il fait [très très ]*
           il fait très beau
```



FIG. 9.1 – Graphe avec cycle

9.14 Fst2Txt

Fst2Txt *texte* *fst2* *alphabet* *mode* [-*char_by_char*]

Ce programme applique un transducteur à un texte en phase de prétraitement, quand le texte n'est pas encore découpé en unités lexicales. Les paramètres de ce programme sont les suivants:

- **texte**: le fichier texte à modifier, avec l'extension **.snt**;
- **fst2**: le transducteur à appliquer;
- **alphabet**: le fichier alphabet de la langue du texte;
- **mode**: le mode d'application du transducteur. Les deux valeurs possibles sont **-merge** et **-replace**;
- **-char_by_char**: ce paramètre facultatif permet d'appliquer le transducteur en mode caractère par caractère. Cette option doit être utilisée pour les textes en langues asiatiques.

Ce programme a pour effet de modifier le fichier texte passé en paramètre.

9.15 Grf2Fst2

```
Grf2Fst2 graphe [y/n]
```

Ce programme compile une grammaire en un fichier `.fst2` (pour plus de détails, voir section 6.2). Le paramètre `graphe` désigne le chemin d'accès complet au graphe principal de la grammaire, sans omettre l'extension `.grf`.

Le second paramètre est optionnel; il indique au programme s'il doit ou non effectuer une recherche d'erreur sur la grammaire. Par défaut, le programme effectue cette recherche d'erreur.

Le résultat est un fichier portant le même nom que le graphe passé en paramètre, mais avec l'extension `.fst2`. Ce fichier est sauvegardé dans le même répertoire que `graphe`.

9.16 ImplouseFst2

```
ImplouseFst2 txtauto -o out
```

Ce programme calcule et stocke dans `out`, la forme *compacte* de l'automate de texte `txtauto`.

9.17 Inflect

```
Inflect delas resultat rep
```

Ce programme effectue la flexion automatique d'un dictionnaire DELAS. Le paramètre `delas` indique le nom du dictionnaire à fléchir. Le paramètre `resultat` indique le nom du dictionnaire qui sera généré. Le paramètre `rep` indique le chemin d'accès complet au répertoire dans lequel sont supposés se trouver les transducteurs de flexion auxquels le dictionnaire `delas` fait référence.

Le résultat de la flexion est un dictionnaire DELAF sauvegardé sous le nom indiqué par le paramètre `resultat`.

9.18 Locate

```
Locate texte fst2 alphabet s/l/a i/m/r n [-thai] [-space]
```

Ce programme applique une grammaire à un texte et construit un fichier d'index des occurrences trouvées. Ses paramètres sont les suivants:

- `texte`: chemin d'accès complet au fichier texte, sans omettre l'extension `.snt`;
- `fst2`: chemin d'accès complet à la grammaire, sans omettre l'extension `.fst2`;
- `alphabet`: chemin d'accès complet au fichier alphabet;
- `s/l/a`: paramètre indiquant si la recherche doit se faire en mode *shortest matches* (`s`), *longest matches* (`l`) ou *all matches* (`a`);

- **i/m/r**: paramètre indiquant le mode d'application des transductions: mode MERGE (**m**) ou mode REPLACE (**r**). **i** indique que le programme ne doit pas tenir compte des transductions;
- **n**: paramètre indiquant le nombre d'occurrences à rechercher. La valeur **all** indique au programme qu'il doit rechercher toutes les occurrences;
- **-thai**: paramètre optionnel nécessaire pour une recherche dans un texte thaï;
- **-space**: paramètre optionnel indiquant au programme qu'il peut démarrer les recherches de motifs sur les espaces. Ce paramètre ne doit être employé que pour effectuer des recherches de motifs morphologiques.

Ce programme sauvegarde les références des occurrences trouvées dans un fichier nommé `concord.ind`. Le nombre d'occurrences, le nombre d'unités couvertes par ces occurrences ainsi que le pourcentage d'unités reconnues dans le texte sont sauvegardés dans un fichier nommé `concord.n`. Ces deux fichiers sont sauvegardés dans le répertoire du texte.

9.19 MergeTextAutomaton

```
MergeTextAutomaton automaton
```

Ce programme reconstruit l'automate du texte `automaton` en prenant en compte les modifications manuelles qui ont été faites. Ainsi, si le programme trouve un fichier `sentenceN.grf` dans le même répertoire que `automaton`, il va remplacer l'automate de la phrase N par celui qui est représenté par `sentenceN.grf`. Le fichier `automaton` est remplacé par le nouvel automate du texte. L'ancien automate du texte est sauvegardé dans un fichier nommé `text.fst2.bck`.

9.20 Normalize

```
Normalize txt
```

Ce programme effectue une normalisation des séparateurs du texte. Les séparateurs sont l'espace, la tabulation et le retour à la ligne. Toute suite de séparateurs contenant au moins un retour à la ligne est remplacée par un unique retour à la ligne. Toute autre suite de séparateurs est remplacée par un espace.

Ce programme vérifie également la syntaxe des étiquettes lexicales présentes dans le texte. Toute séquence entre accolades doit être soit le délimiteur de phrase `{S}`, soit une ligne de DELAF valide (`{aujourd'hui, .ADV}`). Si le programme détecte des accolades employées différemment, il émet un message d'avertissement et remplace ces accolades par des crochets (`[et]`).

Le paramètre `txt` doit représenter le chemin d'accès complet au fichier du texte. Le programme produit une version modifiée du texte qui est sauvé dans un fichier portant l'extension `.snt`.

9.21 PolyLex

```
PolyLex lang alph dic liste out [info]
```

Ce programme prend en paramètre un fichier de mots inconnus `liste` et essaye d'analyser chacun d'eux comme un mot composé obtenu par soudure de mots simples. Les mots qui ont au moins une analyse sont retirés du fichier de mots inconnus et les lignes de dictionnaire correspondant aux analyses sont ajoutées au fichier `out`. Le paramètre `lang` détermine la langue de travail. Les deux valeurs possibles sont `GERMAN` et `NORWEGIAN`. Le paramètre `alph` représente le fichier alphabet à utiliser. Le paramètre `dic` désigne le dictionnaire à consulter pour l'analyse. Le paramètre `out` désigne le fichier dans lequel seront écrites les lignes de dictionnaires produites; si ce fichier existe déjà, les lignes produites sont ajoutées à la fin de ce fichier. Le paramètre optionnel `info` désigne un fichier texte dans lequel sont produites des informations sur les analyses effectuées.

9.22 Reconstrucao

```
Reconstrucao alph concord dic reverse_dic pro res
```

Ce programme génère une grammaire de normalisation destinée à être appliquée lors de la construction de l'automate d'un texte portugais. Le paramètre `alph` désigne le fichier alphabet à utiliser. Le fichier `concord` représente une concordance qui doit avoir été produite par l'application en mode `MERGE` au texte considéré d'une grammaire extrayant toutes les formes à normaliser. Cette grammaire se nomme `V-Pro-Suf`, et se trouve dans le répertoire `/Portuguese/Graphs/Normalization`. Le paramètre `dic` désigne le dictionnaire à utiliser pour retrouver les formes canoniques associées aux radicaux des verbes. `reverse_dic` désigne le dictionnaire inversé à utiliser pour retrouver les formes au futur et au conditionnel à partir des formes canoniques. Ces deux dictionnaires doivent être au format `.bin`, et `reverse_dic` devrait avoir été obtenu en compressant le dictionnaire des verbes au futur et au conditionnel avec le paramètre `-flip` (voir section 9.2). Le paramètre `pro` désigne la grammaire de réécriture des pronoms à utiliser. `res` désigne le fichier `.grf` dans lequel seront produites les règles de normalisation.

9.23 Reg2Grf

```
Reg2Grf fic
```

Ce programme construit un fichier `.grf` correspondant à l'expression rationnelle contenue dans le fichier `fic`. Le paramètre `fic` doit représenter le chemin d'accès complet au fichier contenant l'expression rationnelle. Ce fichier doit être un fichier texte Unicode. Le programme prend en compte tous les caractères jusqu'au premier retour à ligne. Le fichier résultat se nomme `regexp.grf` et est sauvegardé dans le même répertoire que `fic`.

9.24 SortTxt

`SortTxt texte [OPTIONS]`

Ce programme effectue un tri lexicographique des lignes du fichier `texte`. `texte` doit représenter le chemin d'accès complet au fichier à trier. Les options possibles sont:

- `-y`: supprime les doublons;
- `-n`: conserve les doublons;
- `-r`: trie dans l'ordre décroissant;
- `-o fic`: trie en utilisant l'alphabet de tri défini par le fichier `fic`. Si ce paramètre est absent, le tri est effectué selon l'ordre des caractères en Unicode;
- `-l fic`: sauvegarde le nombre de lignes du fichier résultat dans le fichier `fic`;
- `-thai`: option à utiliser pour trier un texte thaï.

L'opération de tri modifie le fichier `texte`. Par défaut, le tri est effectué dans l'ordre des caractères en Unicode, en supprimant les doublons.

9.25 Table2Grf

`Table2Grf table graphe resultat [sousgraphe]`

Ce programme génère automatiquement des graphes à partir de la table de lexique-grammaire `table` et du graphe patron `graphe`. Le nom du graphe principal de la grammaire obtenue est `resultat`. Les noms des sous-graphes produits sont générés à partir du modèle `sousgraphe`. Si ce paramètre est omis, les noms des graphes générés sont formés à partir du paramètre `resultat` auquel s'ajoute un nombre.

9.26 TextAutomaton2Mft

`TextAutomaton2Mft text.fst2`

Ce programme prend en paramètre un automate du texte `text.fst2` et construit un équivalent au format `.mft` d'Intex. Le fichier produit a pour nom `text.mft` et est codé en Unicode.

9.27 Tokenize

`Tokenize texte alphabet [-char_by_char]`

Ce programme découpe le texte en unités lexicales. Le paramètre `texte` doit représenter le chemin d'accès complet au fichier `texte`, sans omettre l'extension `.snt`. Le paramètre `alphabet` doit représenter le chemin d'accès complet au fichier définissant l'alphabet de la langue du texte. Le paramètre optionnel `-char_by_char` indique au programme qu'il doit effectuer un découpage caractère par caractère, à l'exception du séparateur de phrases `{S}` et des étiquettes

lexicales qui seront considérés comme des unités. Sans ce paramètre, le programme considère qu'une unité est soit une suite de lettres (les lettres sont définies par le fichier `alphabet`), soit un caractère qui n'est pas une lettre, soit le séparateur de phrases `{S}`, soit une étiquette lexicale (`{aujourd'hui, .ADV}`).

Le programme code chaque unité par un entier. La liste des unités est sauvegardée dans un fichier texte nommé `tokens.txt`. La suite des codes représentant les unités permet alors de coder le texte. Cette suite est sauvegardée dans un fichier binaire nommé `text.cod`. Le programme produit également les 4 fichiers suivants:

- `tok_by_freq.txt`: fichier texte contenant la liste des unités triées par ordre de fréquence;
- `tok_by_alph.txt`: fichier texte contenant la liste des unités triées par ordre alphabétique;
- `stats.n`: fichier texte contenant des informations sur le nombre de séparateurs de phrases, le nombre d'unités, le nombre de mots simples et le nombre de chiffres;
- `enter.pos`: fichier binaire contenant la liste des positions des retours à la ligne dans le texte. La représentation codée du texte ne contient pas de retours à la ligne, mais des espaces. Comme un retour à la ligne compte pour 2 caractères et l'espace pour un seul, il faut savoir où se trouvent les retours à la ligne dans le texte si l'on veut synchroniser les positions des occurrences calculées par le programme `Locate` avec le fichier texte. Le fichier `enter.pos` est utilisé à cette fin par le programme `Concord`. C'est grâce à cela que lorsque l'on clique sur une occurrence dans une concordance, celle-ci est correctement sélectionnée dans le texte.

Tous les fichiers produits sont sauvegardés dans le répertoire du texte.

9.28 Txt2Fst2

```
Txt2Fst2 texte alphabet [-clean] [norm]
```

Ce programme construit l'automate du texte. Le paramètre `texte` doit représenter le chemin d'accès complet au fichier texte, sans omettre l'extension `.snt`. Le paramètre `alphabet` doit représenter le chemin d'accès complet au fichier alphabet de la langue du texte. Le paramètre optionnel `-clean` indique au programme qu'il doit appliquer le principe de conservation des meilleurs chemins (voir section 7.2.4). Si le paramètre `norm` est précisé, il est interprété comme le nom d'une grammaire de normalisation à appliquer à l'automate du texte.

Si le texte a été découpé en phrases, le programme construit un automate pour chaque phrase. Si ce n'est pas le cas, le programme découpe arbitrairement le texte en séquences de 2000 unités lexicales et construit un automate pour chacune de ces séquences.

Le résultat est un fichier nommé `text.fst2` qui est sauvegardé dans le répertoire du texte.

Chapitre 10

Formats de fichiers

Ce chapitre présente les formats des différents fichiers lus ou générés par Unitex. Les formats des dictionnaires DELAS et DELAF sont déjà présentés aux sections 3.1.1 et 3.1.2.

NOTE: dans ce chapitre, le symbole ¶ représentera le retour à la ligne. Sauf indication contraire, tous les fichiers texte décrits dans ce chapitre sont codés en Unicode Little-Endian.

10.1 Codage Unicode Little-Endian

Tous les fichiers textes manipulés par Unitex doivent être en Unicode Little-Endian. Ce codage permet de représenter 65536 caractères en les codant chacun sur 2 octets. En Little-Endian, les octets sont dans l'ordre poids faible-poids fort. Quand cet ordre est inversé, on parle de codage Big-Endian. Un fichier texte codé en Unicode Little-Endian commence par le caractère spécial de valeur hexadécimale FFFF. Les retours à la ligne doivent être codés par les deux caractères 000D et 000A.

Considérons le texte suivant:

```
Unitex¶
β-version¶
```

Voici la représentation en Unicode Little-Endian de ce texte:

en-tête	U	n	i	t	e	x	¶	β
FFFE	5500	6E00	6900	7400	6500	7800	0D000A00	B203
-	v	e	r	s	i	o	n	¶
2D00	7600	6500	7200	7300	6900	6F00	6E00	0D000A00

TAB. 10.1 – Représentation hexadécimale d'un texte Unicode

Les octets de poids fort et de poids faible ont été inversés, ce qui explique que le caractère d'en-tête soit codé par FFFE au lieu de FFFF, idem pour 000D et 000A qui sont devenus 0D00 et 0A00.

10.2 Fichiers d'alphabet

Il y a deux sortes de fichiers d'alphabet: un fichier qui définit les caractères d'une langue et un fichier indiquant des préférences pour le tri. Le premier est désigné sous le terme d'*alphabet*, et le second sous celui d'*alphabet de tri*.

10.2.1 Alphabet

Le fichier d'alphabet est un fichier texte décrivant tous les caractères d'une langue, ainsi que les correspondances entre lettres minuscules et majuscules. Ce fichier doit s'appeler `Alphabet.txt` et doit se trouver dans la racine du répertoire de la langue concernée. Sa présence est obligatoire pour qu'Unitex puisse fonctionner.

Exemple: le fichier d'alphabet de l'anglais doit se trouver dans le répertoire `.../English/`

Chaque ligne du fichier alphabet doit avoir l'une des 3 formes suivantes, suivie par un retour à la ligne:

- `#가힣` : un dièse suivi de 2 caractères *X* et *Y* indique que tous les caractères compris entre les caractères *X* et *Y* sont des lettres. Tous ces caractères sont considérés comme étant à la fois minuscules et majuscules. Ce mode est utile pour définir les alphabets des langues asiatiques comme le coréen, le chinois ou le japonais où il n'y a pas de distinction de casse et où le nombre de caractères rendrait très fastidieuse une énumération complète;
- `Ëë` : 2 caractères *X* et *Y* indiquent que *X* et *Y* sont des lettres et que *X* est l'équivalent en majuscule de la lettre *Y*;
- `Ⓐ` : un unique caractère *X* définit *X* comme une lettre à la fois minuscule et majuscule. Ce mode est utile pour définir un caractère asiatique de manière ponctuelle.

Pour certaines langues comme le français, il arrive qu'à une lettre minuscule correspondent plusieurs majuscules, comme c'est le cas pour le `é`, qui peut avoir comme majuscule soit `E`, soit `É`. Pour exprimer cela, il suffit d'utiliser plusieurs lignes. L'inverse est également vrai: à une majuscule peuvent correspondre plusieurs minuscules. Ainsi, le `E` peut être la majuscule de `e`, `é`, `è`, `ê` ou `ë`. Voici l'extrait du fichier alphabet du français qui définit les différentes lettres `e`:

```
Ee¶
Eé¶
Éé¶
Eè¶
Èè¶
Eê¶
Êê¶
Eë¶
Ëë¶
```

10.2.2 Alphabet de tri

L'alphabet de tri est un fichier texte qui définit les priorités des lettres d'une langue lors du tri à l'aide du programme `SortTxt`. Chaque ligne de ce fichier définit un groupe de lettres. Si un groupe de lettres *A* est défini avant un groupe de lettres *B*, n'importe quelle lettre de *A* sera inférieure à n'importe quelle lettre de *B*.

Les lettres d'un même groupe ne sont distinguées que si nécessaire. Par exemple, si l'on a défini le groupe de lettre `eéêëë`, le mot `ébahi` sera considéré comme plus petit que `estuaire`, lui-même plus petit que `été`. Comme les lettres qui suivent `e` et `é` permettraient de classer les mots, on n'a pas cherché à comparer les lettres `e` et `é` car elles sont du même groupe.

En revanche, si l'on compare les mots `chantés` et `chantes`, `chantes` sera considéré comme plus petit. En effet, il faut comparer les lettres `e` et `é` pour distinguer ces mots. Comme la lettre `e` apparaît en premier dans le groupe `eéêëë`, elle est considérée comme inférieure à `é`. Le mot `chantes` sera donc considéré comme plus petit que le mot `chantés`.

Le fichier d'alphabet de tri permet de définir des équivalences de caractères. On peut donc ignorer les différences de casse et d'accent. Par exemple, si l'on veut ordonner les lettres `b`, `c` et `d` sans tenir compte de la casse ni de la cédille, on peut écrire les lignes suivantes:

```
Bb¶
Ccçç¶
Dd¶
```

Ce fichier est facultatif. Lorsqu'aucun alphabet de tri n'est spécifié au programme `SortTxt`, celui-ci effectue un tri dans l'ordre d'apparition des caractères dans le codage Unicode.

10.3 Graphes

Cette section présente les deux formats de graphes: le format graphique `.grf` et le format compilé `.fst2`.

10.3.1 Format `.grf`

Un fichier `.grf` est un fichier texte contenant des informations de présentation en plus des informations représentant les contenus des boîtes et les transitions du graphe. Un fichier `.grf` commence par les lignes suivantes:

```
#Unigraph¶
SIZE 1313 950¶
FONT Times New Roman: 12¶
OFONT Times New Roman:B 12¶
BCOLOR 16777215¶
FCOLOR 0¶
ACOLOR 12632256¶
SCOLOR 16711680¶
CCOLOR 255¶
```

```

DBOXES y¶
DFRAME y¶
DDATE y¶
DFILE y¶
DDIR y¶
DRIG n¶
DRST n¶
FITS 100¶
PORIENT L¶
#¶

```

La première ligne `#Unigraph` est une ligne de commentaire. Les lignes suivantes définissent les valeurs des paramètres de présentation du graphe:

- `SIZE x y`: définit la largeur `x` et la hauteur `y` du graphe en pixels;
- `FONT name:xyz`: définit la police utilisée pour afficher le contenu des boîtes. `name` représente le nom de la police. `x` indique si la police doit être en gras ou non. Si `x` vaut `B`, cela indique que la police doit être en gras. Pour une police normale, `x` doit être un espace. De la même manière, `y` vaut `I` si la police doit être en italique, un espace sinon. `z` représente la taille de la police;
- `OFONT name:xyz`: définit la police utilisée pour afficher les transductions. Les paramètres `name`, `x`, `y` et `z` sont définis de la même manière que pour `FONT`;
- `BCOLOR x`: définit la couleur de l'arrière-plan du graphe. `x` représente la couleur au format RGB;
- `FCOLOR x`: définit la couleur de dessin du graphe. `x` représente la couleur au format RGB;
- `ACOLOR x`: définit la couleur utilisée pour dessiner les lignes des boîtes qui correspondent à des appels à des sous-graphes. `x` représente la couleur au format RGB;
- `SCOLOR x`: définit la couleur utilisée pour écrire le contenu des boîtes de commentaires (*i.e.* les boîtes qui ne sont reliées à aucune autre). `x` représente la couleur au format RGB;
- `CCOLOR x`: définit la couleur utilisée pour dessiner les boîtes sélectionnées. `x` représente la couleur au format RGB;
- `DBOXES x`: cette ligne est ignorée par Unitex. Elle est conservée par souci de compatibilité avec les graphes Intex;
- `DFRAME x`: dessine ou non un cadre autour du graphe selon que `x` vaut `y` ou `n`;
- `DDATE x`: affiche ou non la date en bas du graphe selon que `x` vaut `y` ou `n`;
- `DFILE x`: affiche ou non le nom du fichier en bas du graphe selon que `x` vaut `y` ou `n`;
- `DDIR x`: affiche ou non le chemin complet d'accès au fichier en bas du graphe selon que `x` vaut `y` ou `n`. Cette option n'est prise en compte que si la paramètre `DFILE` a la valeur `y`;
- `DRIG x`: dessine le graphe de droite à gauche ou de gauche à droite selon que `x` vaut `y` ou `n`;
- `DRST x`: cette ligne est ignorée par Unitex. Elle est conservée par souci de compatibilité avec les graphes Intex;

- FITS *x* : cette ligne est ignorée par Unitex. Elle est conservée par souci de compatibilité avec les graphes Intex;
- PORIENT *x* : cette ligne est ignorée par Unitex. Elle est conservée par souci de compatibilité avec les graphes Intex;
- # : cette ligne est ignorée par Unitex. Elle sert à indiquer la fin des informations d'en-tête.

Les lignes suivantes donnent le contenu et la position des boîtes du graphe. Les lignes suivantes correspondent à un graphe reconnaissant un chiffre:

```
3¶
"<E>" 84 248 1 2 ¶
"" 272 248 0 ¶
s"1+2+3+4+5+6+7+8+9+0" 172 248 1 1 ¶
```

La première ligne indique le nombre de boîtes du graphe, immédiatement suivi d'un retour à la ligne. Ce nombre ne doit jamais être inférieur à 2, car un graphe est toujours sensé posséder un état initial et un état final.

Les lignes suivantes définissent les boîtes du graphe. Les boîtes sont numérotées à partir de 0. Par convention, l'état 0 est l'état initial et l'état 1 est l'état final. Le contenu de l'état final doit toujours être vide.

Chaque boîte du graphe est définie par une ligne qui doit avoir le format suivant:

contenu *X Y N transitions* ¶

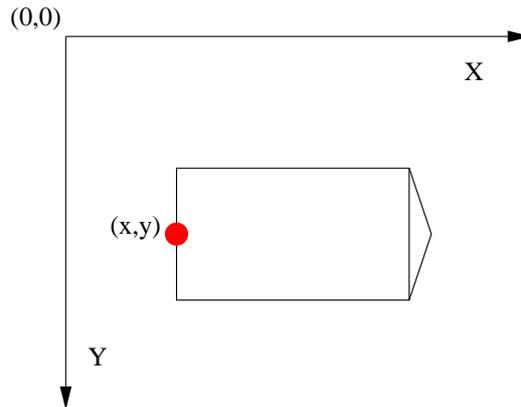
contenu est une chaîne de caractères entourée de guillemets qui représente le contenu de la boîte. Cette chaîne peut éventuellement être précédée d'un *s* dans le cas d'un graphe Intex importé; ce caractère est alors ignoré par Unitex. Le contenu de la chaîne est le texte qui a été entré dans le contrôle de texte de l'éditeur de graphes. Le tableau suivant donne le codage des deux séquences spéciales qui ne sont pas codées telles quelles dans les fichiers *.grf*:

Séquence dans l'éditeur de graphe	Séquence dans le fichier <i>.grf</i>
"	\"
\"	\\\"

TAB. 10.2 – Codage des séquences spéciales

NOTE: les caractères compris entre < et > ou entre { et } ne sont pas interprétés. Ainsi, le caractère + contenu dans la chaîne "le <A+Conc>" n'est pas interprété comme un séparateur de lignes, car le motif <A+Conc> est interprété en priorité.

X et *Y* représentent les coordonnées de la boîte en pixels. La figure 10.1 montre comment ces coordonnées sont interprétées par Unitex.

FIG. 10.1 – *Interprétation des coordonnées des boîtes*

N représente le nombre de transitions qui sortent de la boîte. Ce nombre doit toujours valoir 0 pour l'état final.

Les transitions sont définies par les numéros des boîtes vers lesquelles elles pointent.

Chaque ligne de définition de boîte doit se terminer par un espace suivi d'un retour à la ligne.

10.3.2 Format .fst2

Un fichier `.fst2` est un fichier texte qui décrit un ensemble de graphes. Voici un exemple de fichier `.fst2`:

```
0000000002
-1 GN
: 1 1
: 2 2 -2 2
: 3 3
t
f
-2 Adj
: 6 1 5 1 4 1
t
f
%<E>
%le/DET
%<A>/ADJ
%<N>
%beau
@joli
```

```
%petit¶
f¶
```

La première ligne représente le nombre de graphes codés dans le fichier. Le début de chaque graphe est identifié par une ligne indiquant le numéro et le nom du graphe (-1 GN et -2 Adj dans le fichier ci-dessus).

Les lignes suivantes décrivent les états du graphe. Si l'état est terminal, la ligne débute par le caractère **t** et par le caractère **:** sinon. Pour chaque état, la liste des transitions est une suite éventuellement vide de couples d'entiers:

- le premier entier indique le numéro d'étiquette ou de sous-graphe correspondant à la transition. Les étiquettes sont numérotées à partir de 0. Les sous-graphes sont représentés par des entiers négatifs, ce qui explique que les numéros précédant les noms des graphes soient négatifs;
- le deuxième entier représente le numéro de l'état d'arrivée de la transition. Dans chaque graphe, les états sont numérotés à partir de 0. Par convention, l'état 0 d'un graphe est son état initial.

Chaque ligne de définition d'état doit se terminer par un espace. La fin de chaque graphe est marquée par une ligne contenant un **f** suivi d'un espace.

Les étiquettes sont définies après le dernier graphe. Si la ligne débute par le caractère **@**, cela signifie que le contenu de l'étiquette doit être recherchée sans variante de casse. Cette information n'est utile que lorsque l'étiquette est un mot. Si la ligne débute par le caractère **%**, les variantes de casse sont autorisées. Si une étiquette porte une transduction, les séquences d'entrée et de sortie sont séparées par le caractère **/** (exemple: **1e/DET**). Par convention, la première étiquette doit toujours être le mot vide (**<E>**), et ce, même si cette étiquette n'est utilisée dans aucune transition.

La fin du fichier est indiquée par une ligne contenant le caractère **f** suivi d'un retour à la ligne.

10.4 Textes

Cette section présente les différents fichiers utilisés pour représenter des textes.

10.4.1 Fichiers .txt

Les fichiers **.txt** doivent être des fichiers texte codés en Unicode Little-Endian. Ces fichiers ne doivent pas contenir d'accolade ouvrante ou fermante, à moins qu'elles soient utilisées pour écrire un séparateur de phrase (**{S}**) ou une étiquette lexicale valide (**{aujourd'hui, .ADV}**). Les retours à la ligne doivent être codés par les deux caractères spéciaux de valeurs hexadécimales **000D** et **000A**.

10.4.2 Fichiers `.snt`

Les fichiers `.snt` sont des fichiers `.txt` qui ont été prétraités par Unitex. Ces fichiers ne doivent pas contenir de tabulation. Ils ne doivent pas non plus contenir plusieurs espaces ou retours à la ligne consécutifs. Les seules accolades autorisées dans des fichiers `.snt` sont celles du séparateur de phrases `{S}` et celles des étiquettes lexicales (`{aujourd'hui, .ADV}`)

10.4.3 Fichier `text.cod`

Le fichier `text.cod` est un fichier binaire contenant une suite d'entiers représentant le texte. Chaque entier i renvoie au token d'indice i dans le fichier `tokens.txt`. Ces entiers sont codés sur 4 octets.

NOTE: les tokens sont numérotés à partir de 0.

10.4.4 Fichier `tokens.txt`

Le fichier `tokens.txt` est un fichier texte contenant la liste de toutes les unités lexicales du texte. La première ligne de ce fichier indique le nombre d'unités contenus dans le fichier. Les unités sont séparées par des retours à la ligne. Quand une séquence est trouvée dans le texte avec des variantes de casse, chaque variante est codée par une unité distincte.

NOTE: les retours à la ligne éventuellement présents dans le fichier `.snt` sont codés comme des espaces. Il n'y a donc jamais d'unité codant le retour à la ligne.

10.4.5 Fichiers `tok_by_alph.txt` et `tok_by_freq.txt`

Ces deux fichiers sont des fichiers texte qui contiennent la liste des unités lexicales triée par ordre alphabétique ou par ordre de fréquence.

Dans le fichier `tok_by_alph.txt`, chaque ligne est composée d'une unité, suivie par le caractère tabulation et le nombre d'occurrences de cette unité dans le texte.

Les lignes du fichier `tok_by_freq.txt` sont formées sur le même principe, mais le nombre d'occurrences apparaît avant le caractère tabulation et l'unité.

10.4.6 Fichier `enter.pos`

Ce fichier est un fichier binaire contenant la liste des positions des retours à la ligne dans le fichier `.snt`. Chaque position est l'indice dans le fichier `text.cod` d'un retour à la ligne ayant été remplacé par un espace. Ces positions sont des entiers codés sur 4 octets.

10.5 Automate du texte

10.5.1 Fichier `text.fst2`

Le fichier `text.fst2` est un fichier `.fst2` spécial qui représente l'automate du texte. Dans ce fichier, chaque sous-graphe représente un automate de phrase. Les emplacements réservés

aux noms des sous-graphes sont utilisés pour stocker les phrases à partir desquelles ont été construits les automates de phrases.

À l'exception de la première étiquette qui doit toujours être epsilon (<E>), les étiquettes doivent être soit des unités lexicales, soit des entrées de DELAF encadrées par des accolades.

Exemple: Voici le fichier correspondant au texte *Il mange une pomme de terre.*

```
0000000001¶
-1 Il mange une pomme de terre. ¶
: 1 1 ¶
: 2 2 ¶
: 3 3 4 3 ¶
: 5 4 6 4 7 4 ¶
: 8 5 9 5 10 5 ¶
: 11 6 12 6 ¶
: 13 7 ¶
t ¶
f ¶
%<E>¶
%{Il,il.PRO+z1:3ms}¶
%{mange,manger.V+z1:P1s:P3s:S1s:S3s:Y2s}¶
%{une,une.N+z1:fs}¶
%{une,un.DET+z1:fs}¶
%{pomme,pomme.A+z1:ms:fs:mp:fp}¶
%{pomme,pomme.N+z1:fs}¶
%{pomme,pommer.V+z3:P1s:P3s:S1s:S3s:Y2s}¶
%{de,de.DET+z1}¶
%{de,de.PREP+z1}¶
%{terre,terre.N+z1:fs}¶
%{terre,terrifier.V+z1:P1s:P3s:S1s:S3s:Y2s}¶
%.¶
f¶
```

10.5.2 Fichier cursentence.grf

Le fichier `cursentence.grf` est généré par Unitex lors de l'affichage d'un automate de phrase. Le programme `Fst2Grf` construit un fichier `.grf` représentant l'automate d'une phrase à partir du fichier `text.fst2`.

10.5.3 Fichier sentenceN.grf

Lorsque l'utilisateur modifie l'automate d'une phrase, cet automate est sauvegardé sous le nom `sentenceN.grf`, où N représente le numéro de la phrase.

10.5.4 Fichier `cursorsentence.txt`

Lors de l'extraction de l'automate de phrase, le texte de la phrase est sauvegardé dans le fichier texte `cursorsentence.txt`. Ce fichier est utilisé par Unitex pour afficher le texte de la phrase au-dessus de l'automate. Ce fichier contient le texte de la phrase, suivi par un retour à la ligne.

10.6 Concordances

10.6.1 Fichier `concord.ind`

Le fichier `concord.ind` est l'index des occurrences trouvées par le programme `Locate` lors de l'application d'une grammaire. C'est un fichier texte qui contient les positions de début et de fin de chaque occurrence, éventuellement accompagnées d'une chaîne de caractères si la concordance a été obtenue en prenant en compte les éventuelles transductions de la grammaire. Voici un exemple de fichier:

```
#M
3036 3040 le[ADJ= petit] salon
3071 3075 Le nouveau domestique
5600 5604 le jeune Lord
6052 6056 le second étage
6123 6127 le premier étage
6181 6185 le même instant
6461 6465 le méthodique gentleman
7468 7472 le grand salon
7520 7524 le laborieux dépliage
7675 7679 le grand salon
8590 8594 le fait plus
10990 10994 le mauvais temps
13719 13723 le brave garçon
13896 13900 le modeste sac
15063 15067 le même compartiment
```

La première ligne indique dans quel mode de transduction la concordance a été calculée. Les 3 valeurs possibles sont:

- #I: les transductions ont été ignorées;
- #M: les transductions ont été insérées dans les séquences reconnues (mode MERGE);
- #R: les transductions ont remplacé les séquences reconnues (mode REPLACE).

Chaque occurrence est décrite par une ligne. Les lignes commencent par les positions de début et de fin de l'occurrence. Ces positions sont données en unités lexicales.

Si le fichier comporte la ligne d'en-tête #I, la position de fin de chaque occurrence est immédiatement suivie d'un retour à la ligne. Dans le cas contraire, elle est suivie d'un espace et d'une chaîne de caractères. En mode REPLACE, cette chaîne correspond à la transduction

produite pour la séquence reconnue. En mode MERGE, elle représente la séquence reconnue dans laquelle ont été insérées les transductions. En mode MERGE ou REPLACE, c'est cette chaîne qui est affichée dans la concordance. Si les transductions ont été ignorées, le contenu de l'occurrence est extrait du fichier texte.

10.6.2 Fichier concord.txt

Le fichier `concord.txt` est un fichier texte représentant une concordance. Chaque occurrence est codée par une ligne composée de 3 chaînes de caractères séparées par le caractère de tabulation et qui représentent le contexte gauche, l'occurrence (éventuellement modifiée par des transductions) et le contexte droit.

10.6.3 Fichier concord.html

Le fichier `concord.html` est un fichier `.html` qui représente une concordance. Ce fichier est codé en UTF-8.

Le titre de la page est le nombre d'occurrences qu'elle décrit. Les lignes de la concordance sont codées par des lignes où les occurrences sont considérées comme des liens hypertextes. La référence associée à chacun de ces liens est de la forme ``. X et Y représentent les positions de début et de fin de l'occurrence en caractères dans le fichier `nom_du_texte.snt`. Z représente le numéro de la phrase dans laquelle apparaît cette occurrence.

Tous les espaces sont codés comme des espaces insécables (` ` en HTML), ce qui permet de conserver l'alignement des occurrences même si l'une d'elles, se trouvant en début de fichier, a un contexte gauche complété avec des espaces.

NOTE: dans le cas d'une concordance construite avec le paramètre `glossanet`, le fichier HTML obtenu a la même structure, sauf en ce qui concerne les liens. Dans ces concordances, les occurrences sont des liens réels renvoyant vers le serveur web de l'application GlossaNet. Pour plus d'information sur GlossaNet, consulter les liens sur le site web d'Unitex (<http://www-igm.univ-mlv.fr/~unitex>).

Voici un exemple de fichier:

```
<html lang=en>¶
<head>¶
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">¶
  <title>6 matches</title>¶
</head>¶
<body>¶
<font face="Courier new" size=3>¶
MAÏTRE,&nbsp;L'<a href="104 109 2">AUTRE</a>&nbsp;COMM<br>¶
TRE&nbsp;COMME&nbsp;<a href="116 126 2">DOMESTIQUE</a><br>¶
-&nbsp;,&nbsp;Ëtait&nbsp;<a href="270 277 3">habitËle</a>&nbsp;pa<br>¶
'UN&nbsp;COMME&nbsp;<a href="94 100 2">MAÏTRE</a>,&nbsp;L'<br>¶
```

```

un de les;<a href="314 321 3">membres</a> le<br>
la maison <a href="158 165 3">portant</a> le<br>
</font>
</body>
</html>

```

La figure 10.2 montre la page correspondant au fichier ci-dessus.

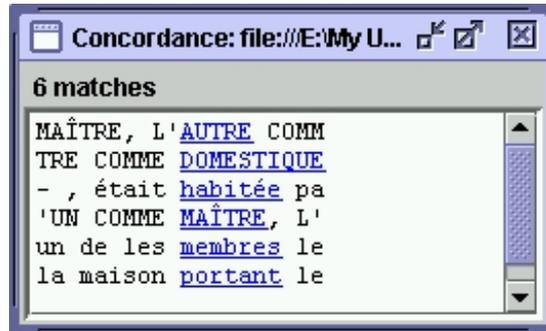


FIG. 10.2 – Exemple de concordance

10.7 Dictionnaires

La compression des dictionnaires DELAF par le programme **Compress** produit 2 fichiers: un fichier **.bin** qui représente l'automate minimal des formes fléchies du dictionnaire, et un fichier **.inf** qui contient les formes comprimées permettant de reconstruire les lignes du dictionnaire à partir des formes fléchies. Cette section décrit le format de ces deux types de fichiers, ainsi que le format du fichier **CHECK_DIC.TXT** qui contient le résultat de la vérification d'un dictionnaire.

10.7.1 Fichiers **.bin**

Un fichier **.bin** est un fichier binaire représentant un automate. Les 4 premiers octets du fichier représentent un entier indiquant la taille du fichier en octets. Les états de l'automate sont ensuite codés de la manière suivante:

- les 2 premiers octets indiquent si l'état est terminal ainsi que le nombre de transitions qui en sortent. Le bit le plus fort vaut 0 si l'état est terminal et 1 sinon. Les 15 autres bits codent le nombre de transitions.

Exemple: un état non terminal avec 17 transitions est codée par la séquence hexadécimale 8011

- si l'état est terminal, les 3 octets suivants codent l'indice dans le fichier **.inf** de la forme comprimée à utiliser pour reconstruire les lignes de dictionnaires pour cette forme fléchie.

Exemple: si l'état renvoie à la forme comprimée d'indice 25133, la séquence hexadécimale correspondante est 00622D

- chaque transition sortante est ensuite codée sur 5 octets. Les 2 premiers octets codent le caractère étiquetant la transition, et les 3 suivants codent la position en octets dans le fichier `.bin` de l'état d'arrivée. Les transitions d'un état sont codées les unes à la suite des autres.

Exemple: une transition étiquetée par le caractère `A` pointant vers l'état dont la description débute au 50106^{eme} octet sera représentée par la séquence hexadécimale 004100C3BA.

Par convention, le premier état de l'automate est l'état initial.

10.7.2 Fichiers `.inf`

Un fichier `.inf` est un fichier texte décrivant les formes comprimées associées à un fichier `.bin`. Voici un exemple de fichier `.inf`:

```
0000000006¶
_10\0\0\7.N¶
.PREP¶
_3.PREP¶
.PREP,_3.PREP¶
1-1.N+Hum:mp¶
3er 1.N+AN+Hum:fs¶
```

La première ligne du fichier indique le nombre de formes comprimées qu'il contient. Chaque ligne peut contenir une ou plusieurs formes comprimées. S'il y a plusieurs formes, celles-ci doivent être séparées par des virgules. Chaque forme comprimée est formée d'une séquence permettant de retrouver une forme canonique à partir d'une forme fléchie, suivie par la séquence de codes grammaticaux, sémantiques et flexionnels associés à l'entrée.

Le mode de compression de la forme canonique varie en fonction de la forme fléchie. Si les deux formes sont exactement identiques, la forme comprimée se résume aux informations grammaticales, sémantiques et flexionnelles, comme c'est le cas dans la ligne suivante:

```
.N+Hum:ms
```

Si les formes sont différentes, le programme de compression découpe les deux formes en unités. Ces unités peuvent être soit un espace, soit un tiret, soit une séquence de caractères ne contenant ni espace ni tiret. Ce mode de découpage permet de prendre efficacement en compte les flexions des mots composés.

Si les formes fléchie et canonique ne comportent pas le même nombre d'unités, le programme code la forme canonique par le nombre de caractères à retrancher de la forme fléchie, suivi des caractères à ajouter. Ainsi, la première ligne du fichier ci-dessus correspond à la ligne de dictionnaire:

`James Bond,007.N`

Comme la séquence `James Bond` contient trois unités et `007` seulement une, la forme canonique est codée par `_10\0\0\7`. Le caractère `_` indique que les deux formes n'ont pas le même nombre d'unités. Le nombre qui suit (ici 10) indique le nombre de caractères à retrancher. La séquence `\0\0\7` qui suit ce nombre indique que l'on doit ensuite ajouter la séquence `007`. Les chiffres sont précédés du caractère `\` pour ne pas être confondus avec le nombre de caractères à retrancher.

Lorsque les deux formes ont le même nombre d'unités, les unités sont comprimées deux à deux. Si les deux unités sont composées d'un espace ou d'un tiret, la forme comprimée de l'unité est l'unité elle-même, comme c'est le cas dans la ligne suivante:

`1-1.N+Hum:mp`

Cela permet de conserver une certaine visibilité dans le fichier `.inf` lorsque le dictionnaire contient des mots composés.

Lorsqu'au moins une des unités n'est ni un espace ni un tiret, la forme comprimée est composée du nombre de caractères à retrancher suivi de la séquence de caractères à ajouter. Ainsi, la ligne de dictionnaire:

`première partie,premier parti.N+AN+Hum:fs`

est codée par la ligne:

`3er 1.N+AN+Hum:fs`

Le code `3er` indique que l'on doit retrancher 3 caractères à la séquence `première` et lui ajouter les caractères `er` pour obtenir `premier`. Le 1 indique que l'on doit simplement retirer un caractère à `partie` pour obtenir la séquence `parti`. Le nombre 0 est utilisé lorsqu'on veut indiquer que l'on ne doit supprimer aucun caractère.

10.7.3 Fichier CHECK_DIC.TXT

Ce fichier est produit par le programme de vérification de dictionnaire `CheckDic`. Il s'agit d'un fichier texte qui donne des informations sur le dictionnaire analysé, et se décompose en 4 parties.

La première partie donne la liste, éventuellement vide, de toutes les erreurs de syntaxe trouvées dans le dictionnaire: absence de la forme fléchie ou de la forme canonique, absence de code grammatical, ligne vide, etc. Chaque erreur est décrite par le numéro de la ligne concernée, un message décrivant la nature de l'erreur, ainsi que le contenu de la ligne. Voici un exemple de message:

`Line 12451: no point found
jardin,N:ms`

Les deuxième et troisième parties donnent respectivement les listes de codes grammaticaux et/ou sémantiques et flexionnels. Afin de prévenir des erreurs de codage, le programme signale les codes qui contiennent des espaces, des tabulations ou des caractères non ASCII. Ainsi, si un dictionnaire grec contient le code ADV où le caractère A est le A grec au lieu du A latin, le programme signalera l'avertissement suivant:

```
ADV warning: 1 suspect char (1 non ASCII char): (0391 D V)
```

Les caractères non ASCII sont indiqués par leur numéro de caractère en hexadécimal. Dans l'exemple ci-dessus, le code 0391 représente le A grec. Les espaces sont indiqués par la séquence SPACE:

```
Km s warning: 1 suspect char (1 space): (K m SPACE s)
```

Lorsqu'on vérifie le dictionnaire suivant:

```
1,2 et 3!,.INTJ
abracadrabra,INTJ
saperlipopette,.INTJ
zut,.INTJ
```

on obtient le fichier CHECK_DIC.TXT suivant:

```
Line 1: unprotected comma in lemma
1,2 et 3!,.INTJ
Line 2: no point found
ah,INTJ
-----
---- All chars used in forms ----
-----
(0020)
! (0021)
, (002C)
1 (0031)
2 (0032)
3 (0033)
I (0049)
J (004A)
N (004E)
T (0054)
a (0061)
e (0065)
h (0068)
p (0070)
r (0072)
s (0073)
```

```

t (0074)¶
u (0075)¶
z (007A)¶
-----¶
----      2 grammatical/semantic codes used in dictionary  ----¶
-----¶
INTJ¶
  INTJ warning: 1 suspect char (1 space): (SPACE I N T J)¶
-----¶
----      0 inflectional code used in dictionary  ----¶
-----¶

```

10.8 Fichiers d'ELAG

10.8.1 Fichier tagset.def

Voir section 7.3.6, page 107.

10.8.2 Fichiers .lst

LES FICHIERS .LST NE SONT PAS CODÉS EN UNICODE.

Un fichier `.lst` contient une liste de noms de fichiers `.grf`, localisés par rapport au répertoire ELAG de la langue courante. Voici le fichier `elag.lst` fourni pour le français:

```

PPVs/PpvIL.grf¶
PPVs/PpvLE.grf¶
PPVs/PpvLUI.grf¶
PPVs/PpvPR.grf¶
PPVs/PpvSeq.grf¶
PPVs/SE.grf¶
PPVs/postpos.grf¶

```

10.8.3 Fichiers .elg

Les fichiers `.elg` contiennent des règles ELAG compilées. Ces fichiers sont au format `.fst2`.

10.8.4 Fichiers .rul

LES FICHIERS .RUL NE SONT PAS CODÉS EN UNICODE.

Ces fichiers listent les différents fichiers `.elg` qui compose un ensemble de règles ELAG. Un fichier `.rul` est constitué d'autant de parties qu'il y a de fichiers `.elg`. Chaque partie est composée de la liste des grammaires ELAG qui correspondent à un fichier `.elg`, où chaque nom de fichier est précédé par une tabulation, suivi par une ligne contenant le nom du fichier

.elg entre angles. Les lignes commençant par une tabulation ont valeur de commentaire et sont ignorées par le programme Elag. Voici le fichier `elag.rul` fourni par défaut pour le français:

```

    PPVs/PpvIL.elg¶
    PPVs/PpvLE.elg¶
    PPVs/PpvLUI.elg¶
<elag.rul-0.elg>¶
    PPVs/PpvPR.elg¶
    PPVs/PpvSeq.elg¶
    PPVs/SE.elg¶
    PPVs/postpos.elg¶
<elag.rul-1.elg>¶

```

10.9 Fichiers de configuration

10.9.1 Fichier Config

Lorsque l'utilisateur modifie ses préférences pour une langue donnée, celles-ci sont sauvegardées dans un fichier texte nommé `Config` qui se trouve dans le répertoire de la langue courante. Ce fichier a la syntaxe suivante:

```

TEXT FONT NAME=Courier new¶
TEXT FONT STYLE=0¶
TEXT FONT SIZE=11¶
CONCORDANCE FONT NAME=Courier new¶
CONCORDANCE FONT HTML SIZE=3¶
INPUT FONT NAME=Times New Roman¶
INPUT FONT STYLE=0¶
INPUT FONT SIZE=10¶
OUTPUT FONT NAME=Times New Roman¶
OUTPUT FONT STYLE=1¶
OUTPUT FONT SIZE=12¶
DATE=true¶
FILE NAME=true¶
PATH NAME=false¶
FRAME=true¶
RIGHT TO LEFT=false¶
BACKGROUND COLOR=16777215¶
FOREGROUND COLOR=0¶
AUXILIARY NODES COLOR=13487565¶
COMMENT NODES COLOR=16711680¶
SELECTED NODES COLOR=255¶
ANTIALIASING=false¶
HTML VIEWER=¶

```

Les trois premières lignes indiquent le nom, le style et la taille de la police utilisée pour afficher les textes, les dictionnaires, les unités lexicales, les phrases de l'automate du texte, etc.

Les paramètres `CONCORDANCE FONT NAME` et `CONCORDANCE FONT HTML SIZE` définissent le nom et la taille de la police à utiliser pour afficher les concordances en HTML. La taille de la police doit être comprise entre 1 et 7.

Les paramètres `INPUT FONT ...` et `OUTPUT FONT ...` définissent le nom, le style et la taille des polices utilisées pour afficher les chemins et les transductions des graphes.

Les 10 paramètres suivants correspondent aux paramètres précisés dans les en-têtes des graphes. Le tableau 10.3 décrit ces correspondances.

Paramètres dans le fichier <code>Config</code>	Paramètres dans un fichier <code>.grf</code>
<code>DATE</code>	<code>DDATE</code>
<code>FILE NAME</code>	<code>DFILE</code>
<code>PATH NAME</code>	<code>DDIR</code>
<code>FRAME</code>	<code>DFRAME</code>
<code>RIGHT TO LEFT</code>	<code>DRIG</code>
<code>BACKGROUND COLOR</code>	<code>BCOLOR</code>
<code>FOREGROUND COLOR</code>	<code>FCOLOR</code>
<code>AUXILIARY NODES COLOR</code>	<code>ACOLOR</code>
<code>COMMENT NODES COLOR</code>	<code>SCOLOR</code>
<code>SELECTED NODES COLOR</code>	<code>CCOLOR</code>

TAB. 10.3 – *Signification des paramètres*

Le paramètre `ANTIALIASING` indique si les graphes ainsi que les automates de phrases doivent être affichés par défaut avec l'effet d'antialiasing.

Le paramètre `HTML VIEWER` indique le nom du navigateur à utiliser pour afficher les concordances. Si aucun nom de navigateur n'est précisé, les concordances sont affichées dans une fenêtre d'Unitex.

10.9.2 Fichier `system_dic.def`

Le fichier `system_dic.def` est un fichier texte décrivant la liste des dictionnaires du système à appliquer par défaut. Ce fichier se trouve dans le répertoire de la langue courante. Chaque ligne correspond à un nom de fichier `.bin`. Les dictionnaires du système doivent trouver dans le répertoire du système, à l'intérieur du sous-répertoire (`langue courante`)/`Dela`. Voici un exemple de fichier:

```
delacf.bin¶
delaf.bin¶
```

10.9.3 Fichier `user_dic.def`

Le fichier `user_dic.def` est un fichier texte décrivant la liste des dictionnaires de l'utilisateur à appliquer par défaut. Ce fichier se trouve dans le répertoire de la langue courante et a le même format que le fichier `system_dic.def`. Les dictionnaires de l'utilisateur doivent se trouver dans le sous-répertoire (`langue courante`)/`Dela` du répertoire personnel de l'utilisateur.

10.9.4 Fichier `user.cfg`

Sous Linux, Unitex considère que le répertoire personnel de l'utilisateur se nomme `unitex` et qu'il se trouve dans son répertoire racine (`$HOME`). Sous Windows, il n'est pas toujours possible d'associer un répertoire par défaut à un utilisateur. Pour remédier à cela, Unitex crée pour chaque utilisateur un fichier `.cfg` contenant le chemin de son répertoire personnel. Ce fichier est sauvegardé sous le nom (`login de l'utilisateur`).`cfg` dans le sous-répertoire du système `Unitex/Users`.

ATTENTION: CE FICHIER N'EST PAS EN UNICODE ET LE CHEMIN DU RÉPERTOIRE PERSONNEL N'EST PAS SUIVI PAR UN RETOUR À LA LIGNE.

10.10 Fichiers divers

Pour chaque texte, Unitex crée plusieurs fichiers contenant des informations destinées à être affichée dans l'interface graphique. Cette section décrit ces différents fichiers.

10.10.1 Fichiers `dlf.n`, `dlc.n` et `err.n`

Ces trois fichiers sont des fichiers texte se trouvant dans le répertoire du texte. Ils contiennent respectivement les nombres de lignes des fichiers `dlf`, `dlc` et `err`. Ces nombres sont suivis par un retour à la ligne.

10.10.2 Fichier `stat_dic.n`

Ce fichier est un fichier texte se trouvant dans le répertoire du texte. Il est formé de trois lignes, contenant les nombres de lignes des fichiers `dlf`, `dlc` et `err`.

10.10.3 Fichier `stats.n`

Ce fichier texte se trouve dans le répertoire du texte et contient une ligne de la forme suivante:

```
3949 sentence delimiters, 169394 (9428 diff) tokens, 73788 (9399) simple
forms, 438 (10) digits¶
```

Les nombres indiqués s'interprètent de la façon suivante:

– `sentence delimiters`: nombre de séparateurs de phrases (`{S}`);

- **tokens**: nombre total d'unités lexicales du texte. Le nombre précédant **diff** indique le nombre d'unités différentes;
- **simple forms**: nombre total dans le texte d'unités lexicales composées de lettres. Le nombre entre parenthèses représente le nombre d'unités lexicales différentes qui sont composées de lettres;
- **digits**: nombre total dans le texte de chiffres. Le nombre entre parenthèses indique le nombre de chiffres différents utilisés (au plus 10).

10.10.4 Fichier `concord.n`

Le fichier `concord.n` est un fichier texte qui se trouve dans le répertoire du texte. Il contient des informations sur la dernière recherche de motifs effectuée sur ce texte et se présente de la manière suivante:

```
6 matches¶  
6 recognized units¶  
(0.004% of the text is covered)¶
```

La première ligne donne le nombre d'occurrences trouvées, la seconde le nombre d'unités couvertes par ces occurrences. La troisième ligne indique le rapport entre le nombre d'unités couvertes et le nombre total d'unités du texte.

Annexe A - GNU General Public License

Voir [21] pour l'original de ce document.

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.

59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
 - (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
 - (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
 - (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice

and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
 - (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the

same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice. This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries

not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) yyyy name of author
```

```
This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2 of the License, or (at your option) any later version.
This program is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public
License for more details.
```

```
You should have received a copy of the GNU General Public License along with
this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place
- Suite 330, Boston, MA 02111-1307, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) yyyy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show
w'.
This is free software, and you are welcome to redistribute it under certain condi-
tions; type 'show c' for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.
```

```
signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Annexe B - GNU Lesser General Public License

Voir [22] pour l'original de ce document.

GNU LESSER GENERAL PUBLIC LICENSE Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages—typically libraries—of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive

or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be

combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) The modified work must itself be a software library.
- b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
- d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the

facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would

be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE

DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.> Copyright (C)
<year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990 Ty Coon, President of Vice

That's all there is to it!

Bibliographie

- [1] Anna ANASTASSIADIS-SYMEONIDIS, Tita KYRIACOPOULOU, Elsa SKLAVOUNOU, Iasson THILIKOS, and Rania VOSKAKI. A system for analysing texts in modern greek: representing and solving ambiguities. In *Proceedings of COMLEX 2000, Workshop on Computational Lexicography and Multimedia Dictionaries*. Patras, 2000.
- [2] Olivier BLANC and Anne DISTER. Automates lexicaux avec structure de traits. 2004. Actes RECITAL 2004.
- [3] Xavier BLANCO. Noms composés et traduction français-espagnol. *Linguisticæ Investigationes*, 21(1), 1997. Amsterdam-Philadelphia: John Benjamins Publishing Company.
- [4] Xavier BLANCO. Les dictionnaires électroniques de l'espagnol (DELASs et DELACs). *Linguisticæ Investigationes*, 23(2), 2000. Amsterdam-Philadelphia: John Benjamins Publishing Company.
- [5] Jean-Paul BOONS, Alain GUILLET, and Christian LECLÈRE. La structure des phrases simples en français: classes de constructions transitives. Technical report, LADL, Paris, 1976.
- [6] Jean-Paul BOONS, Alain GUILLET, and Christian LECLÈRE. *La structure des phrases simples en français: constructions intransitives*. Droz, Genève, 1976.
- [7] Mozilla. Web browser. <http://www.mozilla.org>.
- [8] Netscape. Web browser. <http://www.netscape.com>.
- [9] Folker CAROLI. Les verbes transitifs à complément de lieu en allemand. *Linguisticæ Investigationes*, 8(2):225–267, 1984. Amsterdam-Philadelphia: John Benjamins Publishing Company.
- [10] A. CHROBOT, B. COURTOIS, M. HAMMANI-Mc CARTHY, M. GROSS, and K. ZELLAGUI. Dictionnaire électronique DELAC anglais: noms composés. Technical Report 59, LADL, Université Paris 7, 1999.
- [11] Matthieu CONSTANT and Anastasia YANNAKOPOULOU. Le dictionnaire électronique du grec moderne: Conception et développement d'outils pour son enrichissement et sa validation. In *Studies in Greek Linguistics, Proceedings of the 23rd annual meeting of the Department of Linguistics*. Faculty of Philosophy, Aristotle University of Thessaloniki, 2002.
- [12] Blandine COURTOIS. Formes ambiguës de la langue française. *Linguisticæ Investigationes*, 20(1):167–202, 1996. Amsterdam-Philadelphia: John Benjamins Publishing Company.
- [13] Blandine Courtois and Max Silberztein, editors. *Les dictionnaires électroniques du français*. Larousse, Langue française, vol. 87, 1990.

- [14] Anne DISTER, Nathalie FRIBURGER, and Denis MAUREL. Améliorer le découpage en phrases sous INTEX. In Anne Dister, editor, *Revue Informatique et Statistique dans les Sciences Humaines*, volume Actes des 3èmes Journées INTEX, pages 181–199, 2000.
- [15] Anibale ELIA. *Le verbe italien. Les complétives dans les phrases à un complément*. Schena/Nizet, Fasano/Paris, 1984.
- [16] Anibale ELIA. *Lessico-grammatica dei verbi italiani a completiva. Tavole e indice generale*. Liguori, Napoli, 1984.
- [17] Anibale ELIA and Simoneta VIETRI. Electronic dictionaries and linguistic analysis of italian large corpora. In *Actes des 5es Journées internationales d'Analyse statistique des Données Textuelles*. Ecole Polytechnique fédérale de Lausanne, 2000.
- [18] Anibale ELIA and Simoneta VIETRI. L'analisi automatica dei testi e i dizionari elettronici. In E. Burattini and R. Cordeschi, editors, *Manuale di Intelligenza Artificiale per le Scienze Umane*. Roma:Carocci, 2002.
- [19] Jacqueline GIRY-SCHNEIDER. *Les nominalisations en français. L'opérateur faire dans le lexique*. Droz, Genève-Paris, 1978.
- [20] Jacqueline GIRY-SCHNEIDER. *Les prédicats nominaux en français. Les phrases simples à verbe support*. Droz, Genève-Paris, 1987.
- [21] GNU. General Public License. <http://www.gnu.org/licenses/gpl.html>.
- [22] GNU. Lesser General Public License. <http://www.gnu.org/licenses/lgpl.html>.
- [23] Gaston GROSS. *Les expressions figées en français*. Ophrys, Paris, 1996.
- [24] Maurice GROSS. *Méthodes en syntaxe*. Hermann, Paris, 1975.
- [25] Maurice GROSS. *Grammaire transformationnelle du français. 3 - Syntaxe de l'adverbe*. ASSTRIL, Paris, 1986.
- [26] Alain GUILLET and Christian LECLÈRE. *La structure des phrases simples en français: les constructions transitives locatives*. Droz, Genève, 1992.
- [27] Gaby KLARSFLED and Mary HAMMANI-MC CARTHY. Dictionnaire électronique du *ladl* pour les mots simples de l'anglais (DELA_{Sa}). Technical report, LADL, Université Paris 7, 1991.
- [28] Tita KYRIACOPOULOU. *Les dictionnaires électroniques: la flexion verbale en grec moderne*, 1990. Thèse de doctorat. Université Paris 8.
- [29] Tita KYRIACOPOULOU. Un système d'analyse de textes en grec moderne: représentation des noms composés. In *Actes du 5ème Colloque International de Linguistique Grecque, 13-15 septembre 2001*. Sorbonne, Paris, 2002.
- [30] Tita KYRIACOPOULOU, Safia MRABTI, and Anastasia YANNAKOPOULOU. Le dictionnaire électronique des noms composés en grec moderne. *Linguisticæ Investigationes*, 25(1):7–28, 2002. Amsterdam-Philadelphia: John Benjamins Publishing Company.
- [31] Jacques LABELLE. Le traitement automatique des variantes linguistiques en français: l'exemple des concrets. *Linguisticæ Investigationes*, 19(1):137–152, 1995. Amsterdam-Philadelphia: John Benjamins Publishing Company.
- [32] Eric LAPORTE. Noms appropriés à modifieurs obligatoires. *Langages*, 126, 1997. Paris: Larousse.
- [33] Eric LAPORTE and Anne MONCEAUX. Elimination of lexical ambiguities by grammars: The ELAG system. *Linguisticæ Investigationes*, 22:341–367, 1998.

- [34] Ville LAURIKARI. TRE home page. <http://laurikari.net/tre/>.
- [35] Annie MEUNIER. *Nominalisation d'adjectifs par verbes supports*, 1981. Thèse de doctorat. Université Paris 7.
- [36] Sun Microsystems. Java. <http://java.sun.com>.
- [37] Christian MOLINIER and Françoise LEVRIER. *Grammaire des adverbes: description des formes en -ment*. Droz, Genève, 2000.
- [38] Anne MONCEAUX. Le dictionnaire des mots simples anglais : mots nouveaux et variantes orthographiques. Technical Report 15, IGM, Université de Marne-la-Vallée, 1995.
- [39] Dong-Ho PAK. *Lexique-grammaire comparé français-coréen. Syntaxe des constructions complétives*. PhD thesis, UQAM, Montréal, 1996.
- [40] Soung-Nam PARK. *La construction des verbes neutres en coréen*, 1996. Thèse de doctorat. Université Paris 7.
- [41] Sébastien PAUMIER and Harald ULLAND. Analyse automatique de mots polylexicaux en norvégien. Amsterdam-Philadelphia: John Benjamins Publishing Company (to appear 2004).
- [42] Roger-Bruno RABENILAINA. *Le verbe malgache*. AUPELF-UREF et Université Paris 13, Paris, 1991.
- [43] Agata SAVARY. *Recensement et description des mots composés - méthodes et applications*, 2000. Thèse de doctorat. Université de Marne-la-Vallée.
- [44] Max SILBERZTEIN. Les groupes nominaux productifs et les noms composés lexicalisés. *Linguisticae Investigationes*, 27(2):405–426, 1999. Amsterdam-Philadelphia: John Benjamins Publishing Company.
- [45] Carlos SUBIRATS-RÜGGERBERG. *Sentential complementation in Spanish. A lexicogrammatical study of three classes of verbs*. John Benjamins, Amsterdam/Philadelphia, 1987.
- [46] OpenOffice.org: suite bureautique. <http://www.openoffice.org>.
- [47] Thomas TREIG. Complétives en allemand. classification. Technical Report 7, LADL, 1977.
- [48] Consortium Unicode. <http://www.unicode.org>.
- [49] Lidia VARGA. Classification syntaxique des verbes de mouvement en hongrois dans l'optique d'un traitement automatique. In F. Kiefer, G. Kiss, and J. Pajzs, editors, *Papers in Computational Lexicography (COMPLEX)*, pages 257–265, Budapest, Research Institute for Linguistics, Hungarian Academy of Sciences, 1996.
- [50] Simoneta VIETRI. On the study of idioms in italian. In *Sintassi e morfologia della lingua italiana, Congresso internazionale della Società di Linguistica Italiana*. Roma: Bulzoni, 1984.

Index

+, 30, 41, 48, 58
Elag, 155
-, 110
cat, 109
complete, 109
discr, 109
inflex, 109
t, 18
!, 46
#, 21, 44, 46, 76
\$, 61, 62
*, 48
,, 30, 32
-, 41, 45
., 30, 47
/, 30, 61
1, 34
2, 34
3, 34
:, 30, 59
<CDIC>, 44
<DIC>, 44, 46
<E>, 21, 44, 46, 48, 58, 74, 76
<MAJ>, 21, 44, 46
<MIN>, 21, 44, 46
<MOT>, 21, 44
<NB>, 21, 44, 46
<PNC>, 21
<PRE>, 21, 44, 46
<SDIC>, 44
<^>, 21, 74
=, 31
@%, 119
@, 119
A, 33
ADV, 33
Abst, 33
An1, 33
An1Coll, 33
C, 34
CONJC, 33
CONJS, 33
CheckDic, 34, 125, 152
Compress, 31, 39, 125, 150
Conc, 33
ConcColl, 33
Concord, 126
Convert, 127
DET, 33
Dico, 26, 42, 128
Elag, 129
ElagComp, 129
Evamb, 130
ExploseFst2, 130
Extract, 130
F, 34
Flatten, 77, 130
Fst2Grf, 114, 131, 147
Fst2List, 131
Fst2Txt, 23, 132
G, 34
Grf2Fst2, 77, 133
Hum, 33
HumColl, 33
I, 34
INTJ, 33
ImploseFst2, 133
Inflect, 39, 133
J, 34
K, 34
L, 38, 73
Locate, 133, 148
MergeTextAutomaton, 134
N, 33

- Normalize, 125, 134
- P, 34
- PREP, 33
- PRO, 33
- PolyLex, 28, 135
- R, 38, 73
- Reconstrucao, 97, 135
- Reg2Grf, 135
- S, 34
- SortTxt, 36, 136, 141
- T, 34
- Table2Grf, 136
- TextAutomaton2Mft, 136
- Tokenize, 24, 136
- Txt2Fst2, 137
- V, 33
- W, 34
- Y, 34
- \, 30, 43
- \,, 30
- \., 30
- \=, 31
- _, 62
- en, 33
- f, 34
- i, 33
- m, 34
- n, 34
- ne, 33
- p, 34
- s, 34
- se, 33
- t, 33
- z1, 33
- z2, 33
- z3, 33
- {S}, 21, 47, 134, 137, 145, 146, 157

- Ajout de nouvelles langues, 12
- Alignement des boîtes, 67
- All matches, 50, 89, 133
- Alphabet, 22, 127, 132, 133, 136, 137, 140
 - de tri, 36, 141
- Analyse des mots composés libre en allemand, 135
- Analyse des mots composés libre en norvégien, 135
- Analyse des mots composés libres, 28
- Antialiasing, 66, 69, 156
- Approximation d'une grammaire par un transducteur à états finis, 77, 130
- Arrobas, 119
- Automate
 - acyclique, 93
 - du texte, 44, 75, 93, 131, 134, 136, 137
 - forme compacte, 130, 133
 - forme développée, 130
 - minimal, 39
 - à états finis, 56
- Axiome, 55

- Barre d'icônes, 64
- Boucles infinies, 78
- Boîtes
 - alignement, 67
 - création, 57
 - relier des, 58
 - suppression, 60
 - sélection, 60
 - tri des lignes, 65

- Clitiques
 - normalisation, 97, 135
- Codes flexionnels, 110
- Collection de graphes, 83
- Coller, 60, 62, 64
- Commentaire
 - dans un dictionnaire, 30
 - dans un graphe, 58
- Compilation
 - des grammaires ELAG, 102
- Compilation d'un graphe, 76, 133
- Compression de dictionnaires, 39, 125, 135
- Concaténation d'expressions rationnelles, 43, 47
- Concordance, 51, 89, 126
- Conservation des meilleurs chemins, 97, 137
- Contextes
 - concordance, 52, 89, 126
 - copie de liste, 63

- Contextes des occurrences, 51
- Contraintes flexionnelles, 45
- Contraintes sur les grammaires, 78
- Conversion de fichiers, 15
- Copie de listes, 62
- Copier, 60, 62, 64
- Corpus, voir Texte
- Couleurs
 - configuration des, 67
- Couper, 64
- Création d'une boîte, 57

- Degré d'ambiguïté, 94
- DELA, 20, 29
- DELAC, 29
- DELACF, 29
- DELAF, 29–32, 150
- DELAS, 29, 32
- Diagrammes de syntaxe, 56
- Dictionnaires
 - application de, 25, 41, 128
 - codes utilisés dans les, 32
 - commentaires dans les, 30
 - compression, 39, 125, 135
 - contenu des, 32
 - DELAC, 29
 - DELACF, 29
 - DELAF, 29–32, 125, 133, 150
 - DELAS, 29, 32, 133
 - du texte, 26, 44, 93
 - filtres, 41
 - finesse, 94
 - flexion automatique, 37, 133
 - format, 29
 - priorités, 41
 - référence aux, 44, 76
 - sélection par défaut, 27
 - tri, 36
 - vérification, 34, 125
- Découpage en phrases, 21
- Déplacer des groupes de mots, 86
- Dérivation, 55
- Détection d'erreurs dans les graphes, 81, 133

- Editeur de texte intégré, 17

- ELAG, 100
- Ensembles de grammaires, 105
- Entrées lexicales, 29
- Entrées lexicales factorisées, 104
- Epsilon, voir <E>
- Equivalence de caractères, 36
- Erreurs dans les graphes, 81, 133
- Espace
 - interdit, 44
 - obligatoire, 44
- Etat
 - final, 57
 - initial, 57
- Etiquettes lexicales, 44, 95, 134, 137, 145, 146
- Etoile de Kleene, 43, 48
- Evaluation du taux d'ambiguïté, 107
- Exclusion de codes grammaticaux et sémantiques, 45
- Exploration des chemins d'une grammaire, 81
- Expression rationnelle, 43, 56, 135
- Expressions régulières, 48
- Extraire les occurrences, 51, 91

- Fenêtre de concordance, 52
- Fenêtre de processing d'ELAG, 106
- Fichier
 - `-conc.fst2`, 102
 - `.fst2`, 129
 - `.lst`, 105, 106
 - `.rul`, 102, 106, 129
 - `tagset.def`, 107, 110–112
 - `.bin`, 39, 125, 128, 150, 156
 - `.cfg`, 157
 - `.dic`, 35, 39, 125
 - `.elg`, 154
 - `.fst2`, 51, 77, 114, 133, 144
 - `.grf`, 51, 81, 114, 133, 135, 141
 - `.html`, 127
 - `.ind`, 130
 - `.inf`, 39, 125, 151
 - `.lst`, 154
 - `.rul`, 154
 - `.snt`, 21, 130, 134, 136, 137, 139, 146

- .txt, 90, 127, 139, 145
- Alphabet.txt, 140
- Alphabet_sort.txt, 36
- CHECK_DIC.TXT, 34, 125, 152
- Config, 155
- Replace.fst2, 23
- Sentence.fst2, 22
- Unitex.jar, 12, 13
- Unitex_1.2.zip, 12
- alphabet, 42
- concord.html, 149
- concord.ind, 134, 148
- concord.n, 134, 158
- concord.txt, 149
- cursor.sentence.grf, 131, 147
- cursor.sentence.txt, 131, 148
- dlc, 26, 37, 129, 157
- dlc.n, 157
- dlf, 26, 37, 129, 157
- dlf.n, 157
- enter.pos, 137, 146
- err, 26, 37, 129, 157
- err.n, 157
- regexp.grf, 135
- stat_dic.n, 129, 157
- stats.n, 25, 137, 157
- system_dic.def, 156
- tagset.def, 154
- text.cod, 25, 137, 146
- text.fst2, 131, 137, 146
- text.fst2.bck, 134
- tok_by_alpha.txt, 25, 137, 146
- tok_by_freq.txt, 25, 137, 146
- tokens.txt, 24, 137, 146
- user_dic.def, 157
- alphabet, 15, 22, 24, 34, 127, 132, 133, 136, 137
- formats de, 139
- HTML, 52, 90, 126
- texte, 18, 139
 - taille maximum, 19
- Filtres morphologiques, 48
- Finesse des dictionnaires, 94
- Flexion automatique, 37, 73, 133
- Formats de fichiers, 139
- Forme
 - canonique, 29
 - fléchie, 29
- GlossaNet, 127, 149
- GPL, 11, 159
- Grammaires
 - algébriques, 55
 - algébriques étendues, 56
 - contraintes, 78
 - de découpage en phrases, 21, 74
 - de flexion, 37
 - de levée d'ambiguïtés, 100
 - de normalisation
 - de formes non ambiguës, 23, 74
 - de l'automate du texte, 75
 - ELAG, 76
 - ensembles de, 105
 - formalisme, 55
 - hors-contexte, 55
 - locales, 76
- Graphe
 - alignement des boîtes, 67
 - antialiasing, 66, 69
 - appel à un sous-graphe, 59
 - approximation par un transducteur à états finis, 77, 130
 - commentaire dans un, 58
 - compilation, 76, 133
 - création d'une boîte, 57
 - de flexion, 73
 - détection d'erreurs, 81, 133
 - format, 141
 - impression, 71
 - inclusion dans un document, 70
 - Intex, 56
 - paramétré, 76, 118
 - patron, 136
 - présentation, 65
 - présentation, polices et couleurs, 67
 - relier des boîtes, 58
 - sauvegarde, 59
 - suppression de boîtes, 60
 - syntactique, 76
 - types de, 73

- variables dans un, 61
- zoom, 65
- Grille, 67
- Importer un graphe Intex, 56
- Imprimer
 - un automate de phrase, 116
 - un graphe, 71
- Inclure un graphe dans un document, 70
- Informations
 - flexionnelles, 30
 - grammaticales, 30
 - sémantiques, 30
- Installation
 - sous Linux et MacOS, 12
 - sous Windows, 12
- Java Runtime Environment, 11
- Jeu d'étiquettes ELAG, 107
- JRE, 11
- Kleene, voir Etoile de Kleene
- LADL, 9, 29, 117
- Langages algébriques, 56
- Levée d'ambiguïtés, 102
- Levée d'ambiguïtés lexicales, 100
- Lexique-grammaire, 117
- LGPL, 11, 165
- Licence
 - GPL, 11, 159
 - LGPL, 11, 165
- Limiter les branches *alors*, 113
- Longest matches, 50, 89, 133
- Machine virtuelle Java, 11
- Matrices, 117
- MERGE, 23, 84, 89, 132, 134, 148
- Modification du texte, 90, 126
- Modifier le texte, 51
- Motif, 43, 44
- Mots
 - composés, 25, 44
 - avec espace ou tiret, 31
 - libres, 28
 - libres en allemand, 135
 - libres en norvégien, 135
 - inconnus, 26, 46
 - simples, 25, 44
- Métas, 21, 44, 64
- Navigateur web, 52, 90
- Nom de variable, 62
- Normalisation
 - de formes ambiguës, 75, 95, 137
 - de formes non ambiguës, 23
 - de l'automate du texte, 75, 95, 137
 - des clitiques en portugais, 97, 135
 - des séparateurs, 21, 134
- Norvégien
 - mots composés libres en, 135
- Négation, 46
- Occurrences
 - extraction, 91
 - nombre d', 51, 89, 134
- Optimisation des grammaires ELAG, 112
- Opérateur
 - L, 38, 73
 - R, 38, 73
 - de concaténation, 47
 - de disjonction, 48
 - de Kleene, 48
- Parenthèses, 47
- Pixellisation, 66
- Point de synchronisation, 101
- Polices
 - configuration des, 67
- Portugais
 - normalisation des clitiques, 97, 135
- POSIX, 48
- Priorité
 - aux séquences les plus longues, 86
 - des dictionnaires, 41
 - à gauche, 85
- Programme externe
 - Elag, 102, 106, 107, 155
 - ElagComp, 107, 112
- Programmes externes
 - ElagComp, 102
 - CheckDic, 34, 125, 152

- Compress, 31, 39, 125, 150
- Concord, 126
- Convert, 127
- Dico, 26, 42, 128
- Elag, 129
- ElagComp, 129
- Evamb, 130
- ExploseFst2, 130
- Extract, 130
- Flatten, 77, 130
- Fst2Grf, 114, 131, 147
- Fst2List, 131
- Fst2Txt, 23, 132
- Grf2Fst2, 77, 133
- ImplouseFst2, 133
- Inflect, 39, 133
- Locate, 133, 148
- MergeTextAutomaton, 134
- Normalize, 125, 134
- PolyLex, 28, 135
- Reconstrucao, 97, 135
- Reg2Grf, 135
- SortTxt, 36, 136, 141
- Table2Grf, 136
- TextAutomaton2Mft, 136
- Tokenize, 24, 136
- Txt2Fst2, 137
- Propriétés syntaxiques, 117
- Préférences, 69
- Recherche de motifs, 50, 88, 133
- Reconstruction de l'automate du texte, 134
- REPLACE, 23, 84, 89, 132, 134, 148
- Respect
 - des espaces, 42, 76
 - des minuscules/majuscules, 41, 43, 74–76
- Respect de la casse, 50
- Ressources lexicales, voir Dictionnaires
- RTN, 56
- Règles
 - d'application des transducteur, 84
 - de réécriture, 55
- Référence aux dictionnaires, 44, 76
- Répertoire
 - du texte, 21, 125
 - personnel, 12
- Réseaux de transitions récursifs, 56
- Shortest matches, 50, 89, 133
- Sortie, 61, 68
 - associée à un sous-graphe, 78
 - à variables, 61, 86
- Symboles
 - non-terminaux, 55
 - spéciaux, 63
 - terminaux, 55
- Symboles lexicaux, 113
- Sélection de la langue, 15
- Sélection multiple, 60
 - copier-coller, 60
- Séparateurs, 21
 - de phrases, 21, 47, 134, 137, 145, 146, 157
- Tables de lexique-grammaire, 117, 136
- Taille maximum des fichiers textes, 19
- Taux d'ambiguïté, 107
- Text
 - répertoire du, 125
- Texte
 - automate du, 44, 93, 131, 134, 136, 137
 - découpage en phrases, 21
 - découpage en unités lexicales, 23, 136
 - format, 15
 - modification, 90, 126
 - normalisation, 21, 134
 - normalisation de l'automate du, 75, 95
 - prétraitement, 20, 74
 - répertoire du, 21
- Tokens, voir Unités lexicales
- Transducteur, 56
 - règles d'application, 84
- Transduction, 56
- Tri, 136
 - d'un dictionnaire, 36
 - des concordances, 52, 89, 126
 - des lignes d'une boîte, 65
- Types de graphes, 73
- Underscore, 62, 86

- Unicode, 15, 56, 65, 127, 139
- Union d'expressions rationnelles, 43, 48
- Unités lexicales, 43, 137
 - découpage en, 23, 136
- UTF-8, 127, 128, 149

- Variables
 - dans les graphes, 61, 86
 - dans les graphes paramétrés, 119
- Vérification du format d'un dictionnaire,
 - 34, 125

- Zoom, 65