UNITEX 3.2

MANUEL D'UTILISATION

Université Paris-Est Marne-la-Vallée

http://unitexgramlab.org
unitex-devel@univ-mlv.fr

Sébastien Paumier

avec la participation de Wolfgang Flury, Franz Guenthner, Eric Laporte, Friederike Malchok, Clemens Marschner, Claude Martineau, Cristian Martínez, Denis Maurel, Sebastian Nagel, Alexis Neme, Maxime Petit, Johannes Stiehler, Gilles Vollant

Date de cette version : 14 janvier 2020

Table des matières

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.

Quoi de neuf depuis la version 3.1 ?

Voici les principales nouvelles fonctionnalités :

- Unitex/GramLab inclut maintenant des données expérimentales sur le chinois.
- On peut ouvrir un sous-graphe — ou le créer s'il n'existe pas — en faisant un clic droit sur un appel à ce sous-graphe.
- Le développement fait appel à l'intégration continue ; le code source, les ressources linguistiques et les versions précédentes sont disponibles sur des dépôts de données publics :
  `https://github.com/UnitexGramLab`
- CasSys et les graphes de généralisation d'étiquetage (chapitre 12) sont plus satisfaisants.
- L'exploration des chemins des grammaires (section 6.5) fonctionne mieux.
- Il y a une meilleure compatibilité entre le mode morphologique, l'option de recherche de motifs "All matches", le mode debug, la recherche de motifs dans l'automate du texte (section 7.7) et les variables d'entrée de dictionnaire.
- Il y a une fonctionnalité "rechercher et remplacer" pour les graphes et pour l'automate du texte ; les fonctionnalités "défaire" et "refaire" fonctionnent maintenant aussi sur l'automate du texte.
- On peut sauvegarder un corpus traité dans le répertoire qu'on veut ; on peut sauvegarder et ouvrir des concordances.
- Il y a une fonctionnalité "Verify braces" pour contrôler si les accolades d'une grammaire sont équilibrées.
- L'automate du texte s'affiche avec un marquage en couleur plus satisfaisant.
- Les dictionnaires du français ont été mis à jour. Les orthographes anciennes ont été retirées du dictionnaire du portugais du Brésil conforme à l'orthographe actuelle. Les graphes d'affixes verbaux du malgache ont été mis à jour.
- Des graphes pour les nombres en toutes lettres dans cinq langues sont fournis.
- Le menu pour les dictionnaires a une liste des dictionnaires qui ont été ouverts récemment.
- Le menu "File edition" ouvre un répertoire adéquat en fonction du type de fichier.
- La compatibilité avec les corpus en XML est améliorée.

- L'ordre alphabétique en grec moderne est pris en charge.
- Les codes sémantiques dupliqués sont détectés dans les dictionnaires de lemmes polylexicaux aussi.
- Le format tabulaire de l'automate du texte a un bouton pour filtrer les étiquettes.
- Les boites de dialogue se ferment quand on appuie sur la touche échappement.
- Les guillemets vides (`""`) sont autorisés dans les graphes et interprétés comme `<E>`.
- Le concordancier peut maintenant traiter des textes avec des mots de plus de 5 000 caractères.
- Les boites de dialogue affichant un message d'erreur ont un bouton "Copy". La liste des sous-graphes appelés par un graphe a aussi un bouton de copie.
- On peut configurer une option dabs les préférences pour que les messages d'erreur n'apparaissent pas.
- Des fonctionnalités récentes ont été documentées dans le manuel d'utilisation.
- Diverses anomalies ont été corrigées.

Merci à Cristian Martínez, Marwin Damis, Anubhav Gupta, Renaud Mouronval, Maxime Petit, Victorien Villiers et Gilles Vollant pour leurs contributions substantielles.

Contenu

Le chapitre 1 décrit comment installer et lancer Unitex.

Le chapitre 2 présente les différentes étapes de l'analyse d'un texte.

Le chapitre 3 décrit le formalisme des dictionnaires électroniques DELA et les différentes opérations qui peuvent leur être appliqués.

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 graphe.

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

Le chapitre 7 présente le concept d'automate du texte et décrit les propriétés de cette notion. Ce chapitre décrit également les opérations sur cet objet, en particulier, comment désambiguiser les items lexicaux avec le programme ELAG.

Le chapitre 9 contient une présentation des tables du lexique-grammaire, et la description d'une méthode de construction de grammaires fondées sur ces tables.

Le chapitre 10 décrit le module d'alignement de texte basé sur l'outil XAlign.

Le chapitre 11 décrit le module de flexion des mots composés, en tant que complément du système de flexion des mots simples, présenté au chapitre 3.

Le chapitre 12 décrit le système de cascades de transducteurs CasSys.

Le chapitre 13 explique comment utiliser Unitex/GramLab par l'intermédiaire de scripts qui lancent des programmes.

Le chapitre 14 contient une description détaillée des programmes externes qui composent le système Unitex.

Le chapitre 15 contient une description de tous les formats de fichiers utilisés par Unitex.

Le lecteur trouvera en annexe la licence LGPL sous laquelle le code source Unitex est diffusé, ainsi que la licence LGPLLR qui s'applique pour les données linguistiques distribuées avec Unitex. Il y trouvera aussi la licence 2-clause BSD qui s'applique à la bibliothèque TRE, utilisée par Unitex pour les filtres morphologiques.

Contributions à Unitex

Unitex est né comme un pari sur la puissance de la philosophie Open Source dans le monde universitaire (voir http://unitexgramlab.org/why-unitex), en s'appuyant sur l'hypothèse que les gens seraient intéressés à partager leurs connaissances et leurs compétences dans un tel projet ouvert. La liste ci-dessous laisse penser que l'open source fait avancer les sciences :

- Olivier Blanc : a intégré le système ELAG à Unitex, originellement conçu par Eric Laporte, Anne Monceaux et certains de leurs étudiants, a également écrit `RebuildTfst` (anciennement appelé `MergeTextAutomaton`)
- Matthieu Constant : auteur de `Grf2Fst2`
- Julien Decreton : auteur de l'éditeur de texte intégré à Unitex, a aussi réalisé la fonctionnalité `undo` de l'éditeur de graphe
- Marwin Damis : amélioration de l'interface de l'automate du texte
- Claude Devis : ajout des filtres morphologiques, fondé sur la bibliothèque TRE
- Nathalie Friburger : auteure de `CasSys`
- Anubhav Gupta : a perfectionné `CasSys`
- Hyun-Gue Huh : auteur de l'outil de génération de dictionnaires coréens
- Claude Martineau : a travaillé sur la flexion des mots simples dans `MultiFlex`

- Cristian Martínez : a mis en place la chaine d'intégration continue et corrigé des anomalies importantes
- Renaud Mouronval : a amélioré l'exploration des chemins des grammaires
- Sebastian Nagel : a optimisé de nombreuses parties du code, il a également adapté `PolyLex` pour l'allemand et le russe
- Alexis Neme : a optimisé `Dico` et `Tokenize`, a aussi intégré `Locate` dans `Dico` pour accepter des graphes dictionnaires
- Aljosa Obuljen : auteur de `Stats`
- Sébastien Paumier : développeur principal
- Maxime Petit : a amélioré la fonctionnalité "rechercher et remplacer" pour les graphes
- Agata Savary : auteure de `MultiFlex`
- Anthony Sigogne : auteur de `Tagger` et de `TrainingTagger`
- Gilles Vollant : auteur de `UnitexTool`, a optimisé beaucoup d'aspects du code d'Unitex (mémoire, vitesse, compatibilité multi-compilateur, etc.) et corrigé d'innombrables anomalies
- Patrick Watrin : auteur de `XMLizer`, a travaillé sur l'intégration de `XAlign` à Unitex
- Anthony Sigogne : auteur de `Tagger` et de `TrainingTagger`

Il faut ajouter que Unitex serait inutile sans les précieuses ressources linguistiques qu'il renferme. Toutes ces ressources sont le fruit d'un énorme et difficile travail effectué par des personnes qui ne doivent pas être oubliées. Certaines sont citées dans les avertissements qui sont fournis avec les dictionnaires, une information complète est disponible sur :

`http://unitexgramlab.org/language-resources`

Si vous utilisez Unitex dans des projets de recherche...

Unitex a été utlisé dans plusieurs projets de recherche. Certains sont listés dans la section "Related works" de la page d'accueil d'Unitex. Si vous avez effectuer des travaux de recherche avec Unitex (ressources, projet, article, thèse, ...) et si vous désirez qu'ils soient référencés sur le site envoyez un mail à `unitex-devel@univ-mlv.fr`.

Chapitre 1

Installation d'Unitex

Unitex est un système multi-plateformes capable de fonctionner aussi bien sous Windows que sous Linux ou OS X. 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 le code source des programmes est distribué avec le logiciel, et que chacun peut le modifier et le redistribuer. Le code des programmes d'Unitex est sous licence LGPL ([36]), à l'exception de :

1. la bibliothèque de manipulation d'expressions régulières TRE de Ville Laurikari ([65]), qui est sous une licence du genre des licences BSD à 2 clauses ;
2. la bibliothèque `wingetopt` de Todd Miller et de la Fondation NetBSD, également sous une licence du genre des licences BSD à 2 clauses, plus permissive que la licence LPGL ;
3. l'analyseur syntaxique Xerces2 Java Parser, de l'Apache Software Foundation, sous licence Apache ;
4. la bibliothèque LibYAML de Kirill Simonov, qui est sous licence MIT, également plus permissive que la licence LGPL ;
5. la bibliothèque SVNKit de TMate Software, sous licence TMate.

La licence LGPL est plus permissive que la licence GPL, car elle permet d'utiliser du code LGPL dans des logiciels non libres. Dans les deux cas, le logiciel peut librement être utilisé et distribué.

Toutes les ressources linguistiques distribuées avec Unitex sont soumises à la licence LGPLLR ([54]).

Le texte complet des licences LGPL, BSD à 2 clauses, Apache, MIT, TMate et LGPLLR se trouve dans les annexes à la fin de ce manuel.

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 ou JRE (Java Runtime Environment).

Pour fonctionner en mode graphique, Unitex nécessite une version 1.7 (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 ([70]) à l'adresse suivante : `http://java.sun.com`.

Si vous travaillez sous Linux ou OS X, 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 Programme d'installation

Le programme d'installation d'Unitex/GramLab peut être téléchargé depuis cette page :

`http://releases.unitexgramlab.org/latest-stable`

1.3.1 Sous Windows

Le nom du fichier téléchargé sera par exemple :

```
Unitex-GramLab-3.2_win32-setup.exe
Unitex-GramLab-3.2_win64-setup.exe
```

Ensuite, double-cliquez sur ce fichier et suivez les instructions (fig. 1.1). Il est recommandé de désinstaller toute version existante avant d'en installer une nouvelle. Unitex/GramLab sera installé dans un répertoire (dossier) situé de préférence dans le répertoire `Program Files`, et qui sera appelé dans ce manuel le répertoire système Unitex.

Une fois l'installation terminée, une icône Unitex et une icône GramLab apparaissent sur le bureau : double-cliquez dessus pour lancer Unitex ou GramLab (voir 1.5). (Si le programme d'installation n'a pas créé ces icônes, ouvrez le répertoire Unitex système : il contient plusieurs sous-répertoires, dont un est `App`. Ce dernier répertoire contient deux fichiers `Unitex.jar` et `GramLab.jar`. Ce sont les fichiers Java qui lancent les interfaces graphiques. Double-cliquez sur l'un d'entre eux pour lancer Unitex ou GramLab (voir 1.5). Pour faciliter le lancement du programme, il est conseillé de créer des raccourcis vers ces fichiers sur le bureau.)

FIGURE 1.1 – Programme d'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 le seul utilisateur de votre machine, vous pouvez effectuer l'installation vous-même.

Le programme d'installation sous Windows peut aussi être lancé en ligne de commande et dans ce cas il accepte plusieurs paramètres optionnels. En voici quelques-uns :

- `/AllUsers` Par défaut i'installation vaudra pour tous les utilisateurs
- `/CurrentUser` Par défaut i'installation vaudra pour l'utilisateur seulement
- `/D C:\path\without quotes\` Spécifie le répertoire d'installation par défaut
- `/NCRC` Saute le contrôle de redondance cyclique
- `/S` Supprime toutes les questions

Sous Windows 7, on peut avoir des problèmes avec le fichier de configuration d'Unitex, car Unitex essaie de le créer dans le sous-répertoire Unitex et Windows 7 le lui interdit.

1.3.2 Sous GNU/Linux et OS X

Le nom du fichier téléchargé sera par exemple :

```
Unitex-GramLab-3.2-linux-i686.run
Unitex-GramLab-3.2-linux-x86_64.run
```

Donnez-lui les droits d'exécution, par exemple par :

```
chmod a+x Unitex-GramLab-3.2-linux-i686.run
```

Le fichier `.run` est une archive qu'on décompresse en l'exécutant :

```
./Unitex-GramLab-3.2-linux-i686.run
```

Le fichier d'installation sous GNU/Linux et OS X accepte plusieurs paramètres de ligne de commande optionnels. En voici quelques-uns :

- `--confirm` Demander avant de lancer le script d'installation
- `--quiet` N'afficher que les messages d'erreur
- `--noexec` Ne pas lancer le script d'installation
- `--target dir` Spécifier le répertoire d'installation par défaut

1.4 Installation manuelle

On peut aussi installer Unitex/GramLab manuellement à l'aide du paquet de distribution. Téléchargez-le depuis cette page :

http://releases.unitexgramlab.org/latest-stable/source

Le nom du fichier téléchargé sera par exemple :

```
Unitex-GramLab-3.2-source-distribution.zip
```

Créez un répertoire que vous nommez par exemple `Unitex3.2`, de préférence dans le répertoire `Program Files`, et qui sera appelé dans ce manuel le répertoire système Unitex. Décompressez-y le paquet de distribution.

Si votre ordinateur est sous un des systèmes d'exploitation suivants, l'installation est terminée : Windows (32 bits ou 64 bits), GNU/Linux (i686 ou x86_64) et OS X (10.7+). (S'il fonctionne sous un autre système Unix, comme FreeBSD, ou s'il a une autre architecture de processeur, comme ARM, allez dans le répertoire `App/install` et lancez :

```
sh setup
```

Ce script vérifie si Java est installé, compile les sources C++, crée les répertoires personnels de travail Unitex et GramLab et met en place des raccourcis sur le bureau [1].)

Une fois l'installation terminée, le répertoire système Unitex contient plusieurs sous-répertoires dont un est `App`.

- Sous Windows : `App` contient des fichiers `Unitex.jar` et `GramLab.jar`. Ce sont les fichiers Java qui lancent les interfaces graphiques. Double-cliquez sur l'un d'entre eux pour lancer Unitex ou GramLab (voir 1.5). Pour faciliter le lancement du programme, il est conseillé de créer des raccourcis vers ces fichiers sur le bureau.
- Sous Linux ou OS X : le répertoire `App` contient deux scripts shell `Unitex` et `GramLab`. Lancez l'un d'entre eux pour démarrer Unitex or GramLab (voir 1.5). Si vous avez lancé le script d'installation, il a normalement fait apparaitre sur le bureau des raccourcis vers ces fichiers.

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 dans "Info>Preferences...>Directories". Pour créer un répertoire, cliquez sur l'icône représentant un dossier (voir figure 1.4).

Sous Linux et OS X, le programme créera automatiquement un répertoire personnel de travail, appelé `/unitex`, dans votre répertoire `$HOME`.

Le répertoire personnel de travail, ou répertoire de l'utilisateur, vous permettra de stocker vos données Unitex personnelles. Pour chaque langue que vous utiliserez, le programme copiera l'arborescence de la langue dans votre répertoire de travail, à l'exception des dictionnaires. Vous pourrez ainsi modifier à votre guise votre copie des données sans risquer d'endommager les données du système, stockées dans le répertoire système Unitex.

On peut changer la taille et la police des caractères du menu par "Info > Preferences > General". Cette configuration est sauvegardée dans le fichier Config pour la prochaine fois qu'on lance Unitex.

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 dans le répertoire système Unitex, 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 vous êtes le seul utilisateur concerné par la langue, vous pouvez copier le répertoire en question dans votre répertoire de travail. Vous pourrez ainsi travailler sur cette langue sans qu'elle soit proposée aux autres utilisateurs.

1. Si vous voulez uniquement compiler les sources C++, extrayez les fichiers du paquet de distribution, placez-vous dans le répertoire `Src/C++/build` et lancez `make install`.

FIGURE 1.2 – Première utilisation sous Windows

FIGURE 1.3 – Première utilisation sous Linux

1.7 Désinstallation

Quel que 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 OS X si vous avez créé un alias.

1.8 Unitex pour les développeurs

Si vous êtes programmeur, cela peut vous intéresser de lier votre code avec les sources C++ d'Unitex. Pour faciliter cette opération, vous pouvez compiler Unitex en tant que bibliothèque dynamique qui contient toutes les fonctions Unitex, sauf les `main`s, bien sûr. La page http://docs.unitexgramlab.org/projects/unitex-library/fr/latest/ est une documentation sur la bibliothèque. Les sources C++ d'Unitex contiennent du code pour des bindings Java JNI, Ruby et Microsoft .NET. La page https://github.com/patwat/python-unitex contient des bindings Python.

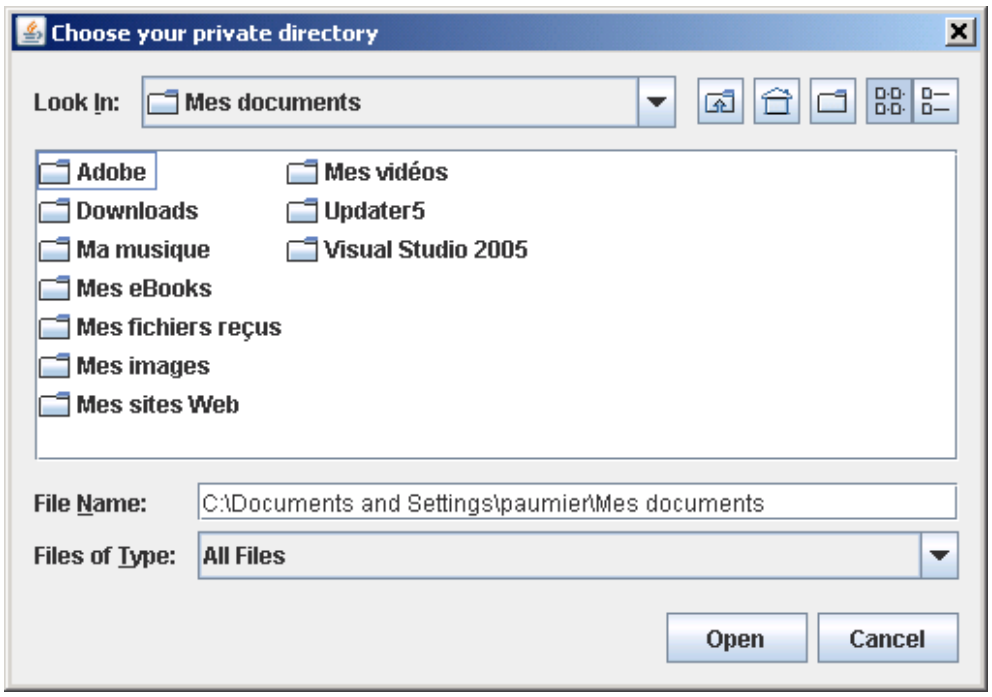


FIGURE 1.4 – Création du répertoire personnel de travail

Sous Linux/OS X, tapez :

```
make LIBRARY=yes
```

et vous obtiendrez une bibliothèque nommée `libunitex.so`. Si vous souhaitez produire DLL Windows nommée `unitex.dll`, utilisez les commandes suivantes :

Windows : `make SYSTEM=windows LIBRARY=yes`
Cross-compilation avec mingw32 : `make SYSTEM=mingw32 LIBRARY=yes`

dans tous les cas, vous obtiendrez aussi un programme nommé `Test_lib(.exe)`. Si tout a bien fonctionné, ce programme devrait afficher l'écran suivant :

```
Expression converted.
Reg2Grf exit code: 0

#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
#
7
"<E>" 100 100 1 5
"" 100 100 0
"a" 100 100 1 6
"b" 100 100 1 4
"c" 100 100 1 6
"<E>" 100 100 2 2 3
"<E>" 100 100 1 1
```

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 de travail. Si vous utilisez une langue pour la première fois, Unitex recopie le répertoire système de cette langue dans votre répertoire de travail, à l'exception des dictionnaires, afin d'économiser de l'espace disque.

Attention, si vous avez déjà un répertoire de travail pour une langue donnée, Unitex n'essaiera pas de recopier les données système dedans. Ainsi, si une mise à jour a modifié un fichier de ressource autre qu'un dictionnaire, il vous faudra soit faire une mise à jour manuelle du fichier dans votre répertoire de travail, soit supprimer votre répertoire pour la langue concernée et laisser à Unitex le soin de le recréer.

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.

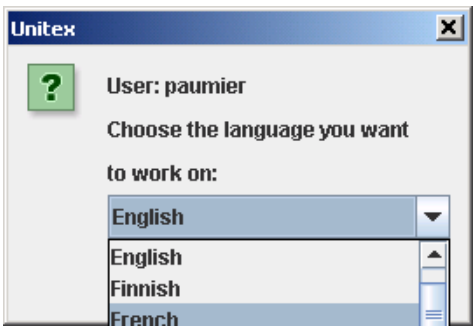


FIGURE 2.1 – Sélection de la langue au lancement d'Unitex

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 [16]).

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 français, Unitex vous proposera de convertir votre texte [1], en supposant qu'il est codé avec un codage français. Par défaut, Unitex vous propose soit de remplacer le texte original, soit de renommer le fichier d'origine en insérant `.old` au début de son extension. Par exemple, si un fichier ASCII est nommé `biniou.txt`, le processus de conversion va créer une copie de ce fichier ASCII nommée `biniou.old.txt`, et va remplacer le contenu de `biniou.txt` par son équivalent en 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 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.

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

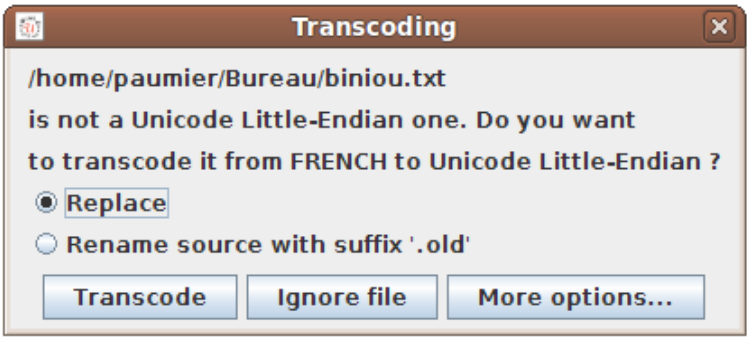


FIGURE 2.2 – Conversion automatique d'un texte non Unicode

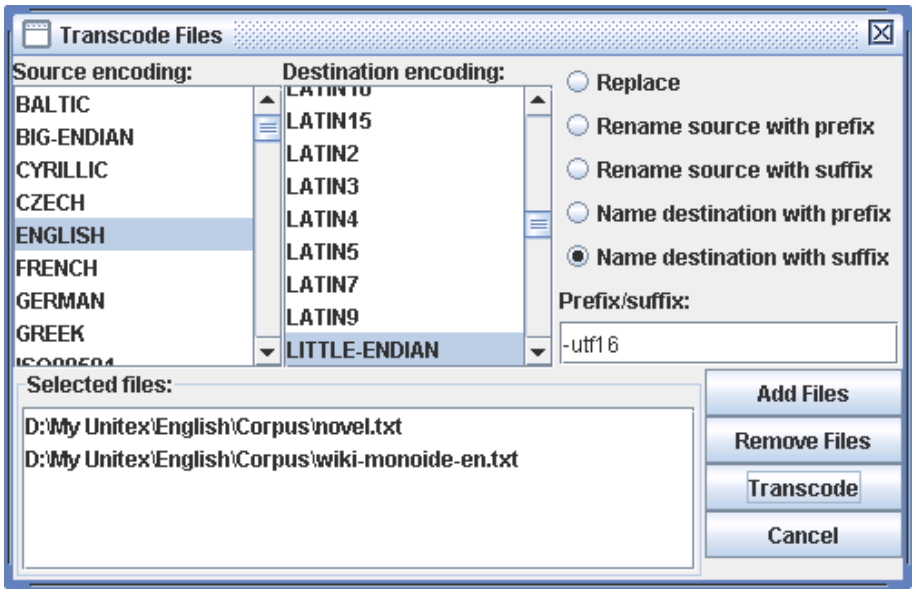


FIGURE 2.3 – Conversion de fichiers

Pour obtenir du texte au bon format, vous pouvez également utiliser un traitement de texte comme le logiciel libre OpenOffice.org ([75]) ou Microsoft Word, et sauvegarder votre document au format "Texte unicode". Dans OpenOffice Writer, vous devez choisir le format "Coded Text (*.txt)" puis le codage "Unicode" dans la fenêtre de configuration comme le montre la figure 2.4.
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.

Vous pouvez choisir le codage par défaut, UTF16LE, UTF16BE ou UTF8 dans l'onglet, "Encoding" grâce au sous-menu "Preference" dans le menu "Info". Ce codage n'est valide que

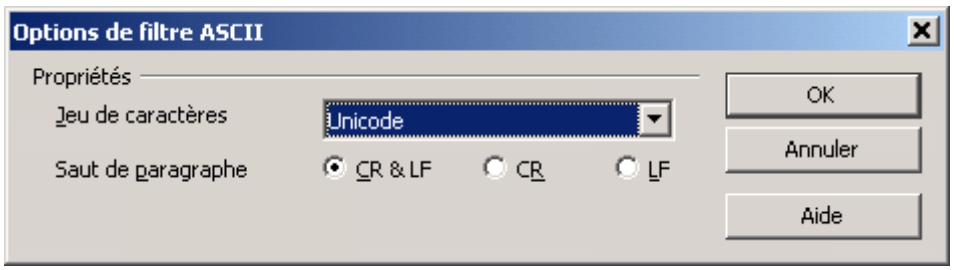


FIGURE 2.4 – Sauvegarde en Unicode dans OpenOffice Writer

pour la langue courante.

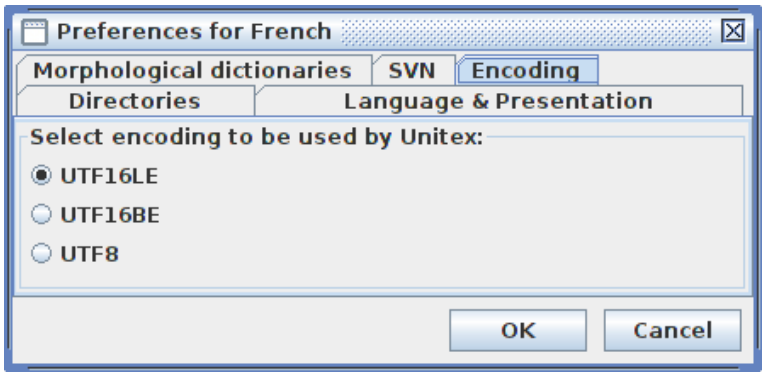


FIGURE 2.5 – Choix de l'encodage par défaut pour la langue courante

2.3 Édition 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 onglets. 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 2.6, 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. 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 obtiendrez ainsi les entrées voulues, sans ambiguïtés avec toutes les autres occurrences de la lettre `t`.

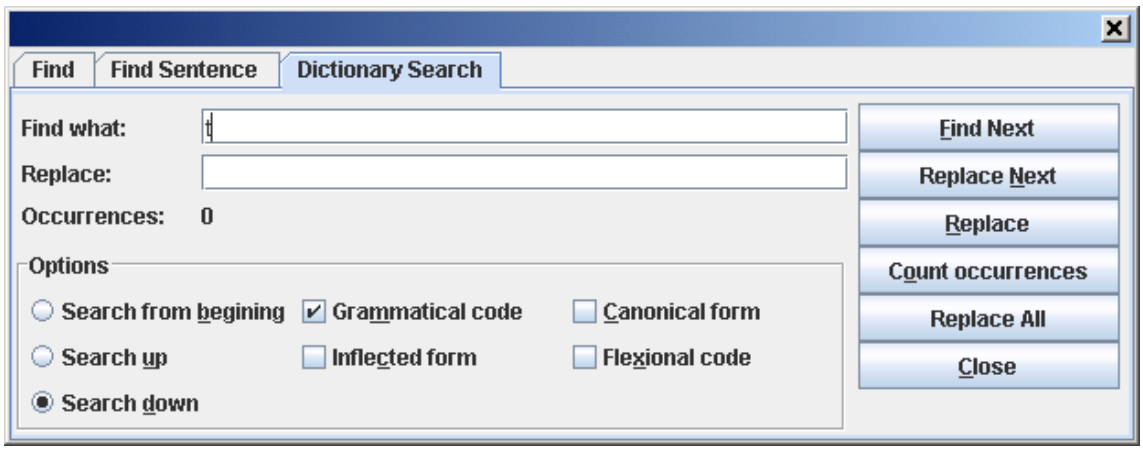


FIGURE 2.6 – Recherche du trait sémantique `t`dans un dictionnaire électronique

2.4 Ouverture d'un texte

Unitex propose d'ouvrir deux types de fichiers textes. 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 bruts. Pour utiliser un texte, il faut donc commencer par ouvrir le fichier `.txt` correspondant en cliquant sur "Open..." dans le menu "Text".

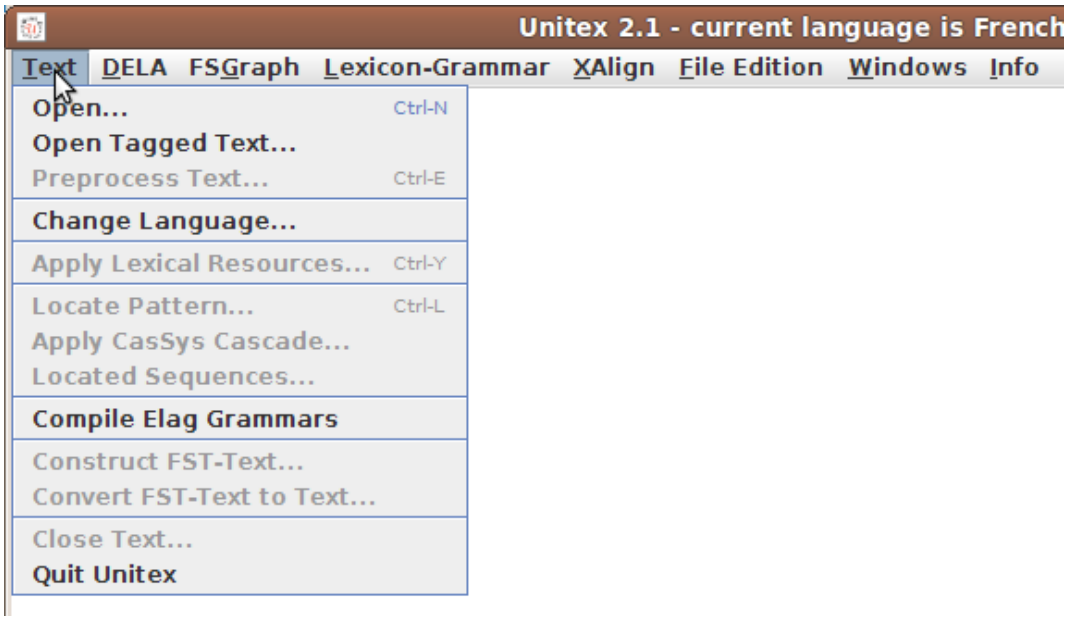


FIGURE 2.7 – Menu Text

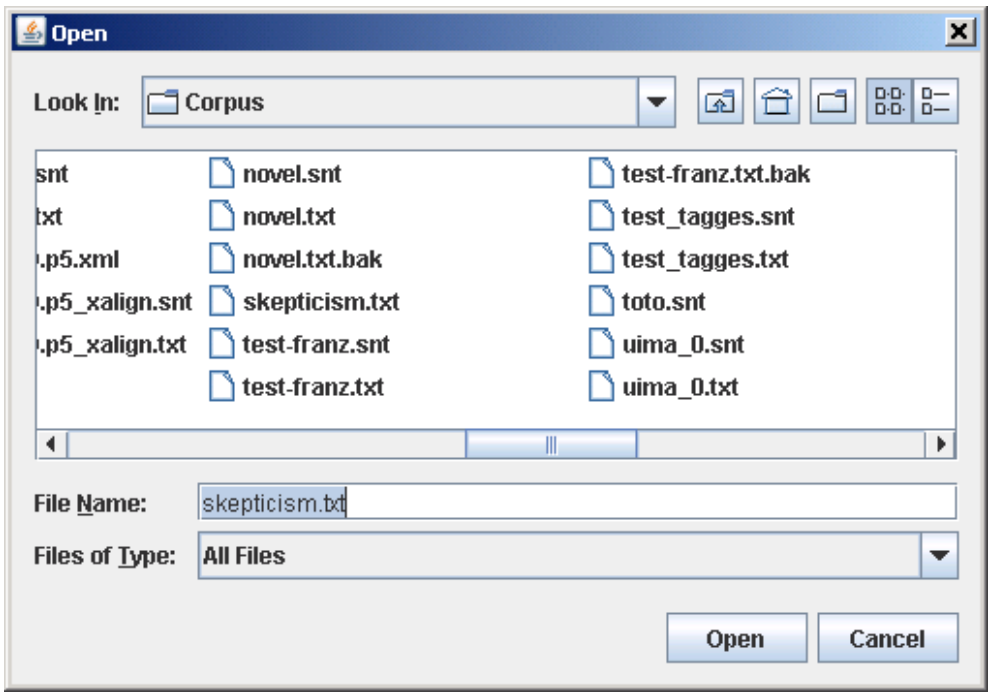


FIGURE 2.8 – Ouverture d'un texte Unicode

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 2.9. 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

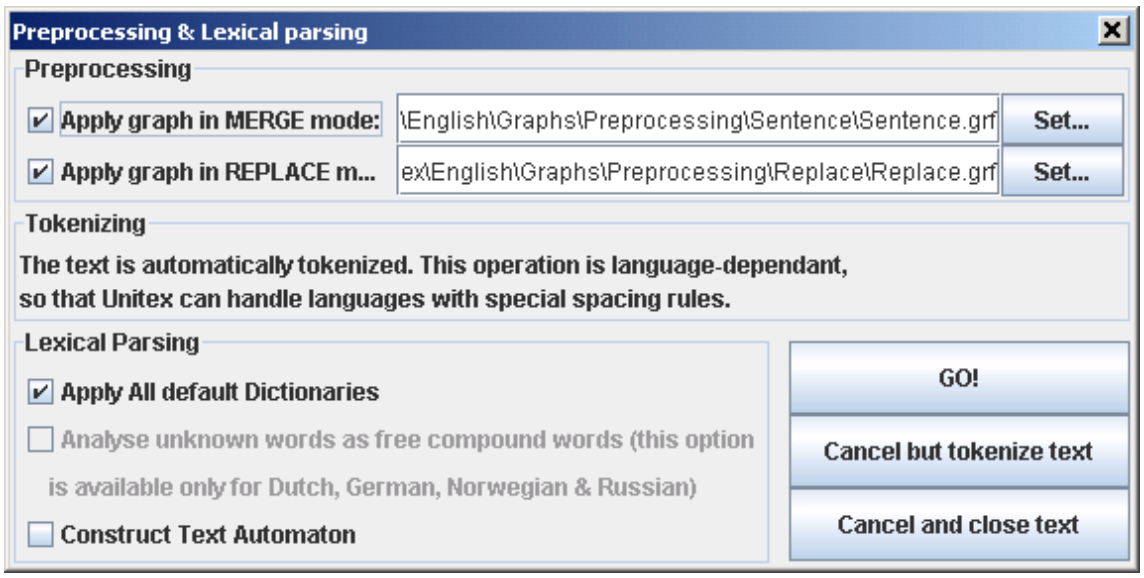


FIGURE 2.9 – Fenêtre de prétraitement

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 2.10 montre un exemple de grammaire de découpage en phrases.

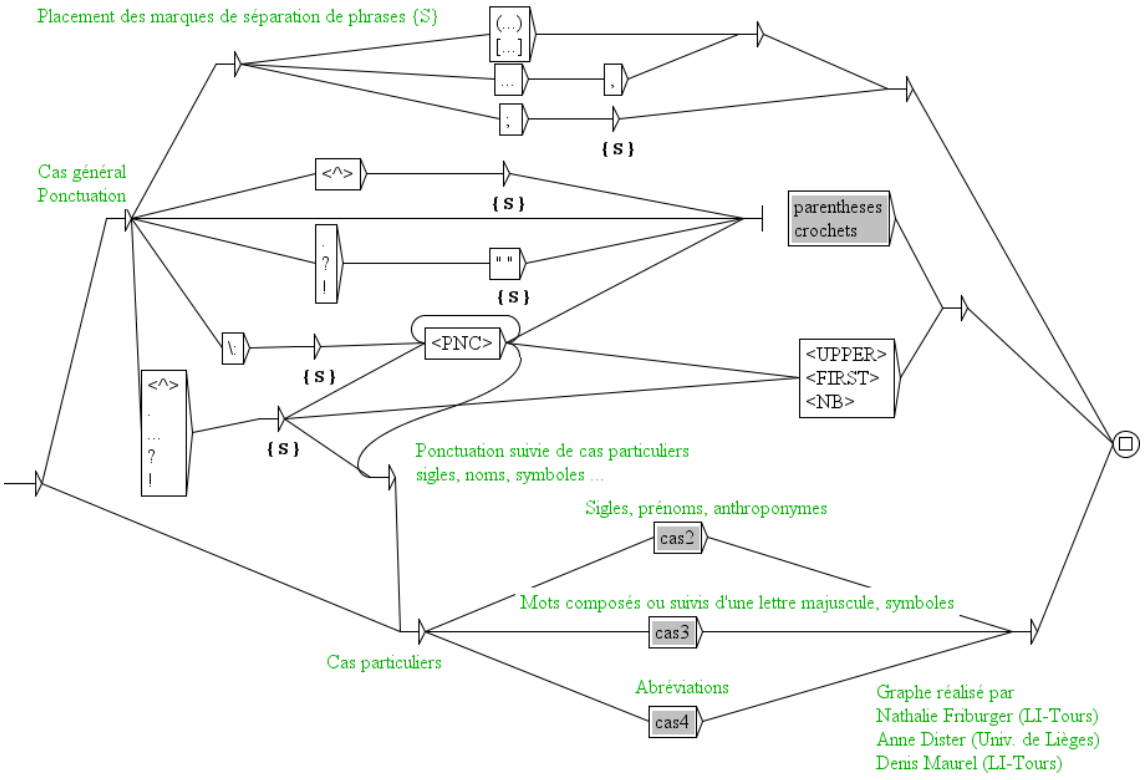


FIGURE 2.10 – Grammaire de découpage en phrases pour le français

Lorsqu'un chemin de la grammaire reconnaît une séquence dans le texte et que ce chemin produit le symbole délimiteur de phrases `{S}`, on insère ce symbole dans le texte. Ainsi, un chemin de la grammaire de la figure 2.10 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, ou méta-symboles, suivants :

- <E> : mot vide, ou epsilon. Reconnaît la séquence vide ;
- <WORD> : reconnaît n'importe quelle suite de lettres ;
- <LOWER> : reconnaît n'importe quelle suite de lettres minuscules ;
- <UPPER> : reconnaît n'importe quelle suite de lettres majuscules ;
- <FIRST> : reconnaît n'importe quelle suite de lettres commençant par une majuscule ;
- <NB> : reconnaît n'importe quelle suite de chiffres contigus (1234 est reconnu mais pas 1 234) ;
- <PNC> : reconnaît les symboles de ponctuation ; , ! ? : ainsi que les points d'exclamation et d'interrogation inversés de l'espagnol et quelques signes de ponctuation asiatiques ;
- <^> : reconnaît un retour à la ligne ;
- # : interdit la présence de l'espace.

Les anciens codes correspondant à <WORD>, <LOWER>, <UPPER> et <FIRST> étaient respectivement <MOT>, <MIN>, <MAJ> et <PRE>. Ils restent opérationnels afin de conserver la compatibilité descendante du système avec les graphes existants. Même s'il n'est pas prévu de supprimer ces codes, on recommande de les éviter dans les graphes conçus pour fonctionner avec les versions plus récentes [2], pour ne pas faire augmenter inutilement le nombre de masques lexicaux en usage.

Par défaut, l'espace est facultatif entre deux boîtes. Si l'on veut interdire la présence de ce séparateur, il faut utiliser le symbole spécial #. À l'inverse, si vous souhaitez forcer la présence de l'espace, vous devez utiliser la séquence " ". Les lettres minuscules et majuscules sont définies par un fichier alphabet (voir chapitre 15). Pour plus de détails sur les graphes, voir le chapitre 5. Pour plus de détails sur le découpage d'un texte en phrases, voir [21]. La grammaire utilisée se nomme Sentence.fst2 et se trouve dans le répertoire suivant :

```
/(répertoire personnel)/(langue)/Graphs/Preprocessing/Sentence
```

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. À partir de la version 3.1bêta, révision 4072 du 2 octobre 2015.

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 ce était 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)/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 2.11 une grammaire qui normalise des contractions verbales en anglais.

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

Une unité lexicale peut être :

- soit le délimiteur de phrases `{S}` ;

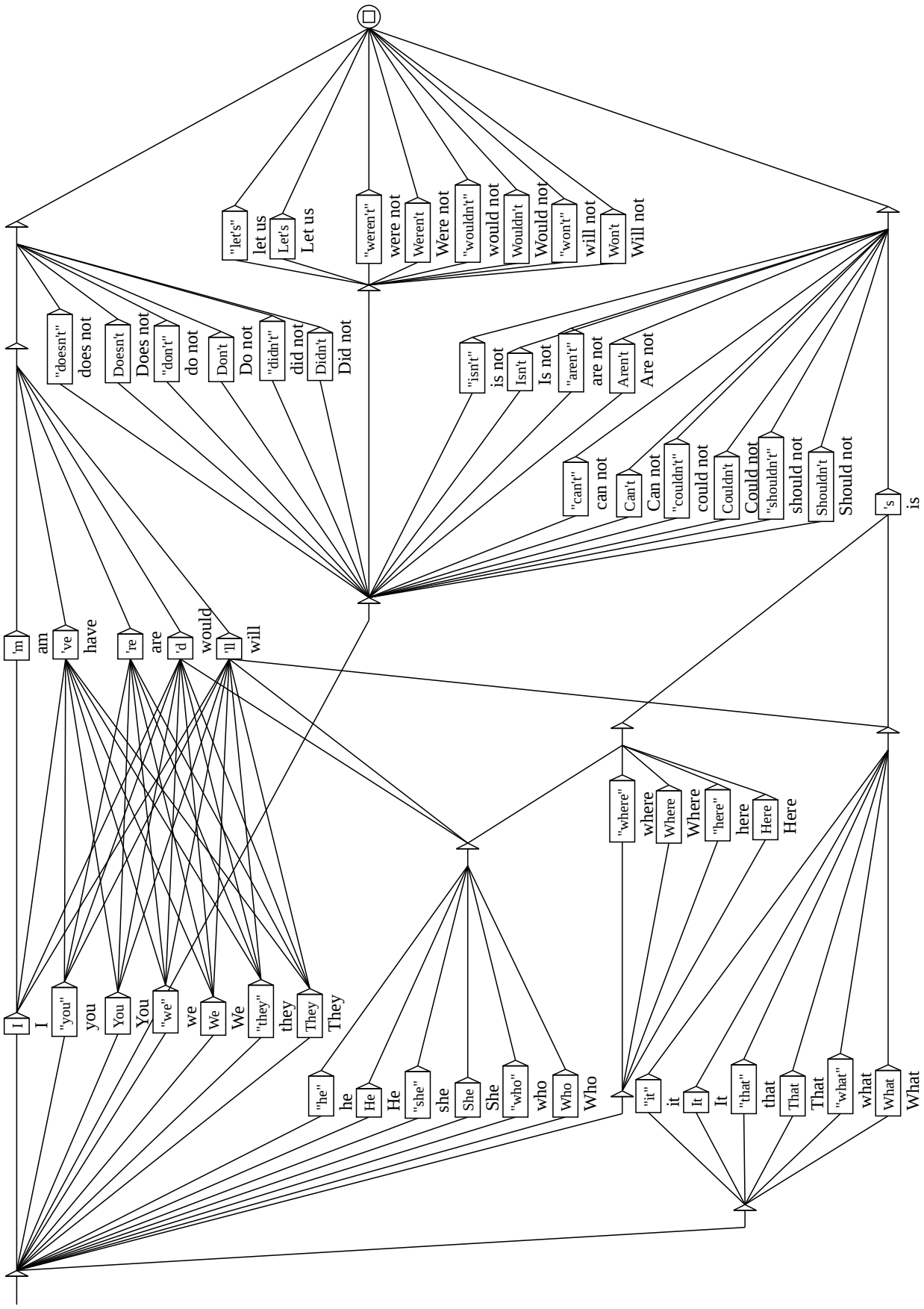


FIGURE 2.11 – Grammaire de normalisation de formes verbales en anglais

- le marqueur `{STOP}`. Contrairement au délimiteur de phrases `{S}`, le marqueur `{STOP}` ne peut JAMAIS être reconnu par une grammaire, de quelque façon que ce soit. Il peut être utilisé dans un corpus pour délimiter des élements. Par exemple, si un corpus est formé de nouvelles séparées par `{STOP}`, il est impossible pour une grammaire de reconnaître une séquence qui chevauche la fin d'une nouvelle et le début de la suivante ;
- une étiquette lexicale `{aujourd'hui,.ADV}` ;
- une séquence de lettres contiguës (les lettres sont définies dans le fichier alphabet de la langue) ;
- un (et un seul) caractère different d'une lettre, i.e. tous les caractères non définis dans le fichier alphabet de la langue courante ; s'il s'agit d'une newline, il est remplacé par un espace.

Pour les autres langues, le découpage est effectué caractère par caractère, à l'exception du délimiteur de phrases `{S}` le marqueur `{STOP}` et des étiquettes lexicales. Ce découpage basique garantit le fonctionnement d'Unitex, mais limite l'optimisation des opérations de recherche de motifs. Par exemple une boite contenant `aujourd'hui` ne reconnaitra pas l'étiquette lexicale `{aujourd'hui,.ADV}`. Pour la reconnaitre, il faut placer dans la boite le masque lexical : `<aujourd'hui>`.

Quel que 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 :

Pour plus de détails, voir le chapitre 15.

Indice	0	1	2	1	3	1	4	1	2	5
Unité lexicale correspondante	*UN*		*sou*		*est*		*UN*		*sou*	.

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

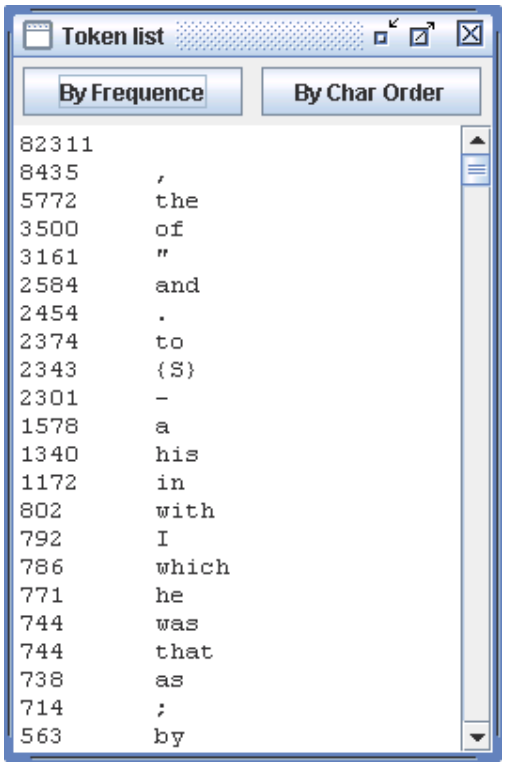


FIGURE 2.12 – Unités lexicales d'un texte anglais triées par fréquence

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
```

La séquence *Igor* n'étant ni un mot simple du français, ni une partie de mot composé, a été considérée comme un 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 2.13 montre les résultats pour un texte anglais.

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 2.14) qui permet de choisir la liste des dictionnaires à appliquer.

La liste "User resources" recense tous les dictionnaires `.bin` et `.fst2` présents dans le répertoire `(langue)/Dela` de l'utilisateur. Les dictionnaires du système sont listés dans le cadre intitulé "System resources". Utilisez <Ctrl+click> pour sélectionner plusieurs dictionnaires. Les dictionnaires systèmes sont appliqués avant les dictionnaires utilisateurs. Vous pouvez choisir l'ordre des dictionnaires des listes utililisateur et système à l'aide des flèches haut et bas (voir figure 2.14). Le bouton "Set Default" vous permet de définir la sélection courante de dictionnaires comme sélection par défaut. Cette sélection par défaut sera utilisée lors du prétraitement si vous choisissez l'option "Apply All default Dictionaries". Si vous effectuez un clic droit au-dessus d'un nom de dictionnaire, la documentation du dictionnaire, si elle existe, s'affichera dans le cadre inférieur.

2.5.6 Analyse des mots composés libres en néerlandais, 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` 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.

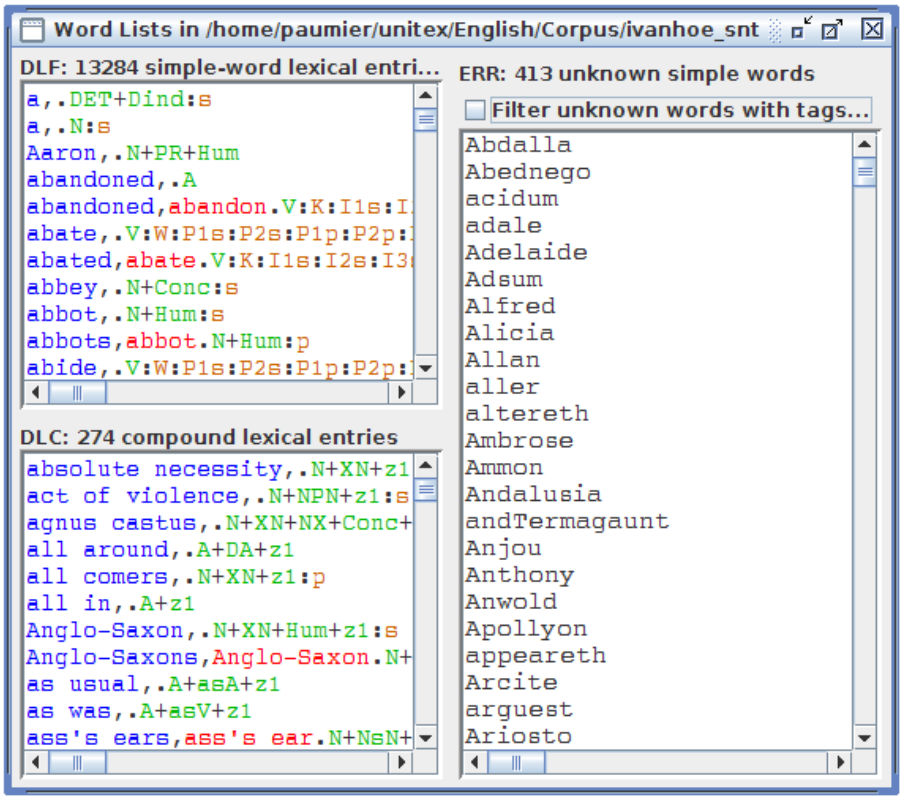


FIGURE 2.13 – Résultats de l'application de dictionnaires sur un texte anglais

2.6 Ouverture d'un texte taggué

Un texte taggué est un texte contenant des entrées lexicales entre accolades comme par exemple :

I do not like the {square bracket,.N} sign ! {S}

De tels tags permettent de lever des ambiguïtés en interdisant tout autre interprétation. Dans notre exemple, on ne pourra pas reconnaître square bracket comme combinaison de deux mots simples.

Toutefois, la présence de ces tags peut perturber l'application des graphes de prétraitement. L'utilisateur dispose donc de la commande "Open Tagged Text..." dans le menu "Text", grâce à laquelle il peut ouvrir un texte contenant des tags sans que les graphes de prétraitements ne soient appliqués, comme on le voit sur la figure 2.15.

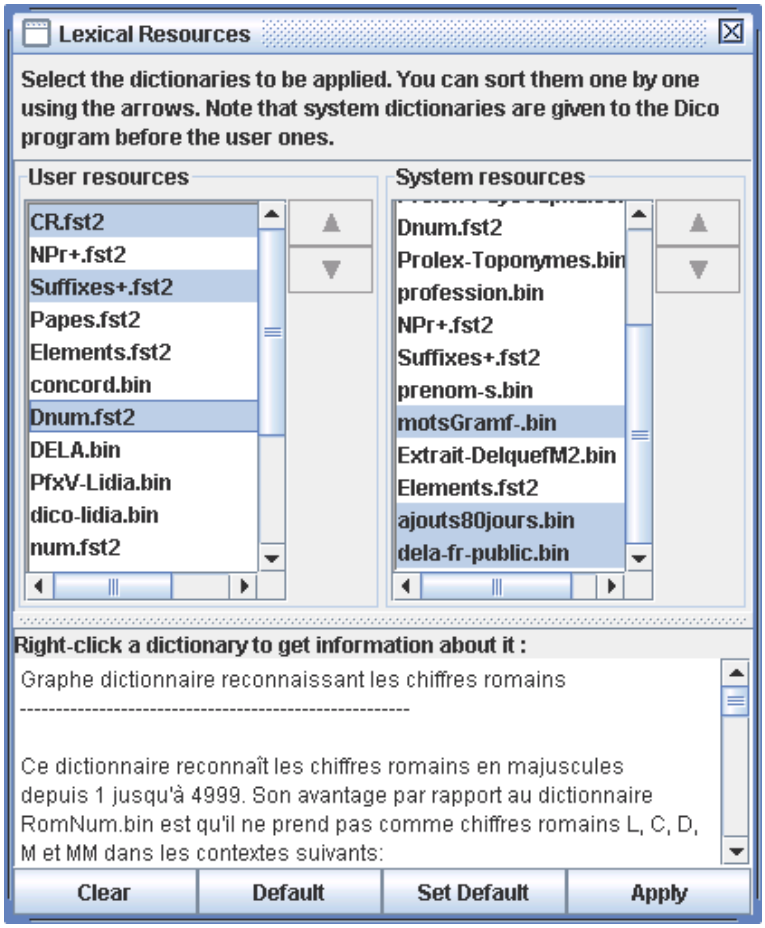


FIGURE 2.14 – Paramétrage de l'application des dictionnaires

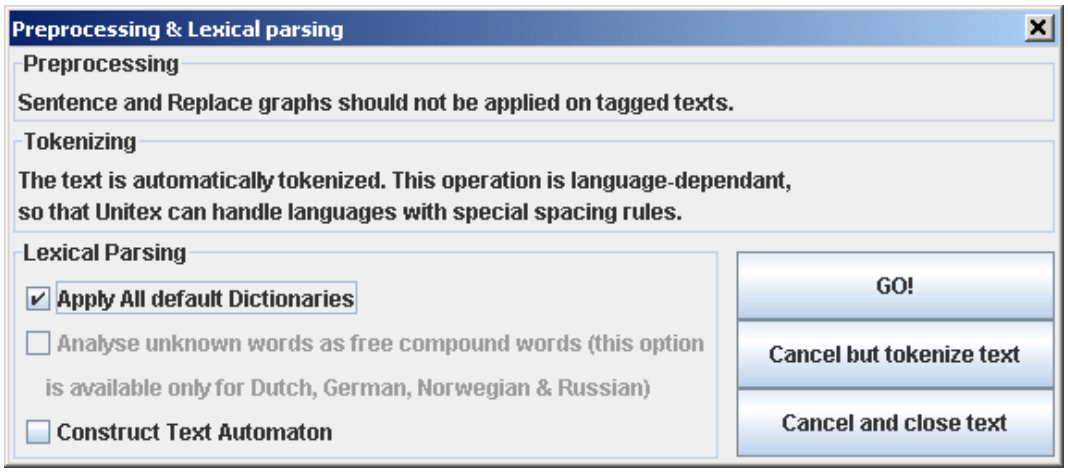


FIGURE 2.15 – Prétraitement d'un texte taggué

Chapitre 3

Dictionnaires

3.1 Les dictionnaires DELA

Les dictionnaires électroniques utilisés par Unitex utilisent le formalisme 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 soient 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 (lemme) 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` ndique 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. Ainsi, `:mp:fp` signifie "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 comprime les dictionnaires.

REMARQUE IMPORTANTE : il est possible d'utiliser le point et la virgule dans une entrée de dictionnaire. Pour cela, il faut les déspécialiser 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 composés de `P`, `3`, `s` et d'un espace.

Il est possible d'insérer des lignes de commentaires dans un dictionnaire DELAF ou DELAS, en faisant débuter 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 =. 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 =, déspécialisez-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 = déspécialisés 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=mc2.
```

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 laissé 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+Anl
```

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 cidessus 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 3.5.

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.).

Code	Signification	Exemples
A	adjectif	fabuleux, broken-down
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

TABLE 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
Anl	animal	cheval de race
AnlColl	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

TABLE 3.2 – Quelques codes sémantiques

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.).

Malgré une base commune à la plupart des langues, les dictionnaires contiennent des particularités de codage propres à chaque langue. Ainsi, les codes de flexion 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.

Code	Signification
m	masculin
f	féminin
n	neutre
s	singulier
p	pluriel
1, 2, 3	1st, 2nd, 3rd 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

TABLE 3.3 – Codes flexionnels usuels

Les codes présentés ne sont absolument pas limitatifs. Chaque utilisateur peut introduire ses 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
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 Recherche d'un mot dans un dictionnaire

Vous pouvez rechercher un mot dans plusieurs dictionnaires de deux manières :

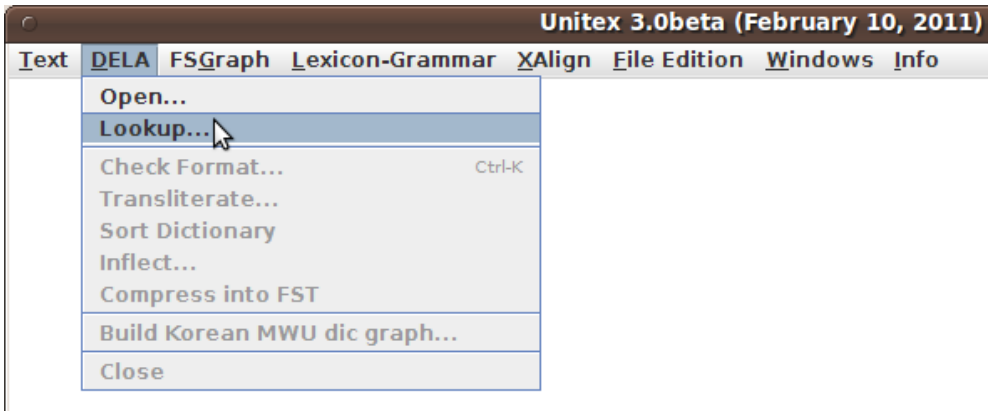


FIGURE 3.1 – Menu "DELA"

Si vous avez ouvert un dictionnaire, la fenêtrre affichée contient un champ qui vous permet d'effectuer une recherche. Si le mot apparaît dans le dictionnaire, le bouton "Find" surligne la première entrée correspondante. Si plusieurs entrées correspondent, vous pouvez les parcourir en cliquant sur les deux boutons en forme de flèche.

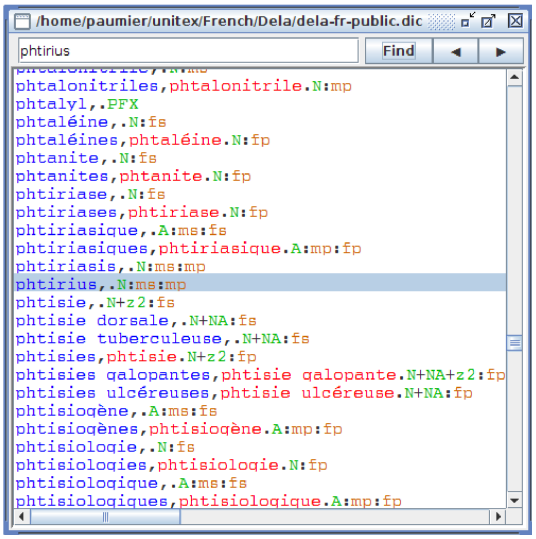


FIGURE 3.2 – Recherche d'un mot dans un dictionnaire

Vous pouvez aussi rechercher un mots dans plusieurs dictionnaires en cliquant sur le bouton "Lookup" du menu "DELA". Vous pouvez ensuite sélectionner les dictionnaires dans lesquels rechercher le mot que vous avez entré.

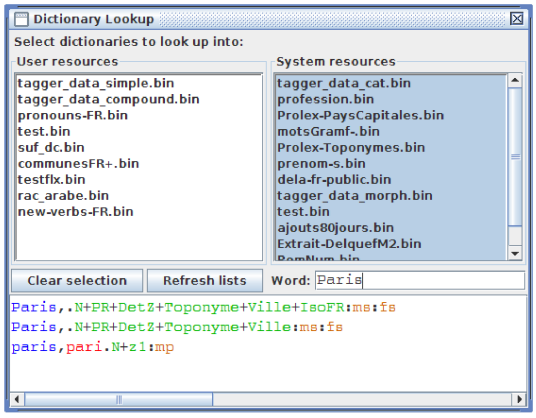


FIGURE 3.3 – Recherche d'un mot dans plusieurs dictionnaires

3.3 Vérification du format du 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 sauvés 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 `CheckDic` 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` . Pour vérifier le format d'un dictionnaire, il faut tout d'abord l'ouvrir en cliquant sur "Open..." dans le menu "DELA".
Chargeons le dictionnaire de la figure 3.4. Pour lancer la vérification automatique, cliquez sur "Check Format..." dans le menu "DELA". la fenêtre de la figure 3.5 apparaît alors. Cette fenêtre vous permet de choisir le type du dictionnaire que vous voulez vérifier. Les résultats de la vérification du dictionnaire de la figure 3.4, sont présentés sur la figure 3.6.

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.

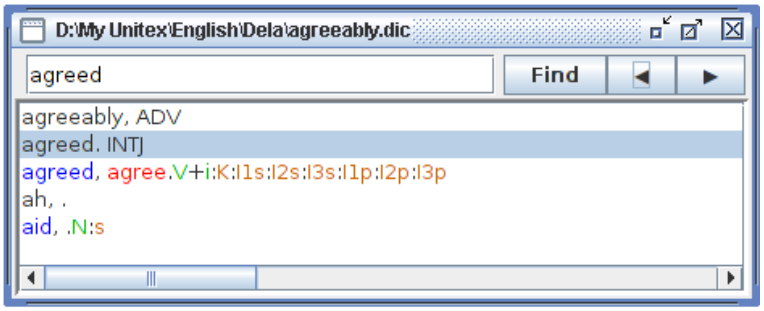


FIGURE 3.4 – Exemple de dictionnaire

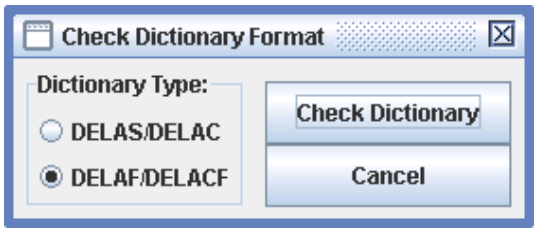


FIGURE 3.5 – Vérification automatique d'un dictionnaire

3.4 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 14).

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 :

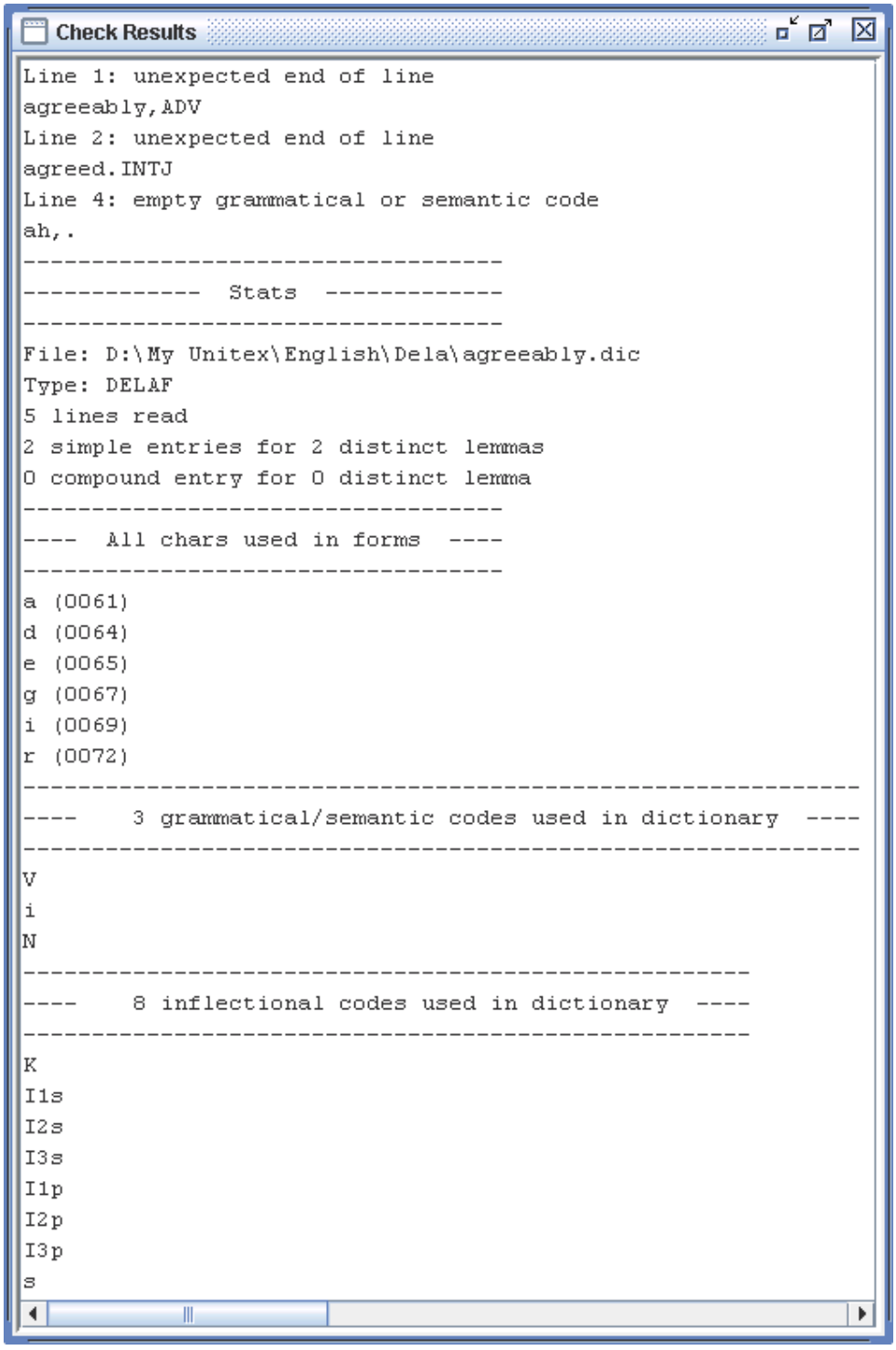


FIGURE 3.6 – Résultats d'une vérification automatique

```
AÀÂÄaàâä
Bb
CÇcç
Dd
EÉÈÊËeéèêë
```

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.

3.5 Flexion automatique

3.5.1 Flexion des mots simples

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 :

```
aviatrix,N4+Hum
matrix,N4+Math
radix,N4
```

Le premier code rencontré est interprété comme le nom de la grammaire à utiliser pour fléchir la forme canonique. Il y a deux formes possibles :

- `N4` : nom de la grammaire=`N4.fst2`, codes grammaticaux=`N` (le plus long préfixe uniquement composé de lettres)
- `N(NC_XXX)` : nom de la grammaire=`NC_XXX.fst2`, codes grammaticaux=`N`

Ces grammaires de flexion seront automatiquement compilées si besoin est. 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 3.7 permet d'indiquer au programme de flexion le répertoire dans lequel se trouvent les grammaires de flexion. Par défaut, le sous-répertoire `Inflection` du répertoire de la langue

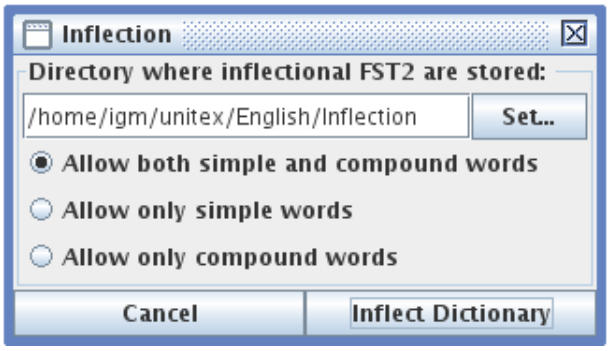


FIGURE 3.7 – Configuration de la flexion automatique

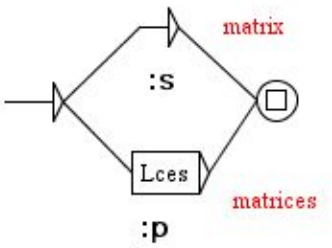


FIGURE 3.8 – Grammaire de flexion N4

courante est utilisé. On peut aussi spécifier quels types de mots le dictionnaire est supposé contenir. Si une entrée non conforme est rencontrée, un message d'erreur sera affiché.

La figure 3.8 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 `:s`. Le second retranche une lettre grâce à l'opérateur `L`, ajoute ensuite le suffixe `ces` et ajoute le code flexionnel `:mp`.

Voici les opérateurs utilisables :

- `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 4e 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` and `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 ;

- `D` (delete) supprime une lettre de l'entrée, en décalant tout ce qui se trouve à sa droite. Si l'on souhaite par exemple fléchir le mot roumain `european` en `europeni`, on utilisera la séquence `LDRi`. Le `L` positionnera le curseur sur la lettre `a`, `D` va supprimer le `a`, en décalant le `n` sur la gauche, puis `Ri` va rétablir le `n` et ajouter un `i`.

- `U` (unaccent) enlève l'accent du caractère courant s'il en comporte un. Par exemple la séquence `LLUx` appliquée au mot `mangés` produit la forme fléchie `mangex`, puisque `U` à transformé le `é` en `e`.

- `P` (uppercase) met en majuscule la première lettre de la pile. Par exemple, la séquence `Px` transforme `foo` en `Foox`.

- `W` (lowercase) met en minuscule la première lettre de la pile.

- `<R=?>` remplace la première lettre de la pile par la lettre `?`.

- `<I=?>` insère la lettre `?` avant la première lettre de la pile.

- `<X=n>` supprime les n premières lettres de la pile.

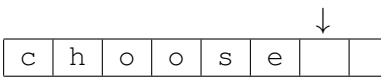
Il y a également deux opérateurs spéciaux pour le Coréen :

- `J` supprime une lettre Jamo. Si le caractère est un Hangul, ce caractère est d'abord remplacé par sa séquence équivalente en alphabet Jamo, ensuite, la dernière lettre Jamo est supprimée. Si le caractère n'est ni un Jamo, ni un Hangul, une erreur est produite.

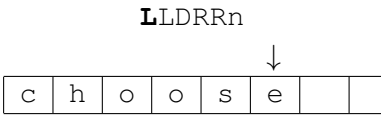
- `.` (latin dot) insère une limite de syllabe. Ceci a un effet de ford, si le haut de la pile contient des lettres Jamo, elles sont recombinées en Hangul.

Voici un exemple qui décrit la flexion de `choose` en `chosen` grâce à la séquence d'opérateurs `LLDRRn` :

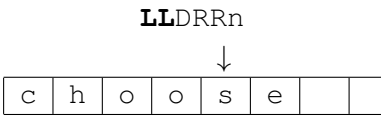
- Étape 0 : initialisation de la pile avec la forme canonique ; on place le curseur après la dernière lettre :



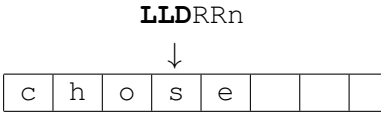
- Étape 1 : on décale le curseur vers la gauche :



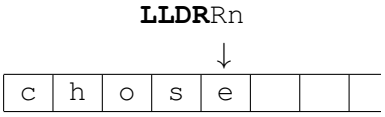
- Étape 2 : on décale une seconde fois le curseur vers la gauche :



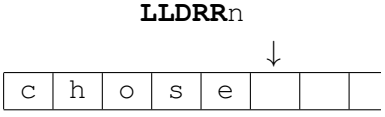
- Étape 3 : on décale tout ce qui est à droite du curseur vers la gauche :



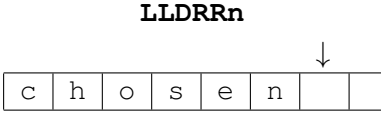
- Step 4 : on décale le curseur vers la droite :



- Step 5 : on décale encore le curseur vers la droite :



- Step 6 : on écrit un `n`



Une fois la séquence d'opérateurs épuisée, on prend le contenu de la pile jusqu'avant le curseur pour former la forme fléchie (ici `chosen`).

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.

La figure 3.9 montre le dictionnaire obtenu après flexion du DELAS de notre exemple.

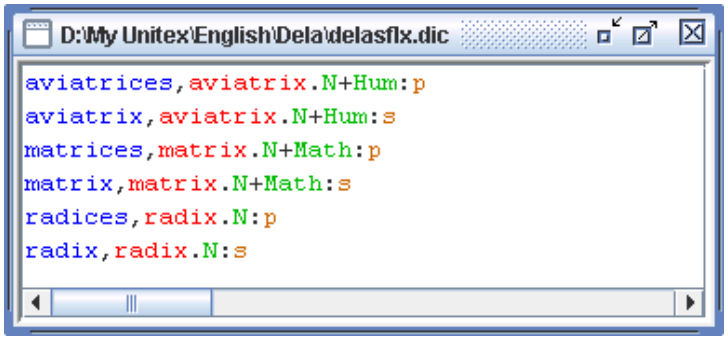


FIGURE 3.9 – Résultat de la flexion automatique

3.5.2 Opérateurs de flexion avancés

Dans certaines langues, le processus de flexion entraine une modification de la racine du mot. Plusieurs opérateurs ont été développés pour faciliter ce type de traitement. Ils permettent de rechercher et d'enlever un suffixe du mot `W` à fléchir. Cette opération peut être accompagnée de la mémorisation dans une variable ($ ou £) d'un facteur de ce suffixe. Ces opérateurs peuvent prendre les formes suivantes :

- `<X$Y>` : On recherche à la fin du mot `W` le suffixe `Y`. Puis, on recherche à partir de la position atteinte la **plus proche** occurrence de `X` qui précède strictement celle de `Y` . La variable $ contient alors le **plus court facteur** (**$**hortest) de `W` strictement compris entre `X` et `Y` (`W = U.X.$.Y`)[1]. L'opérateur `<X$Y>` retire `X.$.Y` de `W` et donne une valeur à $. Une fois qu'il a été appliqué, la séquence qui reste dans la pile est `U`, et la variable $ peut être utilisée dans le reste du chemin.

- `<X£Y>` : On recherche à la fin du mot `W` le suffixe `Y`. Puis, on recherche à partir de la position atteinte l'occurrence de `X` la **plus à gauche** qui précède strictement celle de

1. Le point représente ici l'opération de concaténation.

`Y`. La variable £ contient alors le **plus long facteur** (**£**ongest) de `W` strictement compris entre `X` et `Y` (`W = U.X.£.Y`).

- `<X>` : Si aucune variable n'est présente, on recherche `X` comme suffixe de `W` (`W =U.X`).
- `<$Y>` : Si le facteur `X` est absent, le **plus court facteur** `$` est la première lettre qui précède strictement `Y` .
- `<£Y>` : Si le facteur `X` est absent, le **plus long facteur** `£` est le préfixe de `W` tel que `W = £.Y`.

Pour illustrer l'utilisation des ces opérateurs, considérons le verbe *reprendre* :

Verbe	Opérateur	Variable	Résultat
reprendre	<re>		reprend
reprendre	<$>	$ = e	reprendr
reprendre	<£>	£= reprendre	ε
reprendre	<re$re>	$ = nd	rep
reprendre	<re£re>	£ = prend	
reprendre	<$re>	$ = d	repren
reprendre	<re$>	$ = ε	reprendre
reprendre	<£re>	£ = reprend	ε
reprendre	<re£>	£ = prendre	re

Le programme MultiFlex permet d'utiliser dix variables de type $ dont les noms sont $, $1..., $9 et dix variables de type £ dont les noms sont £, £1..., £9. De plus, plusieurs variables de types différents peuvent être utilisées au sein d'une même opération. Ainsi l'opérateur <£3re$7re> appliqué au verbe *reprendre* donne £3 = rep et $7 = `nd`.

Si l'on considère les verbes `accélérer`, `sécher`, la deuxième personne du présent de l'indicatif peut être générée par l'opération <é$er>è$es :

```
accélérer   <é$er>  →  accél  $ = r    +  è$es  →  accélères
sécher      <é$er>  →  s      $ = ch   +  è$es  →  sèches
```

On remarque que le facteur `$` conservé dans la forme fléchie est de longueur variable (`r`, `ch`). La flexion de `accélérer` et `sécher` ne peut se faire que par des opérateurs de pile classiques à l'aide d'une opération commune. Deux opérations différentes (`-4RèCes`, `-5RèCes`) sont nécessaires. Le graphe de la figure 3.10 permet de fléchir des verbes comme `accélérer` et `sécher` au présent.

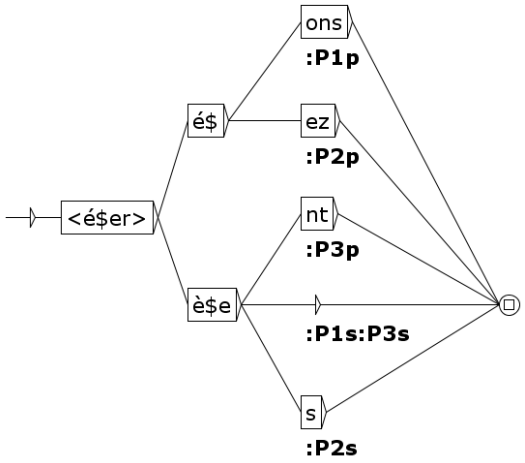


FIGURE 3.10 – Graphe de flexion pour des verbes comme *accélérer*, *sécher*

Voici les flexions obtenues pour les verbes `accélérer` et `sécher` :

```
accélère,accélérer.V:P1s:P3s
accélèrent,accélérer.V:P3p
accélères,accélérer.V:P2s
accélérez,accélérer.V:P2p
accélérons,accélérer.V:P1p
sèche,sécher.V:P1s:P3s
sèchent,sécher.V:P3p
sèches,sécher.V:P2s
séchez,sécher.V:P2p
séchons,sécher.V:P1p
```

Le redoublement de certaines lettres lors de la flexion peut s'effectuer avec l'opérateur $. Par exemple l'adject *tranquil* en anglais possède deux formes au comparatif et deux au superlatf. Le graphe de la figure 3.11 permet de les produire.

Voici les flexions obtenues pour l'adjectif anglais `tranquil` :

Dans certaines langues, certaines formes fléchies comporte un préfixe qui s'ajoute devant la racine. C'est le cas lors de la formation du participe passé en allemand. L'utilisation conjointe des opérateurs `£` et `$` permet de fléchir le verbe allemand `sprechen` (parler) au présent et participe passé comme le montre le graphe de la figure 3.12.

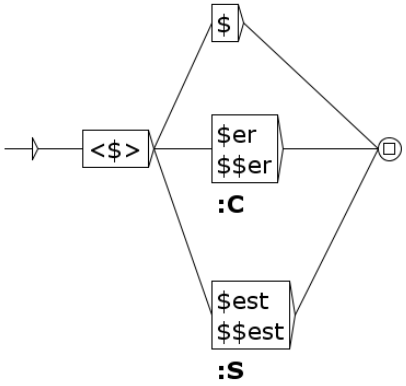


FIGURE 3.11 – Graphe de flexion pour des adjectifs anglais comme *tranquil*

```
tranquil,tranquil.A
tranquiler,tranquil.A:C
tranquilest,tranquil.A:S
tranquiller,tranquil.A:C
tranquillest,tranquil.A:S
```

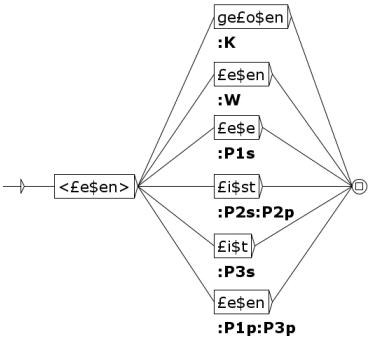


FIGURE 3.12 – Graphe de flexion pour des verbes comme *sprechen*

Voici les flexions obtenues pour le verbe allemand `sprechen` :

```
gesprochen,sprechen.V:K
spreche,sprechen.V:P1s
sprechen,sprechen.V:P1p:P3p:W
sprichst,sprechen.V:P2p:P2s
spricht,sprechen.V:P3s
```

Si l'on veut fléchir le verbe à particule aussprechen on peut utiliser deux variables de type $. Le figure 3.13 montre un graphe qui comport les variables `$1` et `$2`.

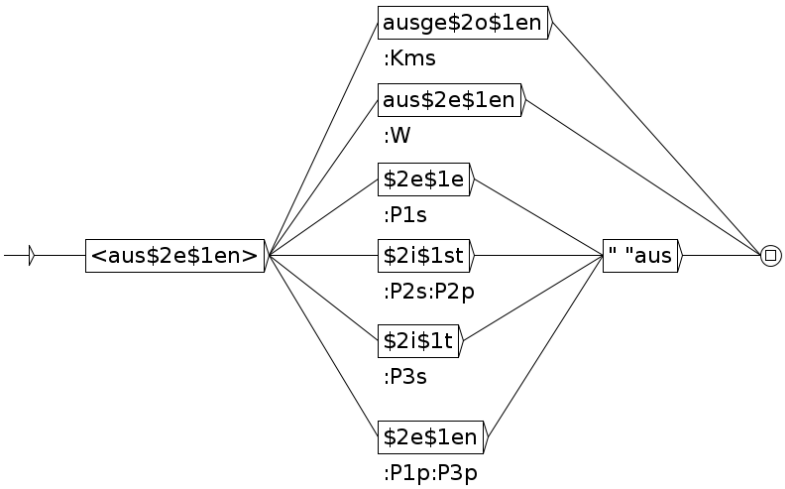


FIGURE 3.13 – Graphe de flexion pour des verbes comme *aussprechen*

Voici les flexions obtenues pour le verbe allemand `aussprechen` :

```
ausgesprochen,aussprechen.V:Kms
aussprechen,aussprechen.V:W
spreche aus,aussprechen.V:P1s
sprichst aus,aussprechen.V:P2p
sprichst aus,aussprechen.V:P2s
spricht aus,aussprechen.V:P3s
sprechen aus,aussprechen.V:P3p
sprechen aus,aussprechen.V:P1p
```

Codes sémantiques Dans certaines langues, il existe des caractéristiques flexionnelles qui correspondent en fait à des caractéristiques sémantiques comme par exemple les marqueurs de la forme passive. Ces codes peuvent ne pas apparaître comme des codes flexionnels, mais plutôt comme des codes sémantiques. Pour produire des codes sémantiques, il faut insérer un signe plus au début de la sortie d'une boîte. Cette boîte doit seulement contenir le code sémantique précédé d'un plus, comme le montre la figure 3.14.

3.5.3 Flexion des mots composés

Voir chapitre 11.

3.5.4 Flexion des langues sémitiques

Les langues sémitiques comme l'arabe ou l'hébreu se fléchissent d'une manière qui n'est pas facilement représentable avec les opérateurs de flexion décrits ci-dessus. Leur morphologie obéit à une logique différente : les mots se fléchissent selon un *squelette consonantique*.

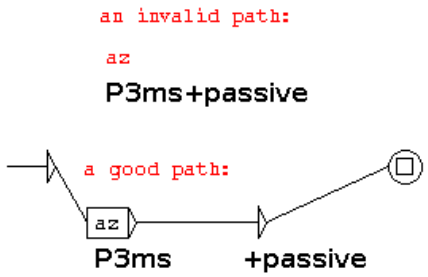


FIGURE 3.14 – Une grammaire de flexion avec un code sémantique

Le processus de flexion combine ce squelette avec des voyelles. Des opérateurs spécifiques ont été implémentés pour les langues sémitiques, et certains pourraient être utiles aussi pour des langues en dehors de la famille sémitique, comme le tagalog.

Tout d'abord, voyons un cas où on ne code que les consonnes dans le champ lemme de l'entrée DELAS :

```
ktb,$V31-123
```

Le signe `$` avant le code grammatical indique que la grammaire de flexion est en mode sémitique, et la forme `ktb` qui figure dans le champ lemme est le squelette consonantique. La figure 3.15 montre une grammaire jouet `V31-123.grf` qui illustre le fonctionnement du mode sémitique. Les grammaires de flexion utilisent la translittération Buckwalter++ de l'écriture arabe (cf. section 3.6).

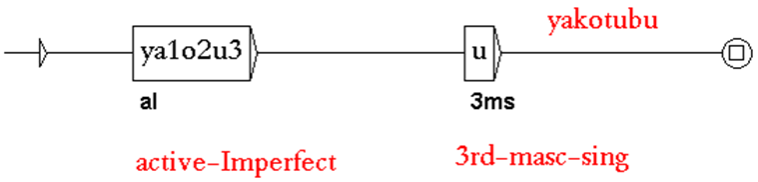


FIGURE 3.15 – Une grammaire de flexion jouet en mode sémitique

Le mode sémitique obéit aux règles suivantes :

1. Tous les opérateurs de flexion standard peuvent être utilisés (`L`, `R`, etc.).

2. Un chiffre représente une lettre du champ lemme (`1` pour la première, `2` pour la seconde, etc). Dans notre exemple, `1`, `2` et `3` représentent respectivement `k`, `t` et `b`. Si on veut désigner une lettre après la neuvième, on doit protéger son numéro avec des chevrons : `<10>`.

Le DELAF produit par cette grammaire est :

```
yakotubu,ktb.V:aI3ms
```

Si on ne code que les consonnes dans le champ lemme et que deux entrées ont les mêmes consonnes mais diffèrent par leurs voyelles, on doit coder les voyelles dans les grammaires de flexion :

```
Hsb,$V3au // compter, Hasaba, yaHosubu
Hsb,$V3ii // penser, Hasiba, yaHosibu
```

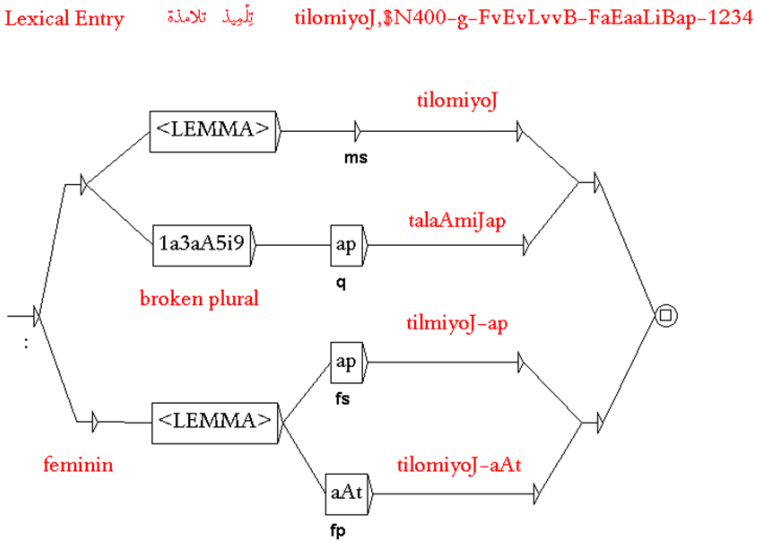


FIGURE 3.16 – Une grammaire de flexion en mode sémitique avec l'opérateur <LEMMA>

Pour copier tout le champ lemme, on peut utiliser l'opérateur <LEMMA>. Une boite contenant cet opérateur récupère tout le champ lemme mais ne dépend pas du nombre de lettres. Cet opérateur est utile pour les noms et adjectifs arabes pour lesquels les formes du masculin sont obtenues en insérant des voyelles dans le squelette consonantique, alors que celles du féminin le sont en ajoutant des suffixes (figure 3.16). Dans cet exemple, on a codé à la fois les consonnes et les voyelles dans le champ lemme.

L'opérateur <n.LEMMA> copie le lemme depuis la $n$ième position jusqu'à la fin. Par exemple, dans certains noms arabes, la voyelle brève de la première syllabe alterne : `a/u`, `a/i` ou `u/i`, comme dans `nufaAyap/nifaAyap` "ordures". La grammaire de flexion de la fig. 3.17 produit à la fois les variantes en `u` et en `i` comme formes fléchies de `nufaAyap`.

En tagalog, une langue austronésienne parlée aux Philippines et qui utilise communément des infixes et des redoublements pour la flexion, <LEMMA> et <n.LEMMA> peuvent être utiles pour produire des temps verbaux. La grammaire de flexion jouet de la fig. 3.18 produit le parfait `kumain`, le futur `kakain` et l'imparfait `kumakain` du verbe `kain` "manger".

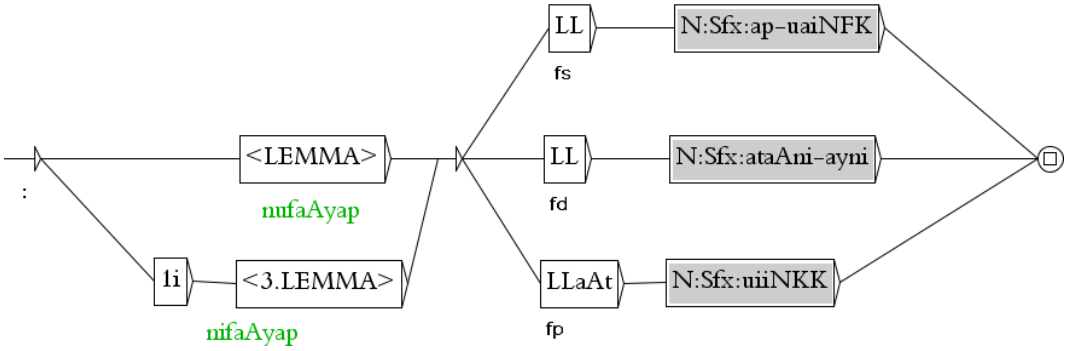


FIGURE 3.17 – Une grammaire de flexion en mode sémitique avec l'opérateur <n.LEMMA>

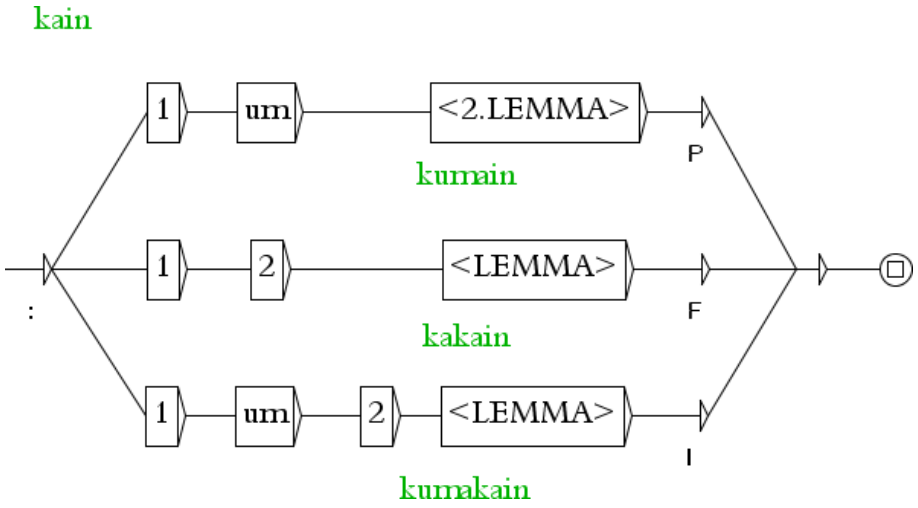


FIGURE 3.18 – Une grammaire de flexion jouet pour le tagalog en mode sémitique

3.6 Translittération des dictionnaires d'arabe

Quand les linguistes arabes analysent des dictionnaires pour y détecter des erreurs, la lecture dans l'écriture arabe est simple et efficace. Cependant, quand ils créent des grammaires de flexion (section 3.5), des éléments de mots arabes apparaissent dans la même boite que des informations morpho-syntaxiques codées dans l'alphabet latin, et dans ce contexte, les allers-retours entre l'écriture arabe, qui se lit de droite à gauche, et l'alphabet latin, qui se lit de gauche à droite, ne sont pas pratiques. Avec Unitex, on peut coder des grammaires de flexion entièrement dans l'alphabet latin, en utilisant la translittération Buckwalter++, une correspondance biunivoque entre un codage Unicode de l'écriture arabe et des lettres de l'alphabet latin (cf. [74], section 3.2, pp. 4–6). La translittération Buckwalter++ est définie par la table des figures 3.19 et 3.20. Unitex offre une fonctionnalité de translittération de dictionnaires DELAS et DELAF de l'arabe vers le code Buckwalter++ et inversement (fig. 3.21). Cette fonctionnalité est accessible par le menu DELA.

Unicode	Nom de la lettre	Arabe	Buckwalter	Buckwalter++
0621	HAMZA ON THE LINE	ء	'	c
0622	ALEF WITH MADDA ABOVE	آ	|	C
0623	ALEF WITH HAMZA ABOVE	أ	>	O
0624	WAW WITH HAMZA ABOVE	ؤ	&	W
0625	ALEF WITH HAMZA BELOW	إ	<	I
0626	YEH WITH HAMZA ABOVE	ئ	}	e
0627	ALEF	ا	A	
0628	BEH	ب	b	
0629	TEH MARBUTA	ة	p	
062A	TEH	ت	t	
062B	THEH	ث	v	
062C	JEEM	ج	j	
062D	HAH	ح	H	
062E	KHAH	خ	x	
062F	DAL	د	d	
0630	THAL	ذ	*	J
0631	REH	ر	r	
0632	ZAIN	ز	z	
0633	SEEN	س	s	
0634	SHEEN	ش	$	M
0635	SAD	ص	S	
0636	DAD	ض	D	
0637	TAH	ط	T	

FIGURE 3.19 – Table de translittération Buckwalter++, première moitié

Unicode	Nom de la lettre	Arabe	Buckwalter	Buckwalter++
0638	ZAH	ظ	Z	
0639	AIN	ع	E	
063A	GHAIN	غ	g	
0641	FEH	ف	f	
0642	QAF	ق	q	
0643	KAF	ك	k	
0644	LAM	ل	l	
0645	MEEM	م	m	
0646	NOON	ن	n	
0647	HEH	ه	h	
0648	WAW	و	w	
064A	ALEF MAKSURA	ي	Y	
0649	YEH	ى	y	
064B	FATHATAN		F	
064C	DAMMATAN		N	
064D	KASRATAN		K	
064E	FATHA		a	
064F	DAMMA		u	
0650	KASRA		i	
0651	SHADDA		~	G
0652	SUKUN		o	
0670	SUPERSCRIPT ALEF		`	R
0671	ALEF WASLA	ٱ	{	L

FIGURE 3.20 – Table de translittération Buckwalter++, deuxième moitié

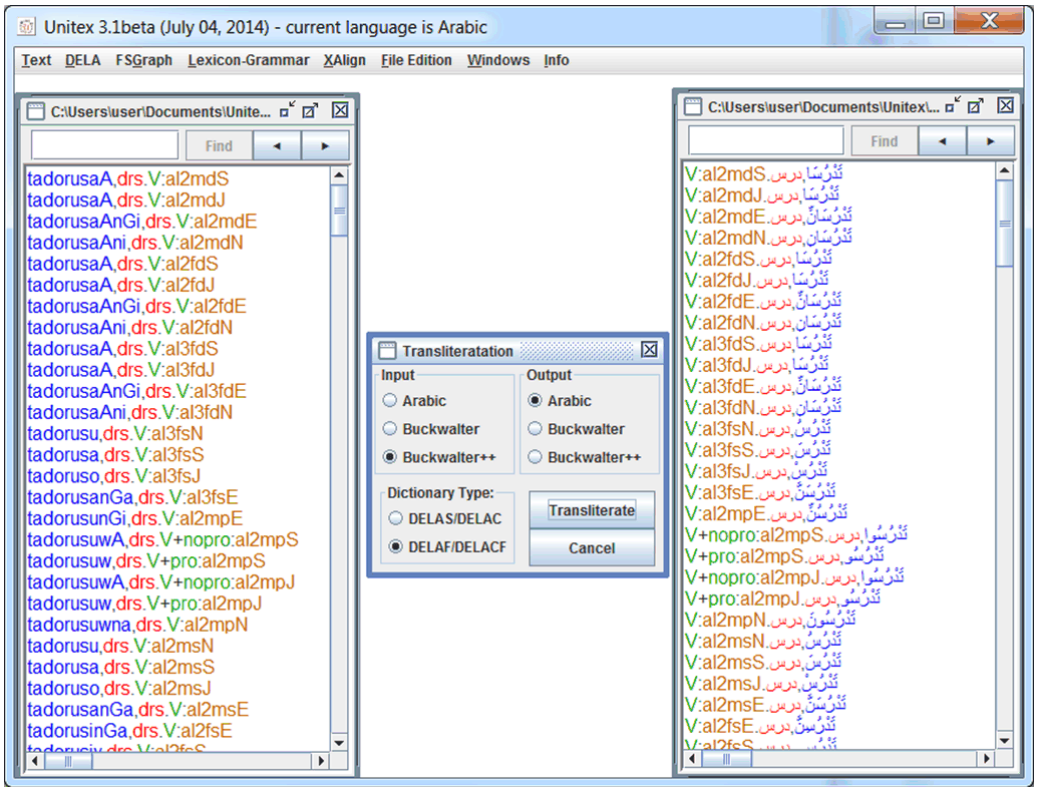


FIGURE 3.21 – Translittération d'un dictionnaire DELAF depuis le code Buckwalter++ (à gauche) vers l'écriture arabe (à droite)

3.7 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 dictionnaire ;
- `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 3.22.

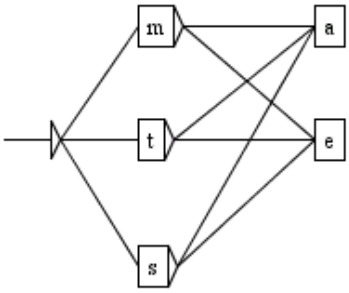


FIGURE 3.22 – Représentation d'un exemple d'automate minimal

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 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 3.23 montre le résultat de la compression d'un dictionnaire de mots simples.

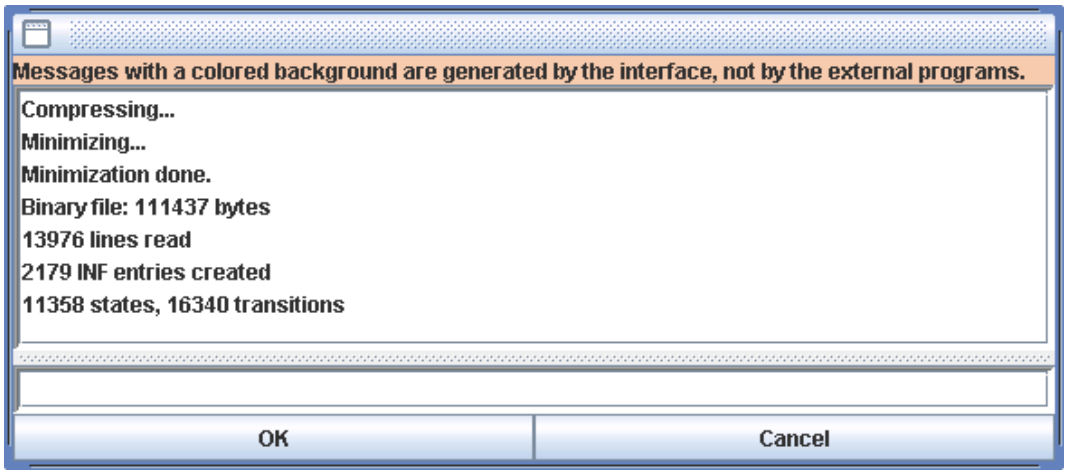


FIGURE 3.23 – Résultat d'une compression

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.

Quand le mode sémitique (3.5.4) a beaucoup été utilisé lors de la flexion d'un dictionnaire, une variante spécifique de l'algorithme de compression peut réduire la taille des fichiers `.bin` et `.inf`. Pour l'invoquer, soit on déclare la langue comme étant sémitique dans les préférences globales, en cochant l'option "Semitic language" dans "Preferences > Language

and Presentation", soit on lance le programme Compress en ligne de commande avec l'option `--semitic`.

3.8 Application de dictionnaires

Unitex peut manipuler soit des dictionnaires compressés (`.bin`) soit des graphes-dictionnaires (`.fst2`). Ces dictionnaires peuvent être appliqués soit lors du prétraitement, soit explicitement en cliquant sur "Apply Lexical Resources..." dans le menu "Text". Nous allons maintenant détailler les règles de l'application des dictionnaires. Le cas des graphes-dictionnaires sera abordé dans la section 3.8.3.

3.8.1 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 dictionnnaire 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 ctr+.bin cities-.bin rivers.bin regions-.bin
```

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

1. `cities-.bin`
2. `regions-.bin`
3. `rivers.bin`
4. `ctr+.bin`

3.8.2 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 majucules 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.

3.8.3 Graphes-dictionnaires

Le programme `Dico` est également capable d'appliquer des graphes-dictionnaires. Un graphe-dictionnaire est un graphe qui sert de dictionnaire. Les graphes-dictionnaires respectent, par défaut[2], la règle suivante : si on les applique avec le programme `Locate` en mode MERGE, ils doivent produire des séquences correspondant à des lignes de DELAF. Quand on les applique à un texte, ils attachent les étiquettes lexicales DELAF à ces séquences.

La figure 3.24 montre un graphe reconnaissant les symboles chimiques. On peut voir sur cette figure un premier avantage par rapport aux dictionnaires compressés : l'utilisation des guillemets permet de forcer le respect de la casse. Ainsi, ce graphe reconnaîtra bien `Fe` mais pas `FE`, alors qu'il est impossible de spécifier une telle interdiction dans un DELAF usuel.

Pour faire un graphe-dictionaire, on utilise les outils pour les graphes (menu FSGraph, section 5.2) mais on les sauvegarde et on les compile de préférence dans le répertoire pour les dictionnaires (le répertoire Dela). Pour appliquer un graphe-dictionnaire à un texte, on utilise un outil pour les dictionnaires : "Text > Apply lexical resources" (section 2.5.5).

Un autre avantage des graphes-dictionnaires est qu'ils peuvent exploiter les résultats fournis par les dictionnaires appliqués précédemment. Ainsi, on peut appliquer le dictionnaire général, puis étiqueter comme noms propres les mots inconnus commençant par une majuscule à l'aide du graphe `NPr+` de la figure 3.25. Le `+` dans le nom du graphe lui donne une priorité basse afin qu'il soit appliqué après le dictionnaire général. Pour fonctionner, ce graphe se base sur les mots qui sont toujours inconnus après le passage du dictionnaire général. Les crochets correspondent à une définition de contexte (voir la section 6.3).

2. Les graphes-dictionnaires morphologiques sont une exception (section 3.8.4) .

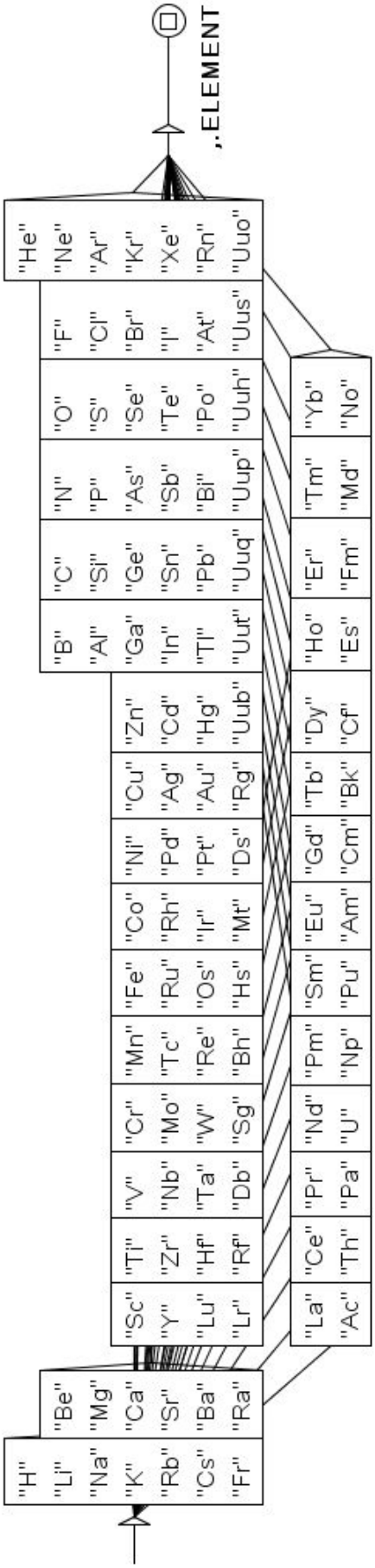


FIGURE 3.24 – Graphe-dictionnaire des éléments chimiques

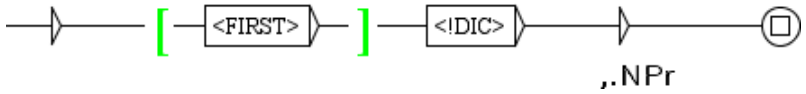


FIGURE 3.25 – Graphe-dictionnaire étiquetant comme noms propres les mots inconnus commençant par une majuscule

Comme les graphes-dictionnaires sont appliqués par le moteur du programme `Locate`, ils peuvent utiliser tout ce que le programme `Locate` autorise. En particulier, il est possible d'utiliser les filtres morphologiques (section 4.7) et le mode morphologique (section 6.4). Ainsi, le graphe de la figure 3.26 utilise ces filtres pour reconnaître les nombres en chiffres romains. Notons qu'il utilise également des contextes afin d'éviter, par exemple, que `C` ne soit pris comme chiffre romain quand il est suivi par une apostrophe.

Par défaut, les graphes-dictionnaires sont appliqués en mode MERGE. Il est possible de les appliquer en mode REPLACE, en ajoutant à leur le nom le suffixe `-r`. Celui-ci se combine avec les priorités + et – :

```
bagpipe-r.fst2  McAdam-r-.fst2  phtirius-r+.fst2
```

Exporter les entrées produites comme dictionnaire du mode morphologique

Les entrées produites par un graphe-dictionnaire sont consultées par le programme `Locate` quand il rencontre des masques lexicaux qui nécessitent la consultation d'un dictionnaire.

Cependant, cette fonctionnalité est restreinte quand le masque lexical est en mode morphologique (cf. section 6.4). On ne peut pas déclarer un graphe-dictionnaire comme dictionnaire du mode morphologique de la manière habituelle (cf. section 6.4.3), car ce n'est pas un fichier `.bin`. Quand on est en mode morphologique, les masques lexicaux qui nécessitent la consultation d'un dictionnaire ne déclenchent pas la consultation de graphes-dictionnaires. En compensation, on dispose de plusieurs solutions.

- On peut envisager d'invoquer le graphe-dictionnaire depuis la partie du graphe qui est en mode morphologique.

- Unitex produit de façon interne un dictionnaire des formes reconnues dans le texte par un graphe-dictionnaire. Si le nom du graphe-dictionnaire contient l'option `b` (voir ci-dessous Conventions de nommage), ce dictionnaire produit automatiquement est inclus implicitement parmi les dictionnaires du mode morphologique, de telle sorte qu'il est consulté quand le programme `Locate` rencontre des masques lexicaux en mode morphologique. Mais cette solution ne fonctionne que pour les formes reconnues par le graphe-dictionnaire pendant l'application initiale des dictionnaires (cf. section 3.8), et non pour celles qui n'apparaissent dans le texte que comme parties de tokens.

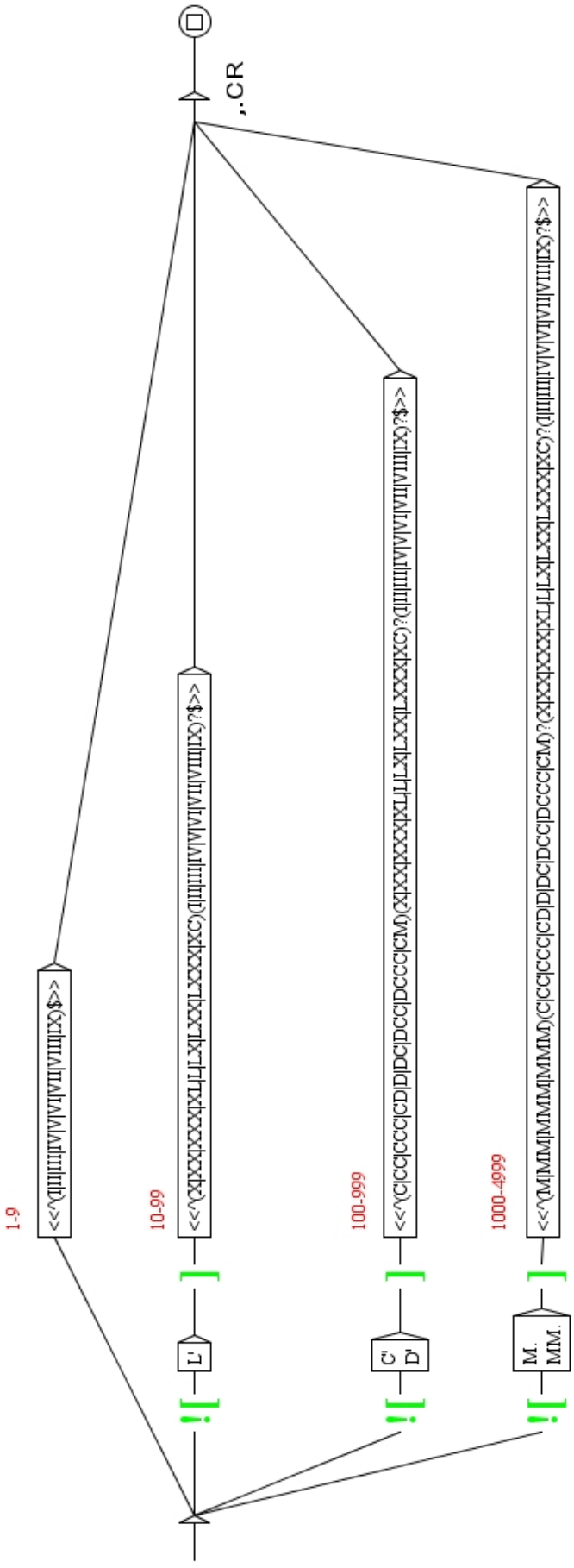


FIGURE 3.26 – Graphe-dictionnaire reconnaissant les nombres en chiffres romains

Si on ajoute `z` à la place de `b`, le dictionnaire produit de façon interne pour le texte est immédiatement compressé, et il peut être consulté quand d'autres graphes-dictionnaires sont appliqués par la suite.

Conventions de nommage

Le processus de nommage d'un graphe-dictionnaire s'établit comme suit :

```
nom(-XYZ)([-+]).fst2
```

où :

- `X` prend l'une des valeurs `[rRmM]` : `r` signifie mode REPLACE ; `M` signifie mode MERGE (mode par défaut) ;
- `Y` prend l'une des valeurs `[bBzZ]` : option qui régit la construction d'un dictionnaire du mode morphologique (voir ci-dessus) ;
- `Z` prend l'une des valeurs `[aAlLsS]` : `a` signifie que le graphe est appliqué en mode "All matches" ; `l` signifie mode "Longest matches" (mode par défault) ; `s` signifie "Shortest matches".

3.8.4 Graphe-dictionnaire morphologique

Dans un graphe-dictionnaire, chaque chemin doit, par défaut, produire une entrée lexicale à inclure dans le dictionnaire du texte. Dans un graphe-dictionnaire morphologique, chaque chemin doit produire une séquence d'une ou plusieurs étiquettes délimitées par des accolades et conformes à la syntaxe des lignes du DELAF (section 3.1.1). Les sorties de tels graphes seront utilisées comme entrées pour construire l'automate du texte. Nous les appelons "graphes-dictionnaires morphologiques" parce que leur principale utilité est de fournir de nouvelles analyses morphologiques dans l'automate du texte, grâce au mode morphologique (voir 6.4). Cette fonctionnalité est utile pour des langues agglutinantes comme le coréen. Pour pouvoir utiliser un graphe comme graphe-dictionnaire morphologique, on le déclare par une barre oblique (slash, /) comme premier caractère de sa sortie, comme dans la figure 3.27.

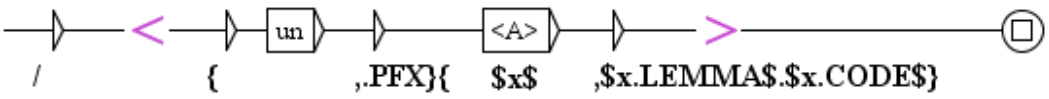


FIGURE 3.27 – Exemple de graphe-dictionnaire morphologique

La règle est simple : toute sortie du graphe-dictionnaire commençant par une barre oblique (slash, /) est ajoutée au fichier `tags.ind`, situé dans le répertoire du texte. Ce fichier est utilisé par le programme `Txt2Fst2` afin d'ajouter des interprétations à l'automate du texte.

La grammaire de la figure 3.27 reconnaît des mots formés par le préfixe `un` suivi d'un adjectif. Si on l'applique comme graphe-dictionnaire, on obtient de nouveaux chemins dans l'automate du texte comme le montre la figure 3.28. Remarquons que lorsque deux tags correspondent à des analyses dans la même unité lexicale, le lien entre eux est affiché par une ligne discontinue.

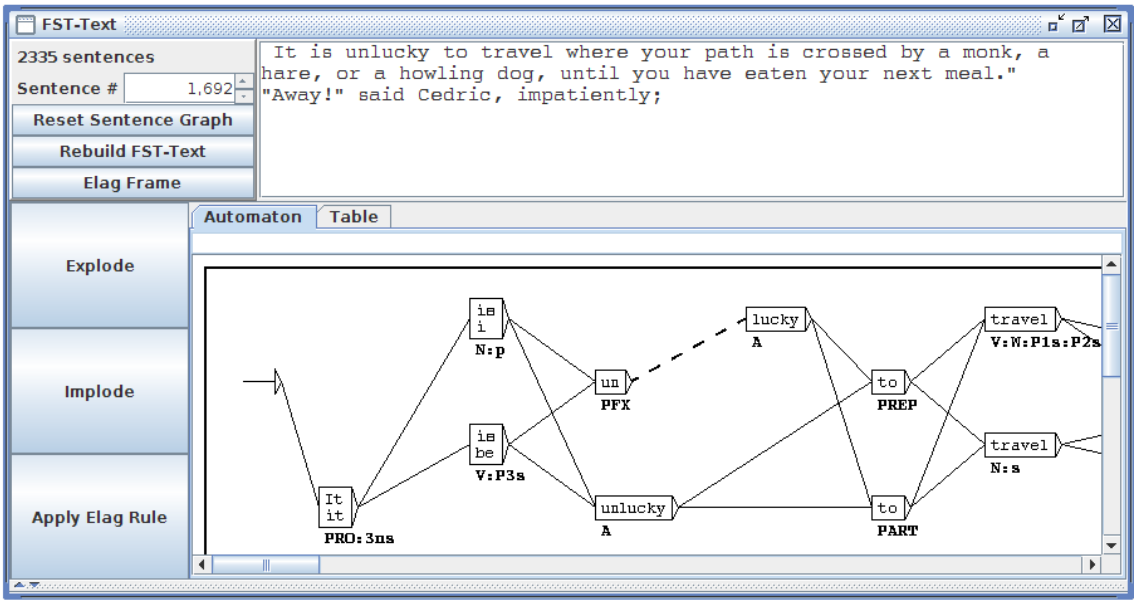


FIGURE 3.28 – Chemin ajouté par un graphe-dictionnaire morphologique

3.8.5 Tolérance à l'omission, à la substitution et à l'insertion de lettres

Pour l'arabe et d'autres langues, la reconnaissance des mots peut tolérer certaines différences entre les entrées de dictionnaire et les séquences présentes dans le texte. En arabe, certaines lettres, surtout des voyelles brèves, sont généralement omises à l'écrit [3]. Ces variations typographiques sont facultatives mais suivent des règles spécifiques.

Si le dictionnaire d'arabe est entièrement voyellé, Unitex peut traiter des mots non voyellés, partiellement voyellés ou entièrement voyellés. Si un mot contient une ou plusieurs voyelles explicites, la procédure de consultation ne retient dans le dictionnaire que les mots qui ont ces mêmes voyelles aux mêmes positions.

Unitex donne la possibilité de paramétrer la consultation du dictionnaire à l'aide d'un fichier de configuration qui spécifie quelles variations typographiques sont admises (section 15.13.9). Ce fichier est constitué de lignes de cette forme :

3. D'autres lettres peuvent être remplacées, par exemple une lettre qui représente l'occlusive glottale (*hamza*) et *a* long (*alif*) peut être remplacée par la lettre qui représente *a* long. Une interversion de lettres adjacentes et une insertion sont aussi acceptées dans l'usage.

```
fatha omission=YES
```

où `fatha omission` est le nom d'une règle. Les données distribuées avec Unitex fournissent ce fichier avec 26 règles prédéfinies, mais on peut les changer en remplaçant `YES` par `NO`. Les règles prédéfinies sont conçues pour être utilisées avec un dictionnaire entièrement voyellé. Voici des exemples, avec la translittération Buckwalter++ (cf. fig. 3.19 et 3.20) :

- Omission d'une lettre

Nom de la règle	Signification	Exemple (forme dans le dictionnaire → forme permise)
fatha omission	*a* → <E>	*kitaAbN* → *kitAbN*
dammatan omission at end	*N* final → <E>	*kitaAbN* → *kitaAb*

- Omission de deux lettres adjacentes

shadda fatha omission	*Ga* → <E>	*katGaba* → *katba*
shadda dammatan omission at end	*GN* final → <E>	*ruwsiyGN* → *ruwsiy*

- Interversion de deux lettres adjacentes

fathatan alef equiv alef fathatan	*FA* final → *AF*	*kitabFA* → *kitabAF*

- Substitution

alef hamza above O	*O* initial → *A*	*Oakala* → *Aakala*

- Insertion

solar assimilation	*Alt* → *AltG*	*AltaAniy* → *AltGaAniy*

et de même pour 14 autres consonnes en plus de *t*.

3.9 Bibliographie

Le tableau 3.9 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://unitexgramlab.org/language-resources

Langue	**Mots simples**	**Mots composés**
English	[58], [72]	[15], [88]
French	[19], [20], [63]	[20], [38], [90], [46]
Modern Greek	[2], [17], [60]	[61], [62]
Italian	[28], [29]	[94]
Spanish	[8]	[7]
Portuguese	[25], [86], [83], [73]	[82], [83]

TABLE 3.9 – 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, ou expression régulière, peut-être :

- une unité lexicale (`livre`) ou un masque lexical (`<manger.V>`);
- une position particulière du texte : le début `{^}` ou la fin `{$}`
- 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, l'unité lexicale a la même définition qu'en 2.5.4 (page 36). Notons que les symboles point, plus, étoile, inférieur ainsi que les parenthèses ouvrantes et fermantes ont une signification particulière, il faut donc les déspécialiser avec le caractère `\` si l'on souhaite les rechercher. Voici quelques exemples d'unités lexicales valides :

```
chat
\.
<N:ms>
{S}
```

Par défaut, Unitex tolère que des mots 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 Masques lexicaux

Un masque lexical est une requête qui reconnaît une unité lexicale ou une suite d'unités lexicales.

4.3.1 Symboles spéciaux

Il y a deux sortes de masques lexicaux. La première catégorie regroupe les symboles spéciaux ou méta-symboles présentés dans la section 2.5.2, sauf <PNC> et <^>. (Le symbole <PNC>, qui reconnaît des signes de ponctuation, n'est valide que pendant le prétraitement ; <^> reconnaît un retour à ligne, mais 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.) Les méta-symboles utilisables pour rechercher des motifs dans un texte sont les suivants :

- <E> : mot vide, ou epsilon. Reconnaît la séquence vide ;
- <TOKEN> : reconnaît n'importe quelle unité lexicale sauf l'espace utilisé par défaut pour les filtres morphologiques ;
- <WORD> : reconnaît n'importe quelle unité lexicale formée de lettres ;
- <LOWER> : reconnaît n'importe quelle unité lexicale formée de lettres minuscules ;
- <UPPER> : reconnaît n'importe quelle unité lexicale formée de lettres majuscules ;
- <FIRST> : 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 ;
- <TDIC> : reconnaît n'importe quelle unité lexicale tagguée comme {XXX,XXX.XXX} ;
- <NB> : reconnaît n'importe quelle suite de chiffres contigus (1234 est reconnue mais pas 1 234) ;
- # : interdit la présence de l'espace.

Les anciens codes correspondant à <WORD>, <LOWER>, <UPPER> et <FIRST> étaient respectivement <MOT>, <MIN>, <MAJ> et <PRE>. Ils restent opérationnels afin de conserver la compatibilité descendante du système avec les graphes existants. Même s'il n'est pas prévu de supprimer ces codes, on recommande de les éviter dans les graphes conçus pour fonctionner avec les versions plus récentes [1], pour ne pas faire augmenter inutilement le nombre de masques lexicaux en usage.

NOTE : comme il a été dit en section 2.5.4, AUCUN des métas ne peut être utilisé pour reconnaître le marqueur {STOP}, pas même <TOKEN>.

4.3.2 Référence aux informations fournies par les dictionnaires

La seconde sorte de masques lexicaux 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. On remarque que cette forme est ambiguë si lire est aussi un code grammatical ou sémantique ;
- <lire.> : reconnaît toutes les entrées qui ont lire comme forme canonique. Ce masque lexical n'est pas ambigu avec le précédent ;
- <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. Ce masque lexical est ambigu comme le premier. Pour lever l'ambiguïté, on peut utiliser <.V> ou <+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 masque 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 masque reconnaît la même chose que la simple unité lexicale lirons.

4.3.3 Contraintes grammaticales et sémantiques

Les masques lexicaux des exemples précédents sont simples. Il est possible d'exprimer des motifs plus complexes en indiquant plusieurs codes grammaticaux ou sémantiques, séparés par le caractère +. Si plusieurs codes sont présents, le caractère + est interprété comme ''et'' : une entrée de dictionnaire ne sera alors reconnue que si elle possède tous les codes présents dans le masque. Le masque <N+z1> reconnaît ainsi les entrées :

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

1. À partir de la version 3.1bêta, révision 4072 du 2 octobre 2015.

mais pas :

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

On peut exclure des codes en les faisant précéder du caractère ~ au lieu de +. Pour être reconnue, une entrée doit contenir tous les codes exigés par le masque, sans aucun des codes qu'il interdit. Par exemple, <A~z3> reconnaît toutes les entrées qui ont le code A sans le code z3 (cf. table 3.2)[2]. Si on veut faire référence à un code contenant le caractère ~, on doit le déspécialiser en le faisant précéder d'un \.

REMARQUE : Avant la version 2.1, l'opérateur de négation était le signe moins. Si l'on veut utiliser d'anciens graphes sans les modifier, il faut appeler Locate en ligne de commande avec l'option -g minus.

La syntaxe des masques lexicaux ne fait aucune différence entre les codes grammaticaux (table 3.1) et sémantiques (table 3.2). Dans le format de dictionnaires électroniques DELAF, les codes grammaticaux sont ceux qui apparaissent en premier et codent la catégorie grammaticale, mais dans les masques lexicaux d'Unitex, l'ordre dans lequel apparaissent les codes grammaticaux et sémantiques n'a pas d'importance. Les trois masques lexicaux suivants sont équivalents :

```
<N~Hum+z1>
<z1+N~Hum>
<~Hum+z1+N>
```

Un masque lexical peut contenir un code sémantique sans code de catégorie grammaticale.

NOTE : il n'est pas possible d'utiliser un masque n'ayant que des codes d'interdiction. <~N> et <~A~z1> sont donc des masques incorrects. Il est toutefois possible d'exprimer de telles contraintes en utilisant des contextes (voir section 6.3).

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 suivent les mêmes conventions de format que les codes flexionnels présents dans les dictionnaires. Voici quelques exemples de masques lexicaux utilisant des contraintes flexionnelles :

- <A:m> reconnaît un adjectif au masculin ;
- <A:mp> reconnaît un adjectif au masculin pluriel.

2. Si les dictionnaires décrivent un mot par deux entrées dont une avec A+z3 et l'autre avec seulement A, ce mot est reconnu par <A+z3> à cause de la première entrée et par <A~z3> à cause de l'autre.

Un code flexionnel est introduit par le caractère : et constitué d'un ou plusieurs caractères, qui représentent une information chacun. Commençons par le cas simple d'entrées lexicales et de masques qui ont un seul code flexionnel. Pour qu'une entrée lexicale E soit reconnue par un masque M, il faut que le code flexionnel de E contienne tous les caractères du code flexionnel de M :

E=sépare,séparer.V:Y2s
M=<V:Y2>

Le code Y2s de E contient les caractères Y et 2. Le code Y2 est inclus dans au moins un code de E, le masque lexical M reconnaît donc l'entrée E.

L'ordre des caractères à l'intérieur d'un code flexionnel est sans importance. Tous les codes grammaticaux et sémantiques doivent précéder les codes flexionnels.

Si plusieurs codes flexionnels sont présents dans un masque lexical, le caractère : est interprété comme ''ou" :

- <A:mp:f> correspond à la fois à <A:mp>et à <A:f> ; il reconnaît un adjectif qui est soit au masculin pluriel, soit au féminin ;
- <V:2:3> reconnaît un verbe à la 2e ou à la 3e personne ; cela exclut tous les temps qui n'ont ni 2e ni 3e 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 masque M, il faut qu'au moins un code flexionnel de E contienne tous les caractères d'au moins un code flexionnel de M. Considérons l'exemple suivant :

E=sépare,séparer.V:W: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 masque lexical M reconnaît donc l'entrée E.

4.3.5 Négation d'un masque lexical

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 métas <WORD>, <LOWER>, <UPPER>, <FIRST>[3], <DIC> ainsi que sur les masques lexicaux 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 méta <!WORD> peut reconnaître toutes les unités lexicales qui ne sont pas formées de lettres, sauf le séparateur de phrases {S} et, bien sûr, le marqueur {STOP}. La négation est sans effet sur <NB>, <SDIC>, <CDIC>, <TDIC> et <TOKEN>.

3. Et sur leurs équivalents anciens <MOT>,<MIN>, <MAJ>, <PRE>. Voir section 4.3.1.

La négation est interprétée d'une façon particulière dans les métas <!DIC>, <!LOWER>, <!UPPER> et <!FIRST>[4]. Au lieu de reconnaître toutes les formes qui ne sont pas reconnues par le méta sans la négation, ces motifs ne donnent que des formes qui sont des séquences de lettres. Ainsi, le méta <!DIC> permet d'obtenir les mots inconnus du texte (cf. figure 4.1). Ces formes inconnues sont le plus souvent des noms propres, des néologismes et des fautes d'orthographe.

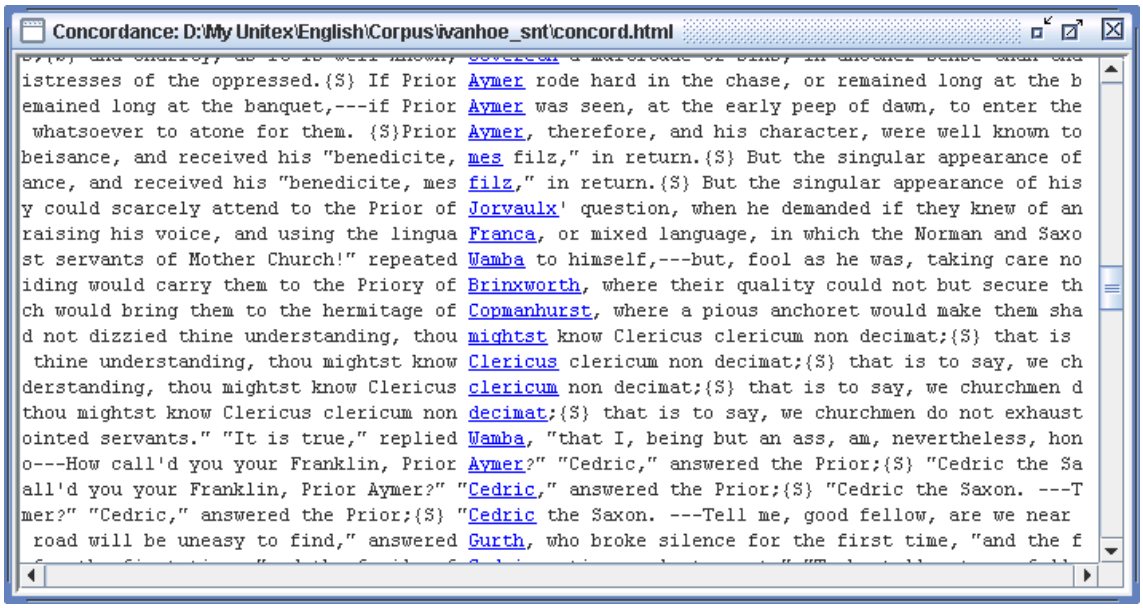


FIGURE 4.1 – Résultat de la recherche du méta <!DIC>

La négation d'un masque lexical comme <V:G> reconnaît tous les mots sauf ceux qui peuvent être reconnus par ce masque. Ainsi, le masque <!V:G> ne reconnaîtra pas la forme anglaise *being*, même s'il existe dans les dictionnaires du texte des entrées non verbales pour ce mot :

```
being,.A
being,.N+Abst:s
being,.N+Hum:s
```

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*) ;

4. Et dans leurs équivalents anciens <MIN>, <MAJ> et <PRE>. Voir section 4.3.1.

- `<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 ;
- `<!WORD>` : tous les caractères qui ne sont pas des lettres, sauf le séparateur de phrases (voir figure 4.2). Ce masque ne reconnait pas le séparateur de phrase `{S}` ni le tag `{STOP}`.

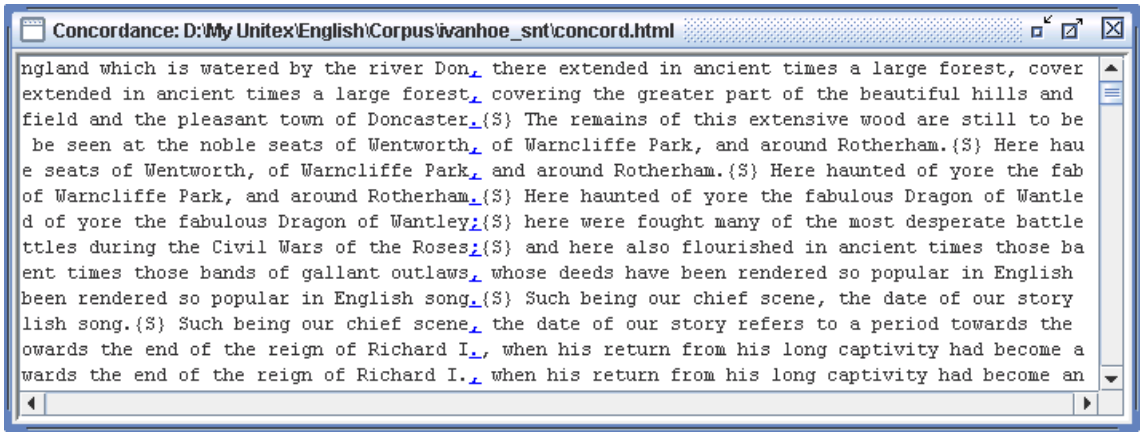


FIGURE 4.2 – Résultat de la recherche du méta `<!WORD>`

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
le<A>chat
```

reconnaît l'unité lexicale *le*, suivie d'un adjectif et de l'unité lexicale *chat*. 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 filtre entre doubles angles :

motif<<*motif morphologique*>>

Les filtres morphologiques s'expriment sous la forme d'expressions régulières au format POSIX (voir [65] pour une syntaxe détaillée). Voici quelques exemples de filtres élémentaires :

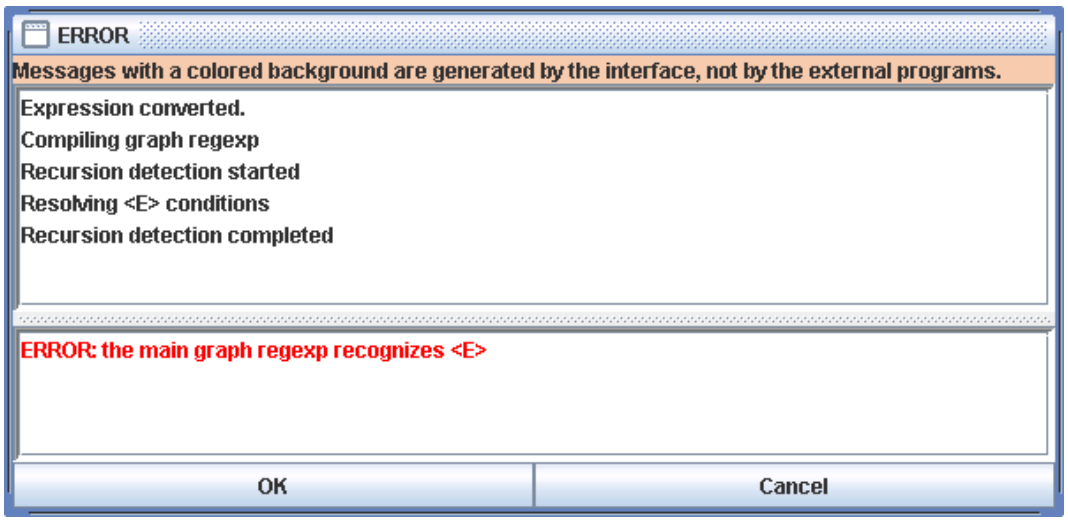


FIGURE 4.3 – Erreur lors de la recherche d'une expression reconnaissant le mot vide

- `<<ss>>` : contient `ss`
- `<<^a>>` : commence par `a`
- `<<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
- `<<es?>>` : contient `e` fsuivi par un `s` facultatif
- `<<ss[^e]?>>` : contient `ss` suivi par un caractère qui n'est pas une voyelle `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

- `<<^([^l]|l[^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`. De telles contraintes peuvent être exprimées plus simplement en utilisant des contextes (voir 6.3).

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

- `<V:K><<i$>>` : participe passé finissant par `i`
- `<CDIC><<->>` : mot composé contenant un tiret
- `<CDIC><< .* >>` : mot composé contenant deux espaces
- `<A:fs><<^pro>>` : adjectif féminin singulier commençant par `pro`
- `<DET><<^([^u]|(u[^n])|(un.+))>>` : déterminant différent de `un`
- `<!DIC><<es$>>` : 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 même variations de casse que les masques lexicaux. Ainsi, le filtre `<<^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 filtre, 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.

Le cadre "Locate Pattern" 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 For instance, if your grammar can recognize the sequences *very hot chili* and *very hot*, the first one will be discarded ;
- "Longest matches" : donne la priorité aux séquences les plus longues. C'est le mode utilisé par défaut ;

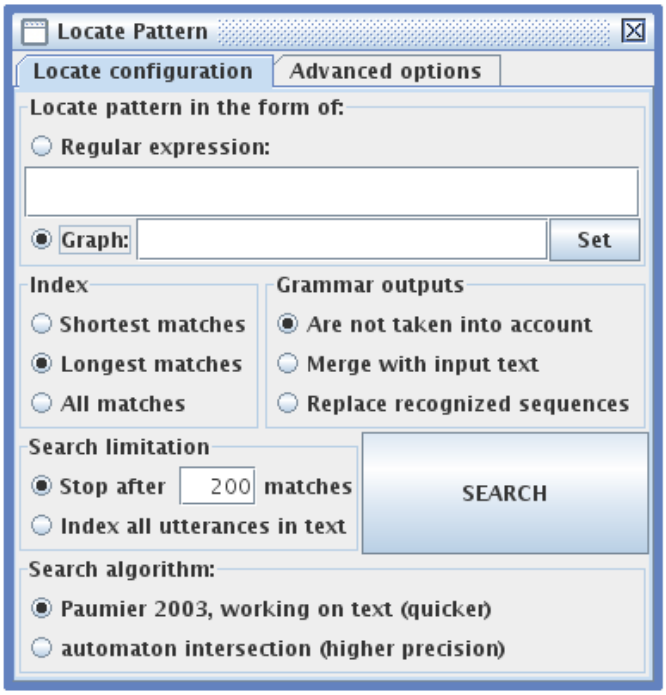


FIGURE 4.4 – Fenêtre de recherche d'expressions

- "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 6.10. De même pour les options de l'onglet "Advanced options" (voir section 6.10.2).

Dans le cadre "Search algorithm", on définit si l'on veut effectuer la recherche dans le texte avec le programme `Locate` ou dans l'automate du texte avec le programme `LocateTfst`. Par défaut la recherche est effectuée avec le programme `Locate`. Pour utiliser `LocateTfst`, il est utile de se référer à la section 7.7.

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.

FIGURE 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 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 de 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 (Firefox [11], Netscape [12], Internet Explorer, etc.).

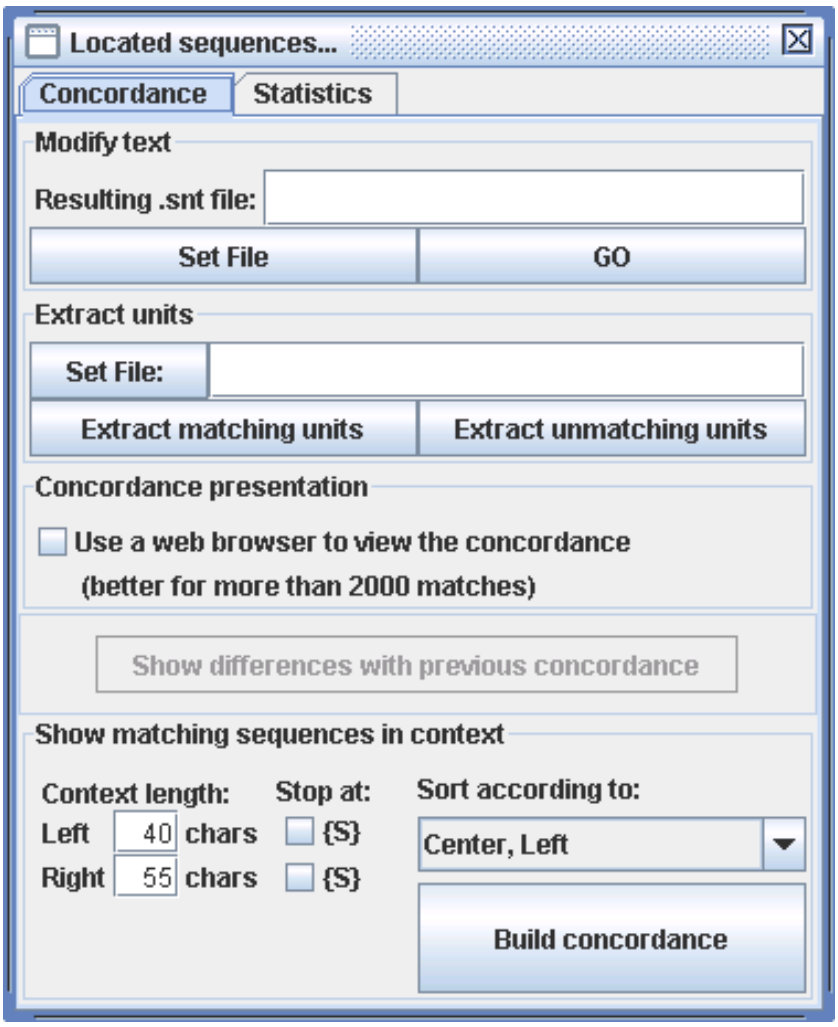


FIGURE 4.6 – Configuration de l'affichage des occurrences trouvées

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 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.

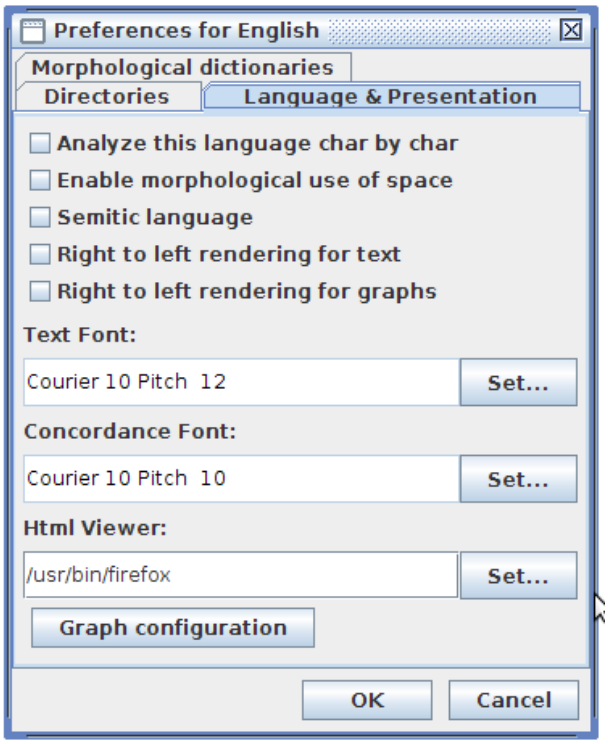


FIGURE 4.7 – Sélection d'un navigateur pour l'affichage des concordances

FIGURE 4.8 – Exemple de concordance

4.8.3 Statistiques

Si l'on sélectionne l'onglet "Statistics" dans le cadre "Located sequences..", le panneau de la figure 4.9 apparaît. Ce panneau permet d'effectuer des calculs statistiques sur les séquences préalablement indexées.

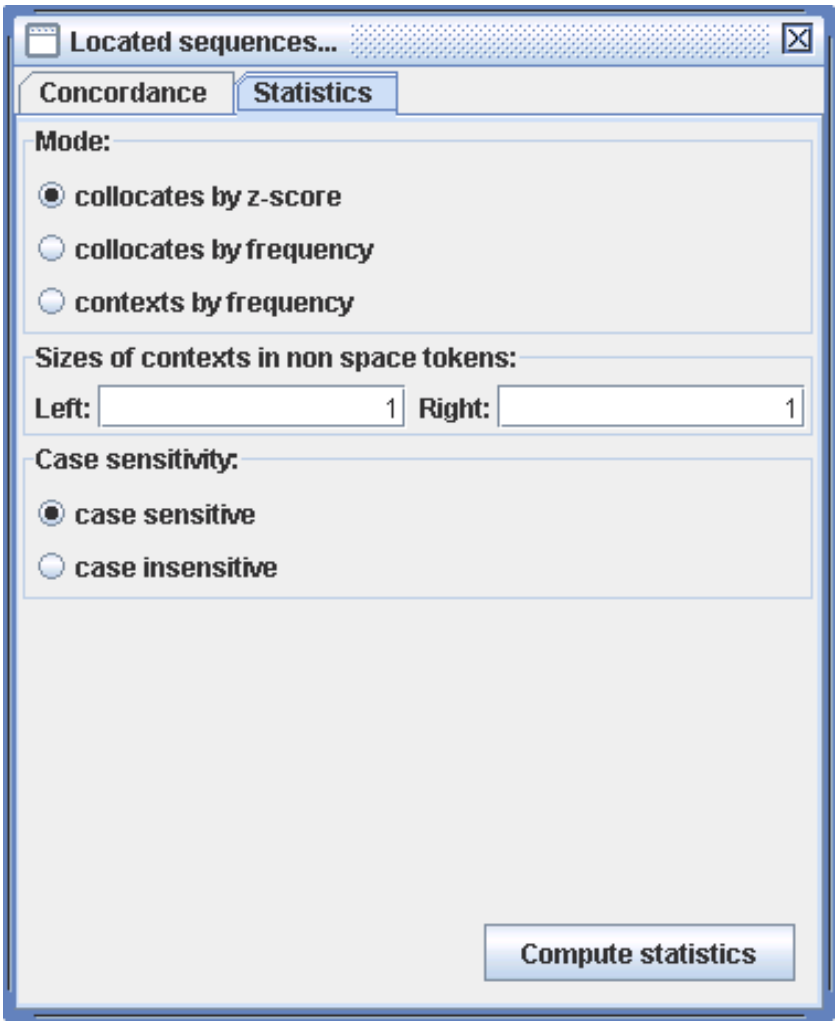


FIGURE 4.9 – Panneau statistiques

Dans le panneau "Mode" il est possible de choisir le type de statistiques désiré :

- collocates by frequency : montre les unités lexicales présentes dans le contexte de la séquence reconnu.
- collocates by z-score : le me mêmes informations avec, en plus (number of occurrences of the collocate in the match context and in the whole corpus, z-score of the collocate)

- contexts by frequency : montre les unités lexicales avec les contextes gauche et droit (voir au dessous). “count” est le nombre d’occurrences d’une séquence reconnue donnée (munie de contexte)

Dans le second panneau, on choisit la longueur des contextes gauche et droit à utiliser en tokens sans espace. NOTE : Cette notion de contexte n’a rien à voir avec celle utilisée dans les grammaires.

Dans le dernier panneau, on peut permettre ou non la variation de casse. Si cette variation est permise, `the` et `THE` sont considérées comme la même unité lexicale, et le résultat est la somme de ceux obtenus pour `the` et `THE`.

Les figures suivantes montrent les statistiques calculées pour chaque mode pour la requête `<have>` sur `ivanhoe.snt`.

Statistics

Left context	Match	Right context	Occurrences
would	have	been	10
which	had	been	10
,	had	been	7
might	have	been	6
,	had	not	5
he	had	been	5
I	have	been	5
to	have	been	5
it	had	been	5
we	have	already	4
he	had	hitherto	4
who	had	been	4
,	had	he	4
I	have	a	4
would	have	thought	4
,	had	they	3
,	having	been	3
I	have	no	3
he	had	received	3
that	had	been	3
thou	hast	seen	3
she	had	been	3
who	have	been	3
,	had	,	2

FIGURE 4.10 – contexte gauche+match+contexte droit+nombre d’occurrence

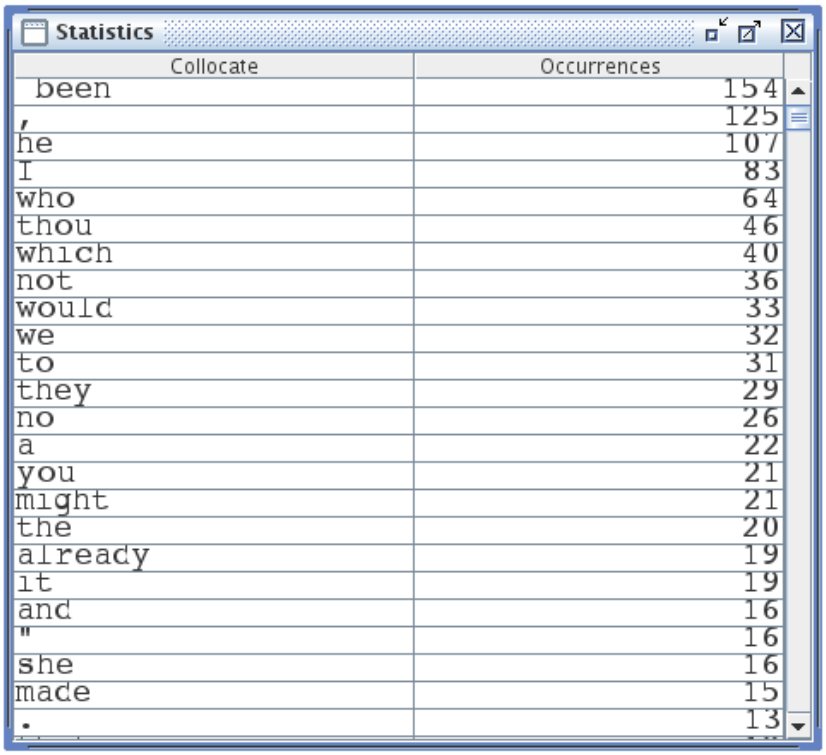


FIGURE 4.11 – collocate count

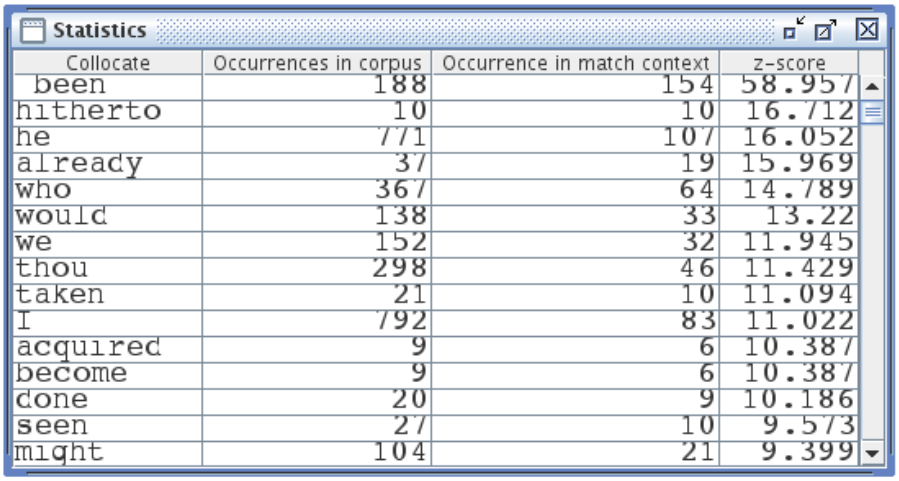


FIGURE 4.12 – collocate, count et d'autres informations

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 lesquel 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 :

$S \rightarrow aS$
$S \rightarrow \varepsilon$

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 to 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 *Languages algébriques* ou *Langages hors-contexte*.

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 intégrent 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 Création d'un graphe

Pour créer un graphe, cliquez sur "New" dans le menu "FSGraph" (5.1).

On voit alors apparaitre une fenêtre comme celle de la figure 5.2.

Pour pouvoir importer 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).

Le symbole en forme de flèche est *l'état initial* du graphe. Le symbole composé d'un rond contenant un carré est *l'état final* du graphe. La grammaire reconnaît les séquences décrites par les chemins allant de l'état initial à l'état final

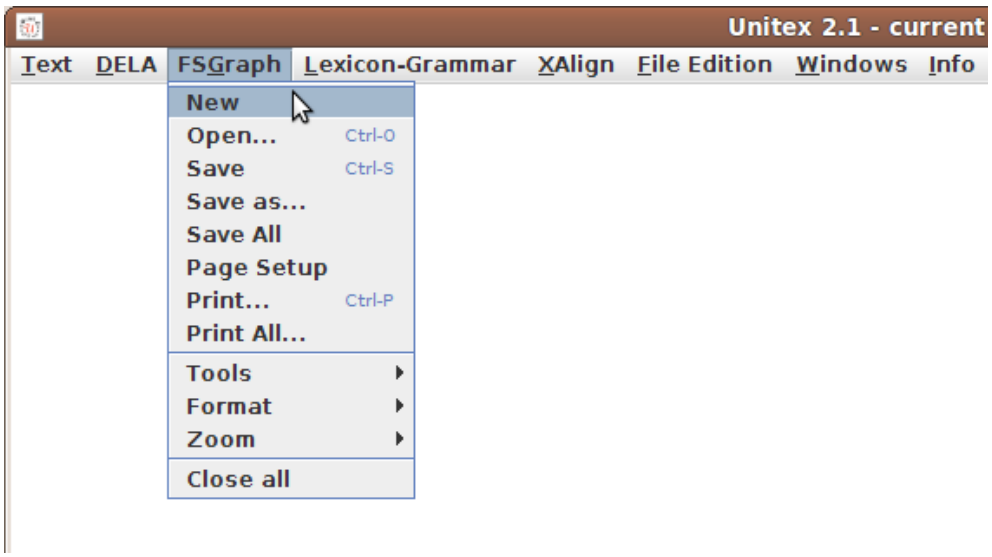


FIGURE 5.1 – Menu FSGraph

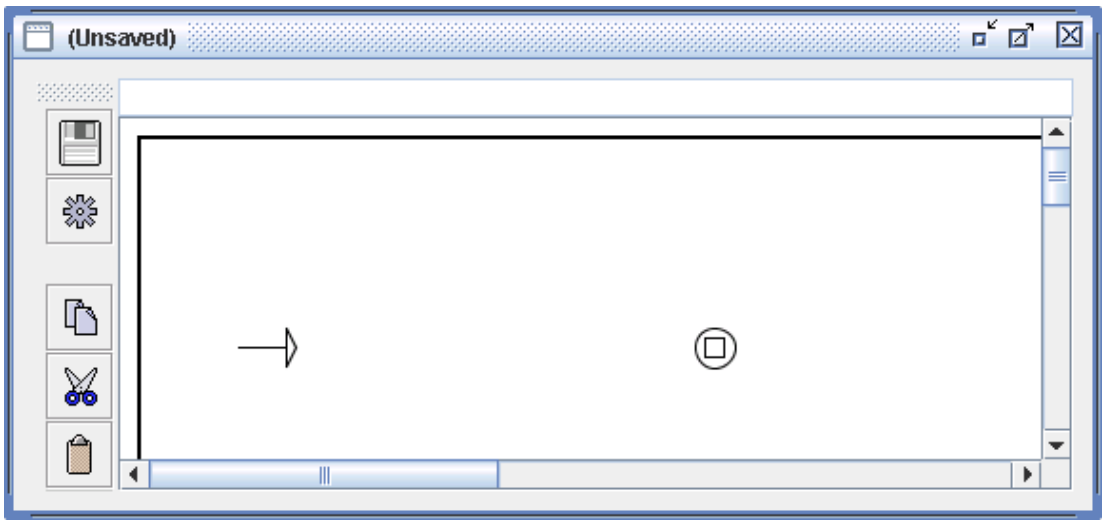


FIGURE 5.2 – Graphe vierge

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.

Le contenu de la boîte s'affiche dans la zone de texte située en haut de la fenêtre (figure 5.3). La boîte créée contient le symbole `<E>` qui représente le mot vide epsilon. Remplacez ce symbole par le texte `I+you+he+she+it+we+they` et validez en appuyant sur la touche Entrée. Vous venez de créer une boîte contenant sept lignes (voir figure 5.4).

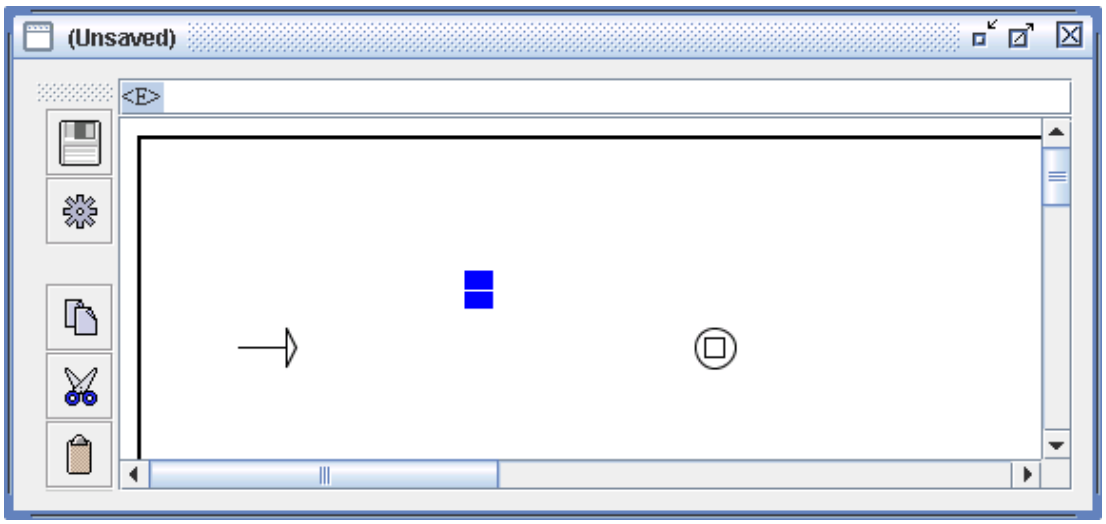


FIGURE 5.3 – Création d'une boîte

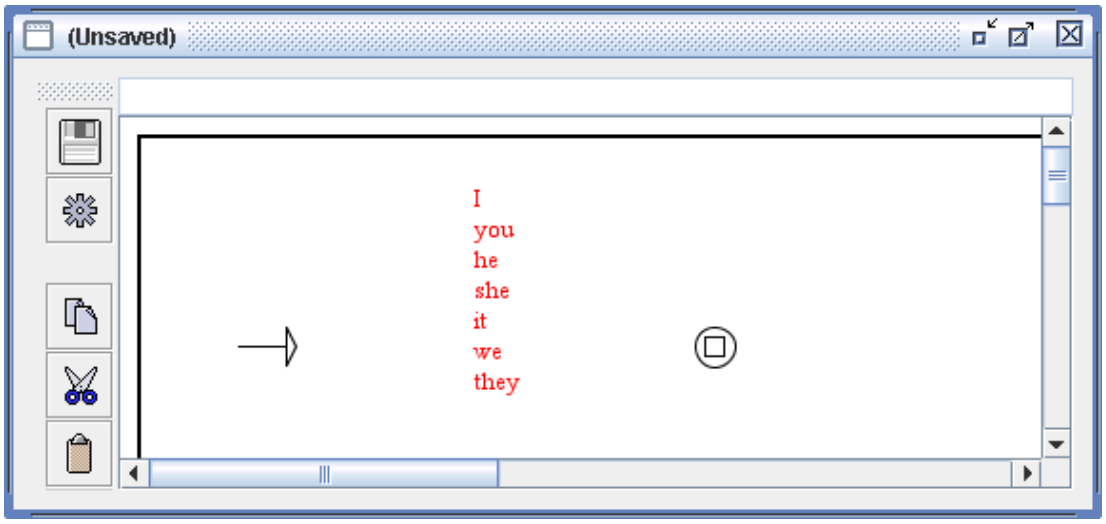


FIGURE 5.4 – Boîte contenant `I+you+he+she+it+we+they`

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.

Si vous souhaitez ajouter un commentaire dans un graphe, vous devez créer une boîte qui commence par /. Le texte de la boîte est affiché en vert, et peut contenir des lignes vides. La boîte ne peut avoir, ni de transition entrante, ni de transition sortante (voir figure 5.5).

Pour relier une boîte à une autre, il faut cliquer sur la boîte de départ, puis sur la boîte

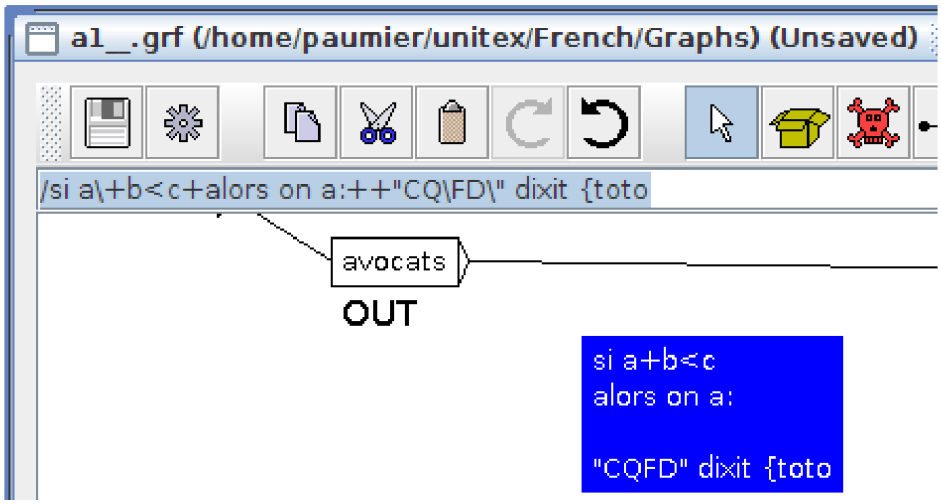


FIGURE 5.5 – Boîte contenant un commentaire

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 5.6 :

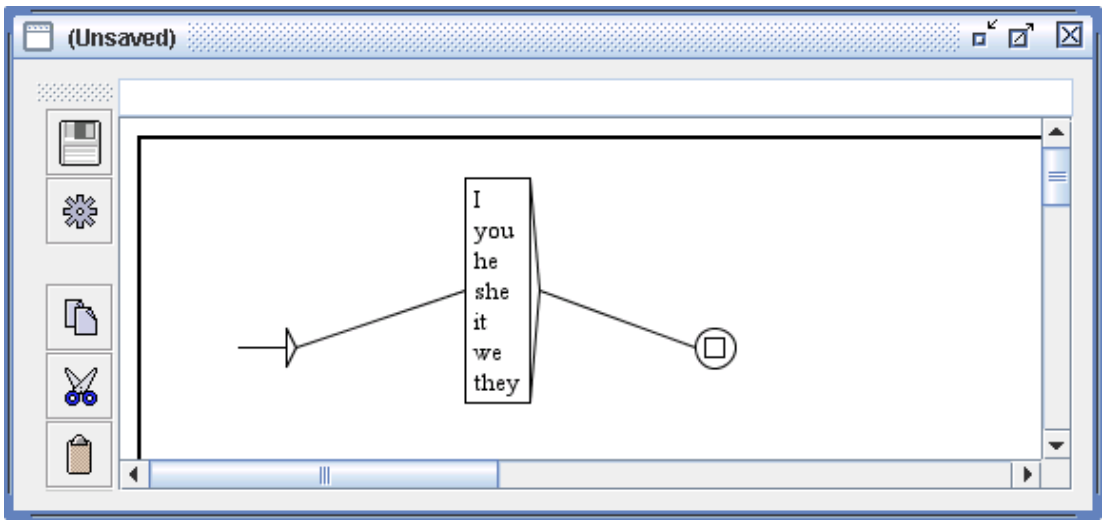


FIGURE 5.6 – Graphe reconnaissant des pronoms anglais

REMARQUE : si vous double-cliquez sur une boîte, vous relierez cette boîte à elle-même (voir figure 5.7). Pour annuler, double-cliquez une nouvelle fois sur la boîte.

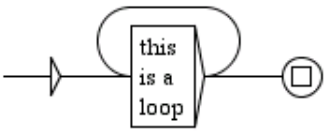


FIGURE 5.7 – Boîte reliée à elle-même

Cliquez sur "Save as..." dans le menu "FSGraph" pour enregistrer le graphe. Par défaut, Unitex propose d'enregistrer le graphe dans le sous-répertoire `Graphs` de votre répertoire de travail. Vous pouvez voir si le graphe a été modifié après le dernier enregistrement en vérifiant si le titre du graphe contient le texte `(Unsaved)`.

Un graphe peut contenir des boucles. Une boucle peut entourer une seule boite, comme dans la fig. 5.7, ou plusieurs, comme dans la fig. 5.16. Le contenu de la boucle sera reconnu n'importe quel nombre de fois en séquence. On peut fixer des limites au nombre de fois, mais uniquement pour une boucle autour d'une seule boite : voir la section 6.2.4.

Lorsqu'on modifie un graphe, on peut faire apparaître, par un clic droit, un menu contextuel (fig. 5.8) qui permet d'effectuer les opérations les plus usuelles :

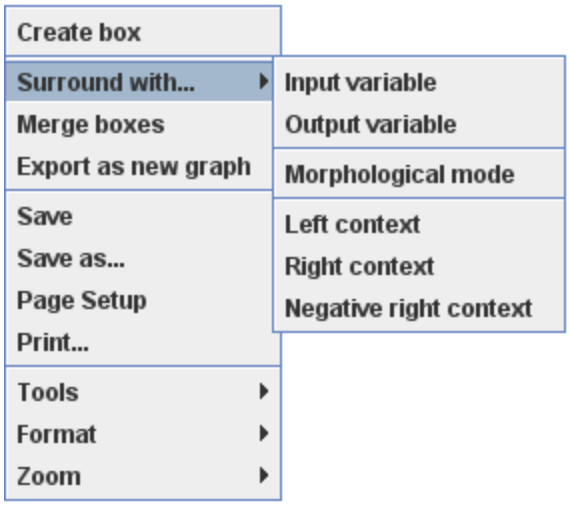


FIGURE 5.8 – Menu contextuel

- créer une boîte
- enregistrer ou imprimer le graphe courant ou modifier les paramètres de la page
- les menus habituels "Tools", "Format" et "Zoom" également accessibles dans le menu "FSGraph"

Si une ou plusieurs boîtes sont sélectionnées, les menus suivants deviennent accessibles, et permettent d'effectuer plusieurs types d'opérations sur cet ensemble de boîtes. Sinon, ils sont inutiles et donc désactivés.

- entourer les boîtes sélectionnées avec la définition d'une variable d'entrée ou de sortie, d'un contexte au sens de la section 6.3, ou des délimiteurs du mode morphologique. Ces opérations sont également réalisables avec la barre d'outils de la fenêtre d'édition du graphe (voir section 5.2.8).
- fusionner les boîtes sélectionnées
- exporter les boîtes sélectionnées en tant que nouveau graphe

5.2.2 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 dans une boîte le texte suivant :

```
alpha+:beta+gamma+:E:\greek\delta.grf
```

vous obtiendrez une boîte similaire à celle de la figure 5.9.

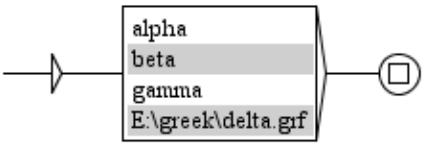


FIGURE 5.9 – Graphe faisant appel aux sous-graphes `beta` et `delta`

Vous pouvez indiquer le nom complet du graphe (`E:\greek\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. Il est déconseillé d'utiliser des noms de graphes comportant des chemins absolus, car cela nuit à leur portabilité. Si vous utilisez un nom de graphe absolu, comme c'est ici le cas pour `E:\greek\delta.grf` le compilateur de graphe émettra un avertissement (voir figure 5.10).

Pour les mêmes raisons de portabilité, il est déconseillé d'utiliser `\` ou `/` comme séparateur dans les noms de graphes. À la place, il vaut mieux utiliser le caractère `:` qui joue le rôle de séparateur universel, valable quel que soit le système sous lequel vous travaillez. On peut d'ailleurs voir sur la figure 5.10 que c'est ce séparateur qui est utilisé en interne par le compilateur de graphe (`E::greek:delta.grf`).

Répertoire de dépôt

Lorsqu'on souhaite réutiliser une grammaire X dans une grammaire Y, une pratique répandue est de recopier tous les graphes de X dans le répertoire où se trouvent les graphes de Y, ce qui pose deux problèmes :

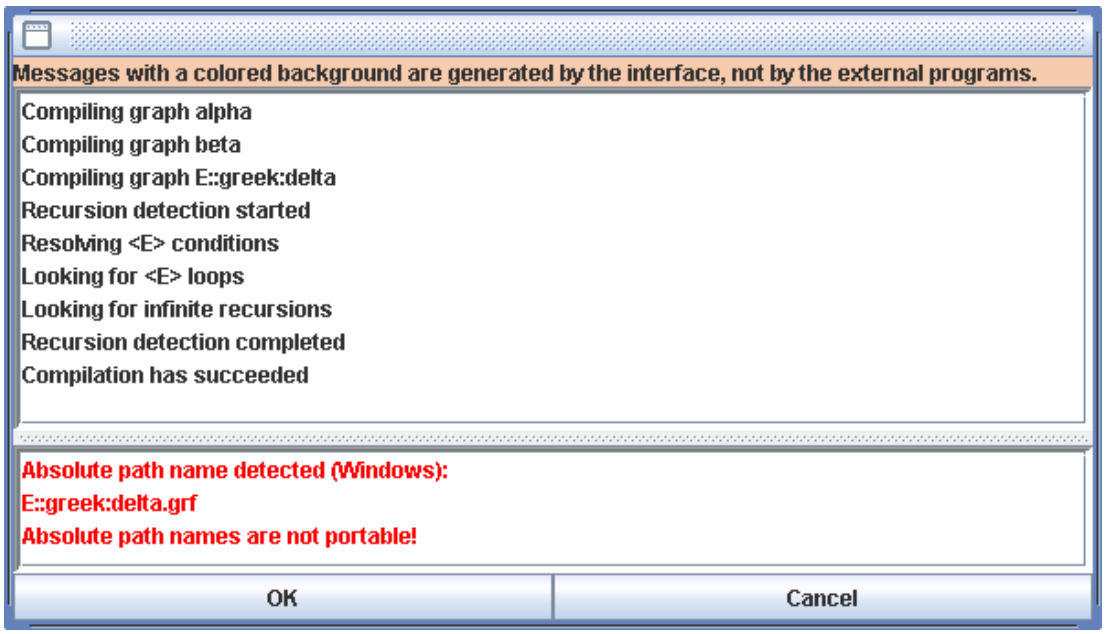


FIGURE 5.10 – Avertissement pour un nom de graphe non portable

- le nombre de graphes dans le répertoire devient vite très important ;
- deux graphes ne peuvent pas avoir le même nom.

Afin d'éviter cela, il est possible de stocker la grammaire X dans un répertoire particulier, appelé *répertoire de dépôt*. Ce répertoire est une sorte de bibliothèque dans laquelle on peut ranger des graphes, et faire ensuite appel à ces graphes au moyen de `::` au lieu de `:`. Pour utiliser ce mécanisme, il faut tout d'abord définir le répertoire de dépôt dans le menu "Info>Preferences...>Directories" (voir figure 5.11). Choisissez votre répertoire dans le cadre "Graph repository". Le répertoire de dépôt est propre à la langue de travail, vous n'êtes donc pas obligé d'utiliser le même répertoire pour plusieurs langues.

Supposons que l'on ait une arborescence comme celle de la figure 5.12. Si l'on souhaite faire appel au graphe `DET` qui se trouve dans le sous-répertoire `Johnson`, on utilisera l'appel `::Det:Johnson:DET` (voir figure 5.13 [1]).

ASTUCE : si vous voulez éviter de mettre dans vos graphes un chemin compliqué comme `::Det:Johnson:DET`, vous pouvez créer un graphe nommé `DET` que vous placerez à la racine du répertoire de dépôt `D:\repository\DET.grf`). Ce graphe contiendra simplement un appel au graphe `::Det:Johnson:DET`. Vous pourrez alors mettre dans vos graphes un simple appel à `::DET`. Cela permet 1) de ne pas avoir de noms compliqués et 2) de pouvoir

1. Dans un souci de clarté, les appels à des graphes du répertoire de dépôt sont affichés sur fond kaki au lieu de gris.

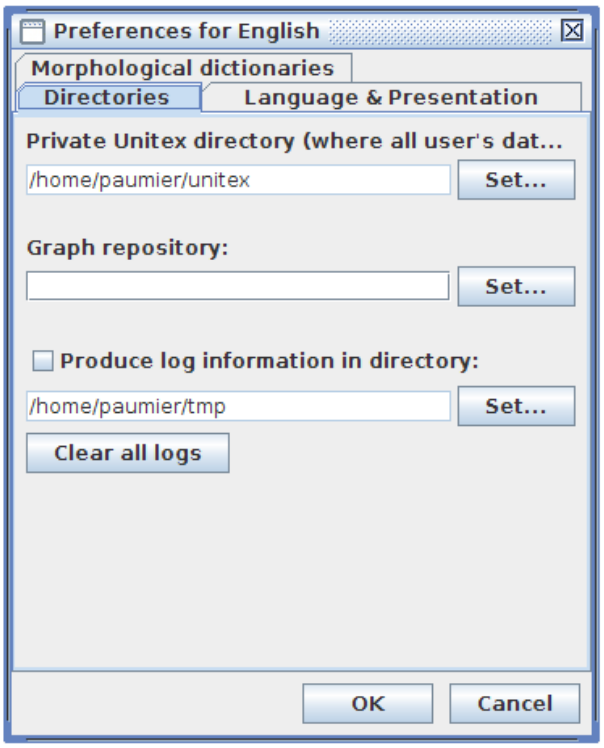


FIGURE 5.11 – Configuration du répertoire de dépôt

FIGURE 5.12 – Exemple de répertoire de dépôt

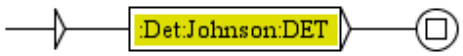


FIGURE 5.13 – Appel un graphe du répertoire de dépôt

modifier les graphes du répertoire de dépôt sans avoir à modifier tous vos graphes. En effet, il vous suffira de mettre à jour le graphe situé à la racine du répertoire de dépôt.

Les appels à des sous-graphes sont représentés dans les boîtes par des lignes sur fond gris (figure 5.9), ou kaki dans le cas de sous-graphes à rechercher dans le répertoire de dépôt (figure 5.13). Si le fichier `.grf` du sous-graphe n'est pas trouvé au chemin indiqué, Unitex

cherchera le fichier `.fst2` de même nom. Si Unitex ne trouve ni le fichier `.grf` ni le fichier `.fst2`, l'appel au graphe manquant apparaît dans une ligne sur fond rouge.

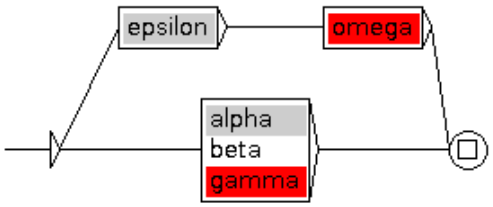


FIGURE 5.14 – Les sous-graphes manquants apparaissent en rouge

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[2] : pour ouvrir un sous-graphe, faites un clic central sur son nom (avec le bouton central) ou faites un clic simultané (avec les boutons gauche et droit).

La liste des graphes appelés par le graphe courant et celle des graphes qui appellent le graphe courant peuvent être affichées en cliquant sur le second et troisième bouton du quatrième groupe de boutons de la barre d'outils (figure 5.15 ; voir aussi figure 5.29, section 5.2.8). Dans ces listes de sous-graphes :

- les sous-graphes directement appelés par le graphe courant apparaissent avec leur simple nom de fichier
- les sous-graphes indirectement appelés par l'un des graphes appelés par le graphe courant apparaissent avec une flèche devant leurs nom
- les sous-graphes qui apparaissent dans des graphes appelés par le graphe courant sans être connectés et donc non traités ont leur nom en orange
- les sous-graphes non trouvés (ni en .grf ni en .fst2) apparaissent en rouge.

5.2.3 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 relâcherez le bouton, toutes les boîtes touchées par le rectangle de sélection seront sélectionnées et s'afficheront alors en blanc sur fond bleu (figure 5.16).

Vous pouvez sélectionner plusieurs boîtes en maintenant les touches <CTRL> et <SHIFT> et en cliquant sur chaque boîte à ajouter à la sélection. De cette manière, vous pouvez sélectionner plusieurs boîtes sans avoir à sélectionner une zone complète (figure 5.17).

2. Si vous travaillez sous KDE, désactivez <Alt+Click> dans kcontrol.

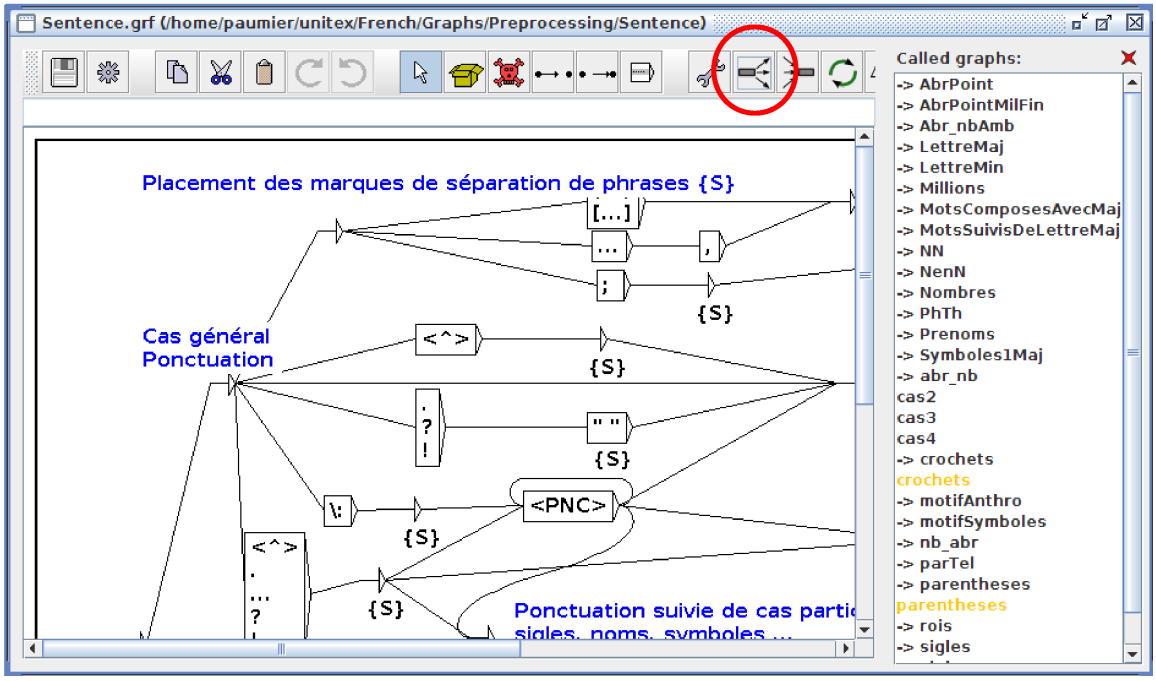


FIGURE 5.15 – Affichage de la liste de tous les graphes appelés

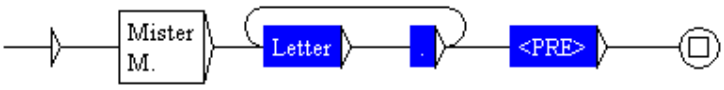


FIGURE 5.16 – Sélection de plusieurs boîtes

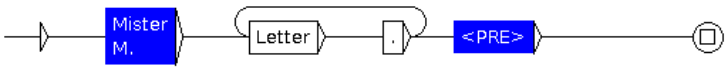


FIGURE 5.17 – Sélection de boîtes éloignées

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, comme dans la figure 5.18. 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".

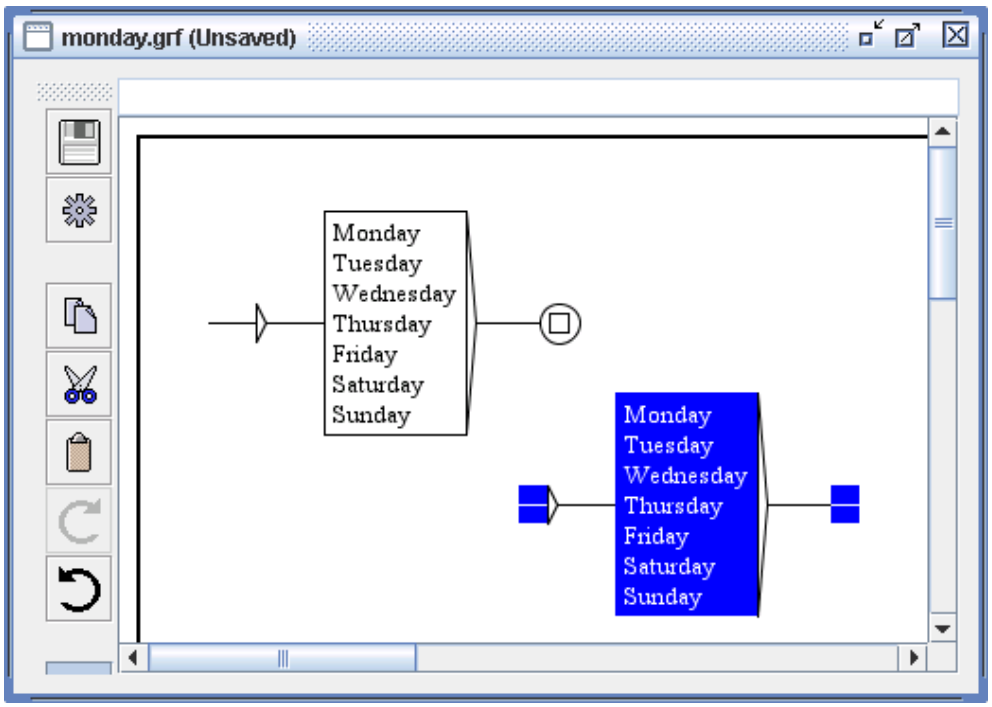


FIGURE 5.18 – Copier-coller d'une sélection multiple

NOTE : Vous pouvez coller une sélection multiple dans des graphes différents de celui dont elle est issue.

Pour supprimer des boîtes, sélectionnez-les, effacez le texte qu'elles contiennent (c'est-à-dire le texte affiché dans le champ situé en haut de la fenêtre) et appuyez sur Enter.

On ne peut pas supprimer l'état initial ni l'état final.

5.2.4 Sorties

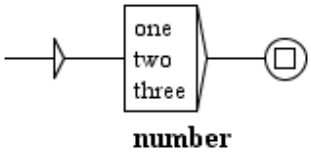


FIGURE 5.19 – Exemple de sortie

Il est possible d'associer une sortie à une boîte. Pour cela, on utilise le caractère spécial /. Tous les caractères situés à droite de celui-ci seront considérés comme faisant partie de la

sortie. Ainsi, le texte `one+two+three/number` donne la boîte de la figure 5.19. La sortie associée à une boîte est représentée en gras au-dessous de celle-ci.

Les graphes dans lesquels les boites ont des sorties sont souvent utilisés pour produire une version modifiée d'un corpus (voir section 6.10.4). Soit les sorties sont insérées dans le corpus, soit elles remplacent le texte reconnu par la boite à laquelle elles sont attachées, selon qu'on lance la recherche (section 6.10) en mode MERGE ou en mode REPLACE (section 6.7).

Cependant, il n'est pas autorisé de placer une sortie sur une boîte contenant un appel à un sous-graphe. Pour obtenir le même effet, il faut insérer une boîte vide avant celle de l'appel, et placer la sortie sur la boîte vide, comme dans la figure 5.20.

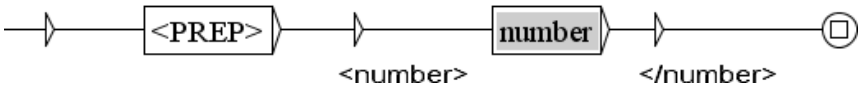


FIGURE 5.20 – La boite vide juste avant l'appel a une sortie

Boites vides avec une sortie

Pour créer une boite vide avec une sortie contenant `number`, on écrit `<E>/number`. Quand on utilise un graphe pour produire une version modifiée d'un corpus, la sortie d'une boite vide sera insérée dans le corpus, qu'on lance la recherche en mode MERGE ou en mode REPLACE (section 6.7) : de toutes façons, la boite vide ne reconnait aucune portion de texte. Par exemple, le graphe de la figure 5.21, utilisé en mode MERGE, insère `<number>` avant les nombres.

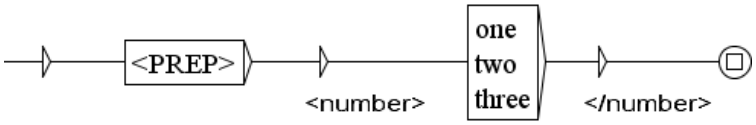


FIGURE 5.21 – Boites vides avec sorties

La sortie est insérée entre le texte produit par la boite précédente (ici la préposition) et celui produit par la boite suivante (ici le nombre).

Dans la version modifiée du corpus, la sortie est soudée au token de gauche : avec le graphe de la figure 5.21, `<number>` est soudé à la préposition. Cela donne, par exemple :

```
(...) with<number> two</number> of (...)
```

Si on veut au contraire souder la sortie au token de droite, on peut utiliser le caractère spécial `#` (section 4.3.1) comme dans la figure 5.22. La version modifiée du corpus contiendra maintenant :

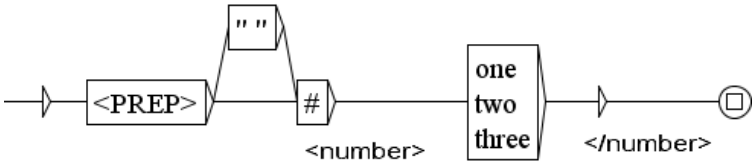


FIGURE 5.22 – Une boite vide dont la sortie sera soudée au mot suivant

```
(...) with <number>two</number> of (...)
```

De même, si on veut insérer une accolade ouvrante et faire en sorte qu'elle soit soudée au texte qui suit dans la version modifiée du corpus, on peut appeler le graphe *B* de la figure 5.23.

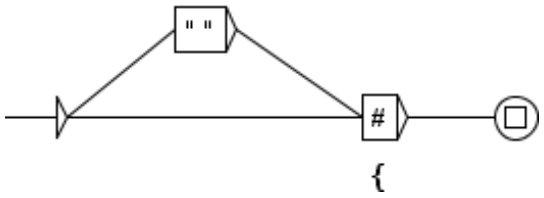


FIGURE 5.23 – Le graphe *B*

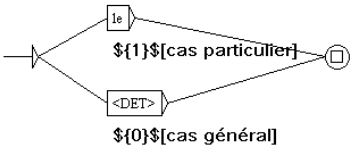
Vérification des accolades

Unitex peut vérifier si toutes les accolades ouvrantes d'une grammaire sont bien fermées sur tous les chemins possibles. (Cette commande est utile, par exemple, dans le cas d'une grammaire qui crée des étiquettes lexicales délimitées par des accolades.) Pour invoquer cette commande, on choisit dans le menu *Graphs* la commande *Tools/Verify braces*. La commande vérifie aussi les sous-graphes appelés, et compile les graphes en même temps [3]. Les accolades à l'intérieur des boites (y compris les accolades déspécialisées) comptent aussi bien que celles dans les sorties. L'outil ne vérifie pas si la grammaire ouvre les accolades avant de les fermer ou après.

La vérification suppose que les accolades qui sont ouvertes dans un graphe sont fermées dans le même, sauf qu'un appel à un graphe nommé *B*, quel que soit son contenu, compte automatiquement comme une accolade ouvrante [4]. Cette exception est conçue pour être utilisée avec le graphe de la figure 5.23 [5].

3. L'outil crée aussi un fichier `<nom du graphe>autolst.txt`, qu'on peut effacer une fois l'analyse terminée.
4. Un appel à un graphe *BB* compte pour deux accolades ouvrantes, et à *BBB* pour trois.
5. L'outil vérifie aussi les accolades dans les sous-graphes qui porteent les noms spéciaux *B*, *BB* et *BBB*, mais dans tout graphe, il ignore les chemins dans lesquels la séquence des contenus des boites se termine par # et la séquence des sorties des boites commence par {, ce qui est le cas de tous les chemins du graphe *B*.

Poids On peut attribuer un poids à des boîtes d'un transducteur. Ainsi, lorsqu'une séquence est reconnue par plusieurs chemins avec des sorties différentes (transducteur ambigu), seul un chemin de poids maximal sera conservé. Après un "Locate", la concordance ne comportera qu'une seule fois la séquence reconnue, et avec la sortie appropiée (figure 5.24).



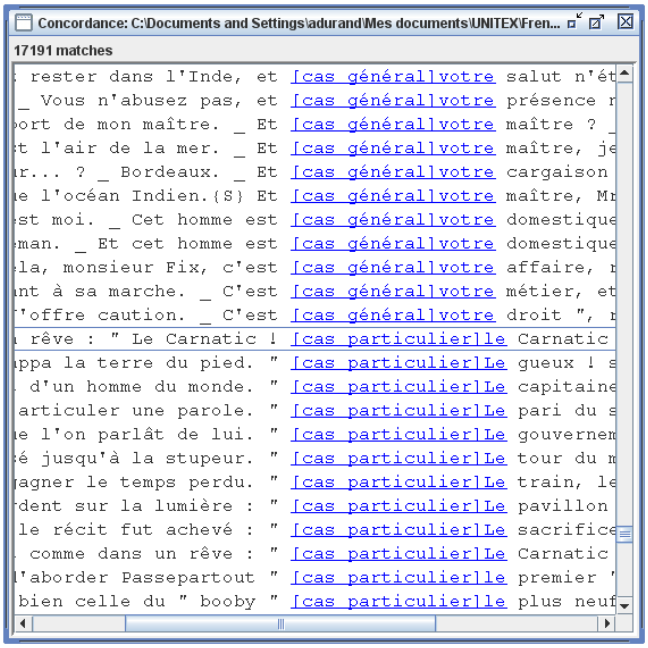


FIGURE 5.24 – Poids dans les graphes

Les poids sont des valeurs entières. Pour donner à une boite le poids 1, on insère `${1}$` dans la sortie de la boite, comme dans `<E>/${1}$`.

Le poids d'un chemin est le dernier poids trouvé en parcourant le chemin. Un poids peut être nul, mais pas strictement négatif. Un chemin qui a un poids, même nul, a la priorité sur un chemin sans poids.

Avec des poids, on peut définir une priorité entre des chemins qui reconnaissent la même séquence. On ne peut pas définir une priorité entre deux séquences dont une est incluse dans l'autre (cf. section 4.8.1), ni entre des séquences qui se chevauchent (cf. section 6.7.3).

Les poids ne sont valides qu'à l'intérieur du graphe, et non dans les sous-graphes ni les graphes appelants.

5.2.5 Variables d'entrée

Il est possible de sélectionner des parties du texte reconnu par une grammaire au moyen de variables d'entrée. Pour associer une variable d'entrée `var1` à une partie d'une grammaire, on utilise soit le bouton avec les parenthèses rouges dans la barre d'icônes au-dessus

du graphe (section 5.2.8), soit les symboles spéciaux `$var1(` et `$var1)`. (Ces symboles définissent respectivement le début et la fin de la zone à mémoriser. 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 5.25, on reconnaît une séquence commençant par un nombre, que l'on stocke dans une variable nommée `var1`, suivi de `dollar` ou `dollars`.

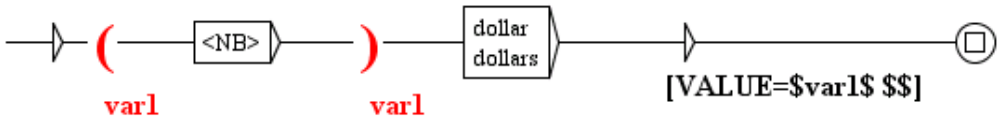


FIGURE 5.25 – Utilisation d'une variable d'entrée `var1`

Les noms de variables peuvent contenir des lettres latines 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 encadrant son nom avec le caractère `$`. La grammaire de la figure 5.26 reconnaît une date formée d'un mois et d'une année, et produit en sortie la même date, mais dans l'ordre année-mois.

Si on veut utiliser le caractère `$` en sortie d'une boîte, on doit le redoubler, comme le montre la figure 5.25.

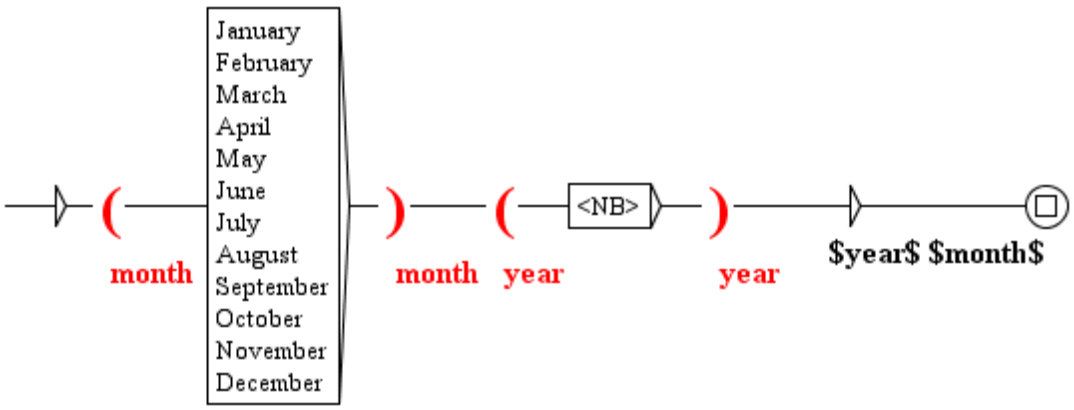


FIGURE 5.26 – Interversion du mois et de l'année dans une date

Quand une boite redéfinit une variable qui avait déjà été définie, la nouvelle valeur écrase l'ancienne. Ainsi, si la variable est définie dans une boucle, la valeur de la variable juste après la boucle dépend du dernier passage dans la boucle.

Par défaut `Locate` et `LocateTfst` considèrent que les variables non définies sont vides. On peut modifier ce comportement (voir section 6.10.2). De plus, il est possible dans un graphe d'interroger une variable pour savoir si elle a été initialisée ou non (section 6.7.5).

5.2.6 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 5.27.

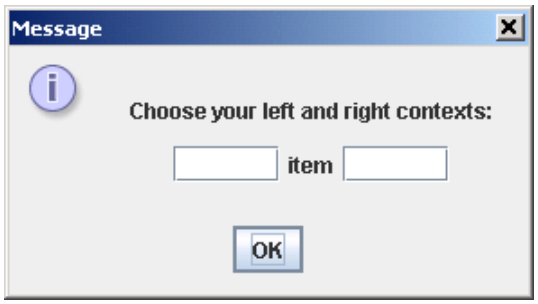


FIGURE 5.27 – Sélection de contexte pour la copie d'une liste

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 :

eat
sleep
drink
play
read

on obtient la boîte de la figure 5.28.

5.2.7 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.

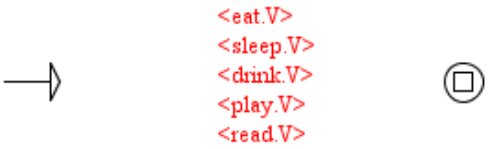


FIGURE 5.28 – Boîte obtenue par copie d'une liste avec ajout de contextes

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

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

5.2.8 Commandes de la barre d'icônes

La barre d'icônes présente au-dessus 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 5.29). Dans ce cas, le fait de fermer cette fenêtre replace la barre d'icônes à sa position initiale. Chaque graphe possède sa propre barre d'icônes.

FIGURE 5.29 – Barre d'outils

Les deux premières icônes sont des raccourcis permettant de sauver et de compiler le graphe. Les cinq suivantes correspondent aux opérations "Copier", "Couper", "Coller", "Redo" et "Undo".

Les six icônes suivantes 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.

Pour que le curseur retrouve sa forme initiale de flèche blanche, faites un clic droit sur le fond du graphe : les clics seront à nouveau interprétés normalement.

L'icône en forme de clé anglaise est un raccourci pour ouvrir la fenêtre des options d'affichage du graphe. Les deux suivantes permettent de voir les listes de graphes en relation avec le graphe courant :

- Le premier bouton affiche la liste des graphes appelés par le graphe courant
- Le deuxième bouton affiche la liste des graphes qui appellent le graphe courant

Le bouton muni de deux flèches vertes rafraîchit le graphe courant en chargeant sa dernière version. Si un fichier `.grf` est modifié alors que le graphe qu'il contient est affiché dans une fenêtre Unitex, une fenêtre pop up vous invitera à le recharger.

Le bouton portant l'icône d'une balance permet de comparer le graphe courant à un autre graphe ou à une autre version du même graphe. Une nouvelle fenêtre est alors affichée (voir figure 5.30) qui contient les deux graphes avec des couleurs qui indiquent les types de différences entre les deux graphes : insertion, suppression, déplacement de boîtes et changement de contenu d'une boîte apparaissent respectivement en vert, rouge, mauve et jaune.

Les sept derniers boutons sont des raccourcis pour définir une variable, utiliser le mode morphologique, déclarer comme contexte une ou plusieurs boîtes sélectionnées, ou définir une boite de généralisation d'étiquetage. Ces boutons ne sont activés que si une ou plusieurs boîtes sont sélectionnées :

- () : variable d'entrée (voir section 5.2.5)
- () : variable de sortie (voir section 6.8)

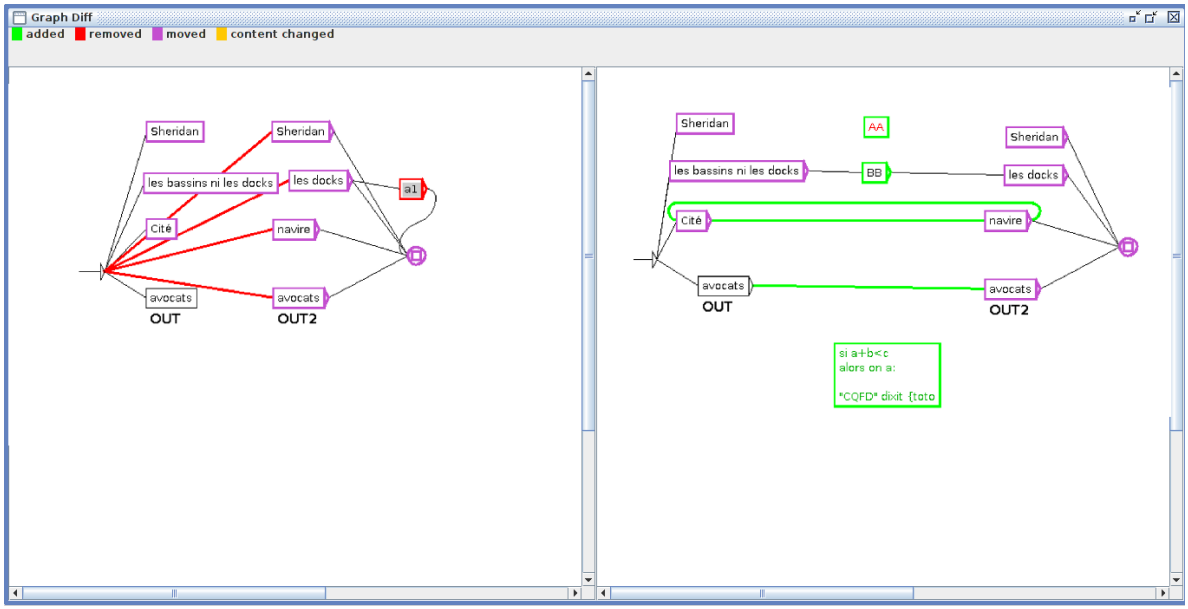


FIGURE 5.30 – DIFF

- <> : mode morphologique (voir section 6.4)
- $* : contexte gauche (voir section 6.3)
- $[: contexte droit (voir section 6.3)
- $![: contexte droit négatif (voir section 6.3)
- $G : généralisation d'étiquetage (voir section 12.3)

5.2.9 Rechercher et remplacer dans les graphes

Pour chercher une expression ou une suite de boites dans des graphes, on ouvre un des graphes et on clique soit sur "Find and replace" dans le menu "FSGraph", soit sur la loupe dans la barre d'outils. Il apparait une fenêtre comme celle de la figure 5.31.

L'onglet "Inside one box" permet de rechercher une expression entièrement contenue dans une des boites. On peut trouver toutes les occurrences d'un mot, d'une expression ou d'une chaine de caractères, dans une boite, sous une boite (cf. 5.2.4) ou les deux. On écrit ce qu'on cherche dans "Find what", puis on clique sur "Find Previous" ou "Find Next" pour atteindre chaque boite qui contient une ou plusieurs occurrences de la requête. La boite trouvée est signalée par un cadre coloré. Si une boite contient le nom d'un sous-graphe ou d'une variable (cf. 5.2.5) ou encore des codes spécifiques, le programme traite cela comme n'importe quel autre contenu de boite.

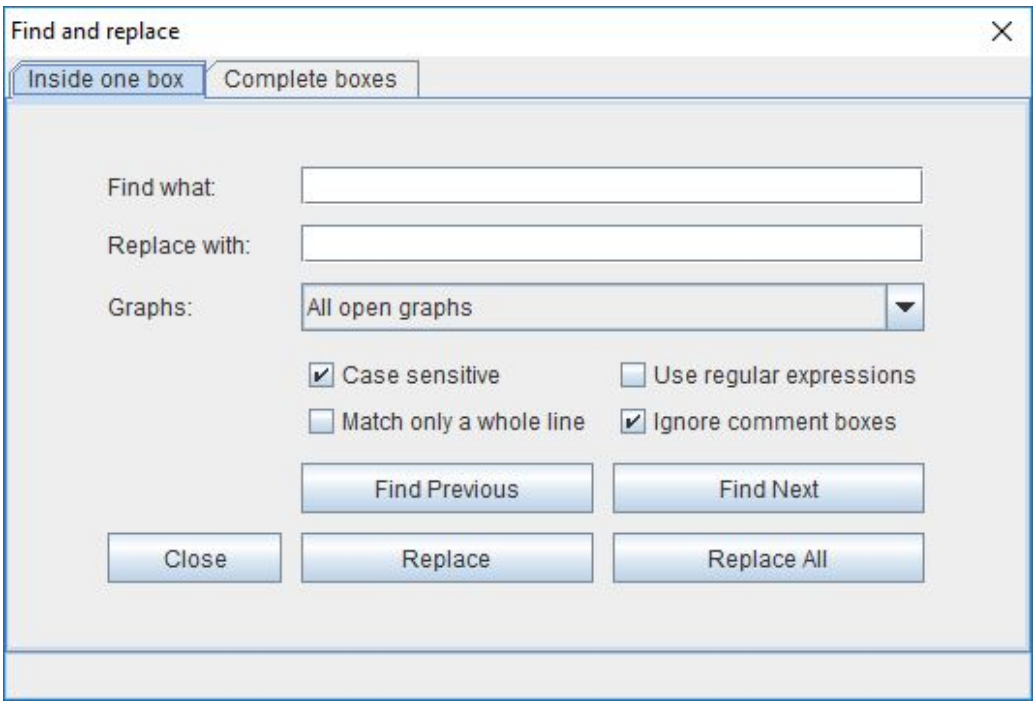


FIGURE 5.31 – Fenêtre pour rechercher et remplacer dans une boite

La liste dans "Graphs" permet de rechercher et de remplacer soit dans un graphe particulier, soit dans tous les graphes ouverts.

On peut remplacer la ou les occurrences présentes dans une boite par le contenu de "Replace with" : on sélectionne la boite qu'on veut modifier, puis on clique sur "Replace". Le contenu de "Find what" dans la boite est remplacé par le contenu de "Replace with".

Pour remplacer toutes les occurrences présentes dans le ou les graphes, on clique sur "Replace All".

On peut restreindre ou étendre la recherche en cochant ou en décochant des options :

- Case sensitive : la recherche fait la différence entre majuscules et minuscules.
- Use regular expressions : la recherche interprète la requête comme une expression régulière conforme aux conventions POSIX, par exemple `un.*able` reconnait une séquence de caractères qui contient *un* quelque part et *able* plus loin.
- Match only a whole line : la recherche cible uniquement des lignes complètes. Dans une boite avec une sortie, le programme considère que la sortie fait partie de la dernière ligne de la boite.
- Ignore comment boxes (par défaut) : la recherche ignore les boites qui ne sont liées à aucune autre.

Pour rechercher une séquence de boites dans un ou plusieurs graphes, on sélectionne l'onglet "Complete boxes" (figure 5.32).

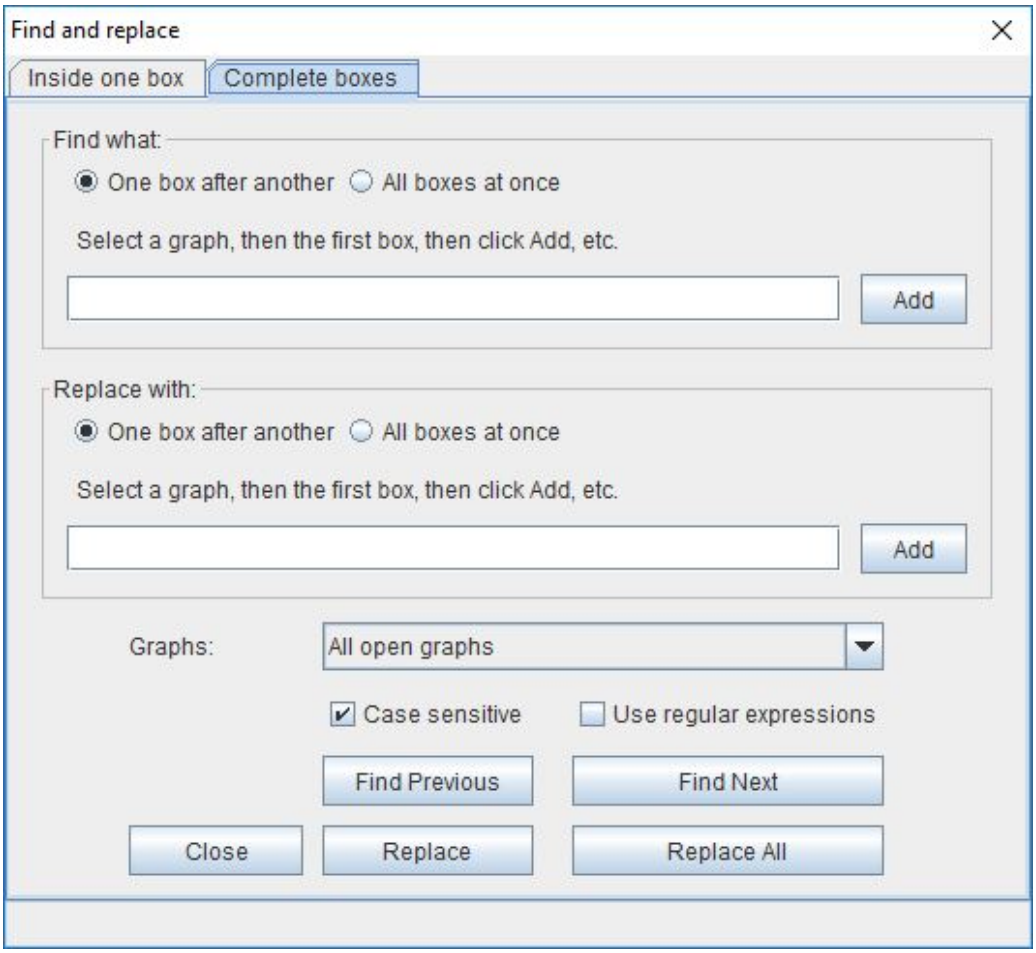


FIGURE 5.32 – Fenêtre pour rechercher et remplacer des boites complètes

On peut trouver toutes les occurrences d'une séquence de boites. Pour former une requête, le plus simple est de trouver d'abord un ou des graphes existants qui contiennent une occurrence des premières boites de la séquence (on peut aussi faire un graphe pour cela, sans qu'il soit obligatoire de le sauvegarder), on sélectionne un graphe soit en cliquant dessus soit en utilisant le menu déroulant "Graphs", on clique sur une occurrence de la première boite et on clique sur le bouton Add. On répète l'opération pour ajouter à la requête les autres boites. Le champ à côté du bouton Add affiche la requête, avec chaque boite séparée de la suivante par un caractère ►. Pour ajouter plusieurs boites en même temps, on coche l'option "All boxes at once", on sélectionne une séquence de boites (cf. 5.2.3) et on les ajoute.

Une fois la requête prête, on s'assure que le menu déroulant "Graphs" indique le ou les graphes dans lesquels on veut rechercher, puis on clique sur "Find Next" ou "Find Previous".

Le programme trouve les séquences de boites qui coïncident avec la requête. Une boite avec plusieurs lignes ne correspond qu'à une boite qui contient les mêmes lignes dans le même ordre.

Le programme ne recherche que les séquences qui ne contiennent aucune transition entrante ni sortante, à part des transitions entrantes dans la première boite ou des transitions sortantes dans la dernière. Grâce à cette restriction, l'opération Replace est clairement définie.

"Replace" et "Replace All" remplacent la séquence indiquée dans "Find what" par celle indiquée dans "Replace with". Les transitions entrantes de la première boite et les sortantes de la dernière sont conservées.

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.

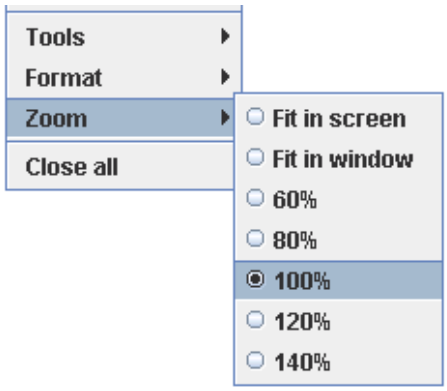


FIGURE 5.33 – Sous-menu Zoom

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 rendu qui permet d'éviter l'effet de pixellisation. Vous pouvez activer cet effet en cliquant sur "Antialiasing..." dans le sous-menu "Format". La figure 5.34 montre deux graphes affichés normalement (graphe du haut) et avec antialiasing (graphe du bas).

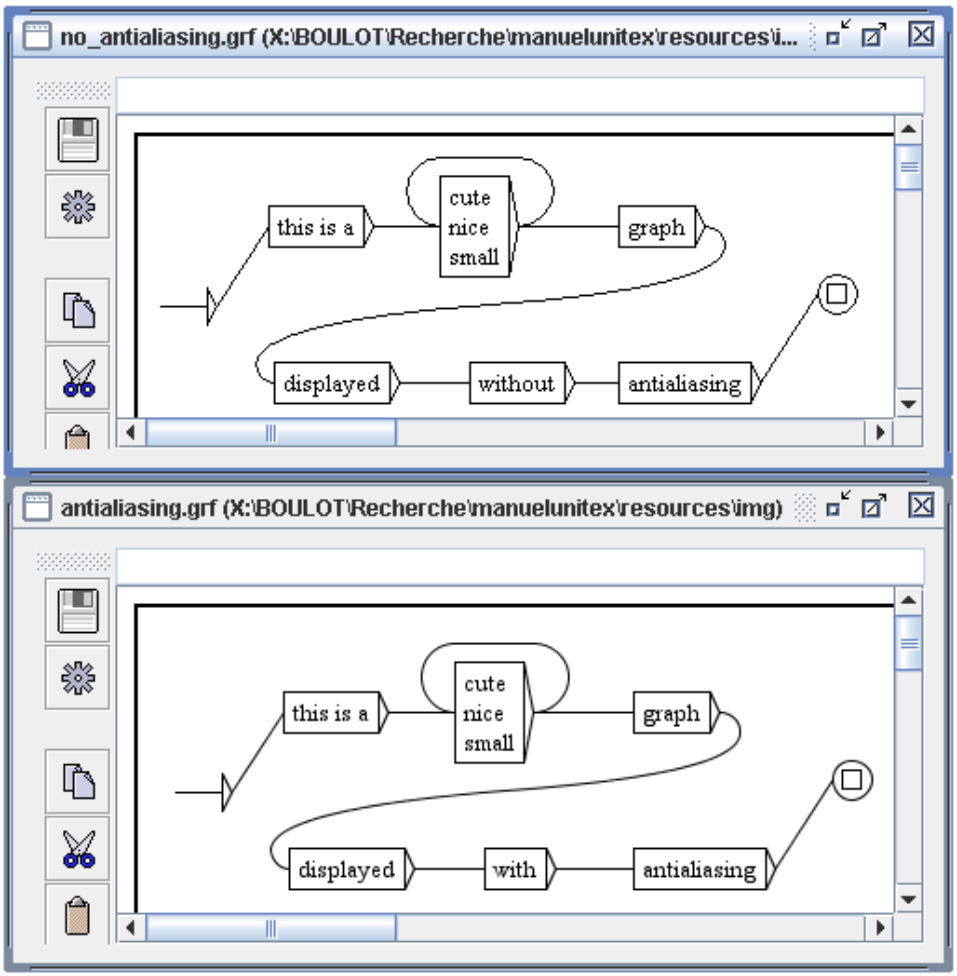


FIGURE 5.34 – Exemple d'antialiasing

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 "Alignment..." dans le sous-menu "Format" du menu "FSGraph" ou appuyez sur <Ctrl+M>. Vous voyez alors apparaître la fenêtre de la figure 5.35.

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.

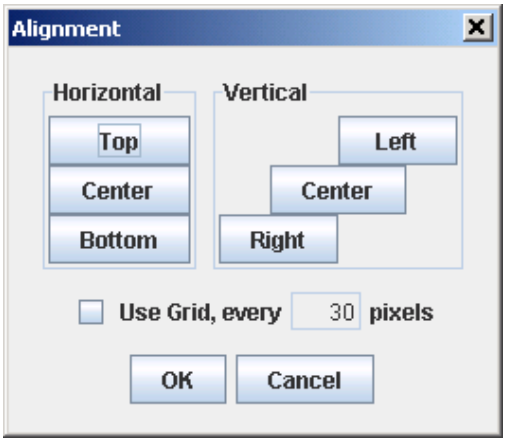


FIGURE 5.35 – Fenêtre d'alignement

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 5.36 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.

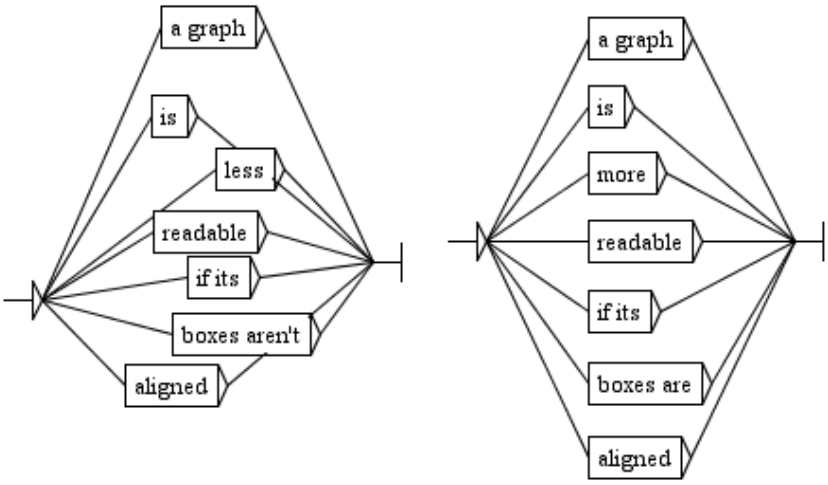


FIGURE 5.36 – Exemple d'alignement vertical gauche

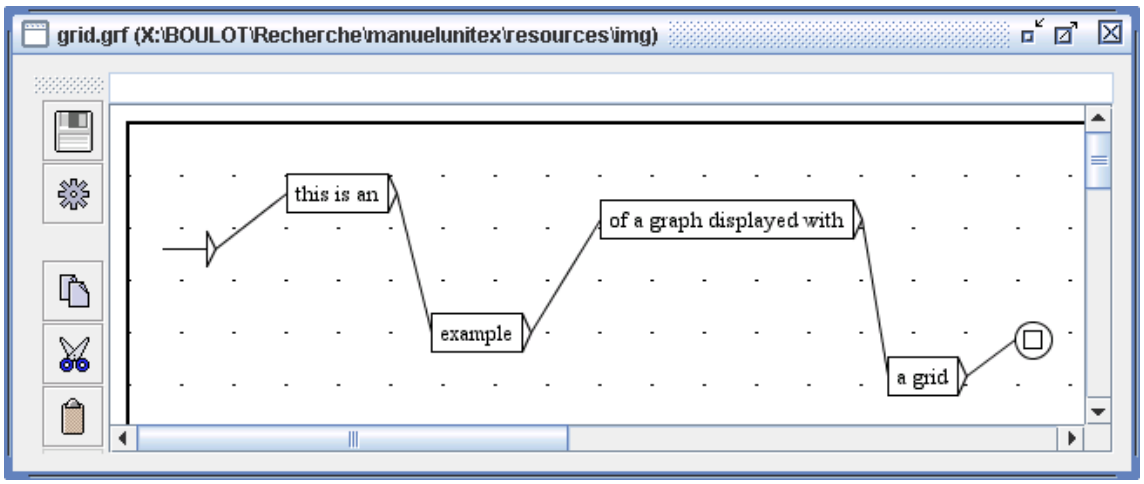


FIGURE 5.37 – Exemple d'utilisation d'une grille

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 5.38.

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.

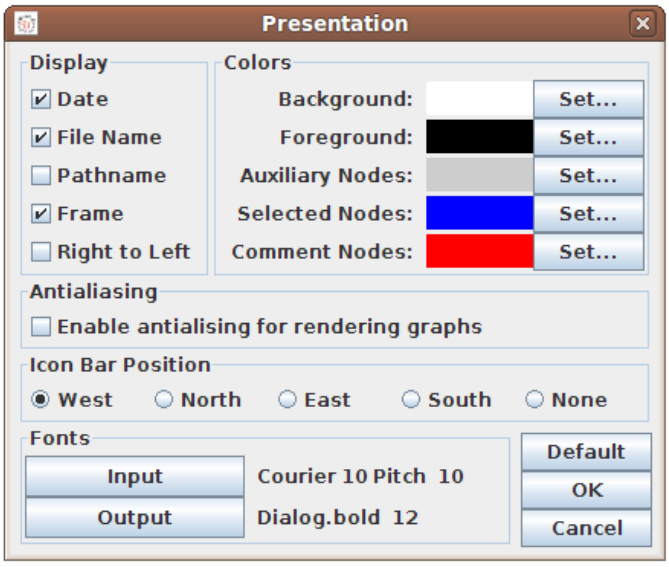


FIGURE 5.38 – Configuration de l'aspect d'un graphe

Les paramètres de couleur sont :

- Background : couleur de fond ;
- Foreground : couleur utilisée pour le texte et le dessin des boîtes ;
- 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 ;
- Pathname : 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 5.39).

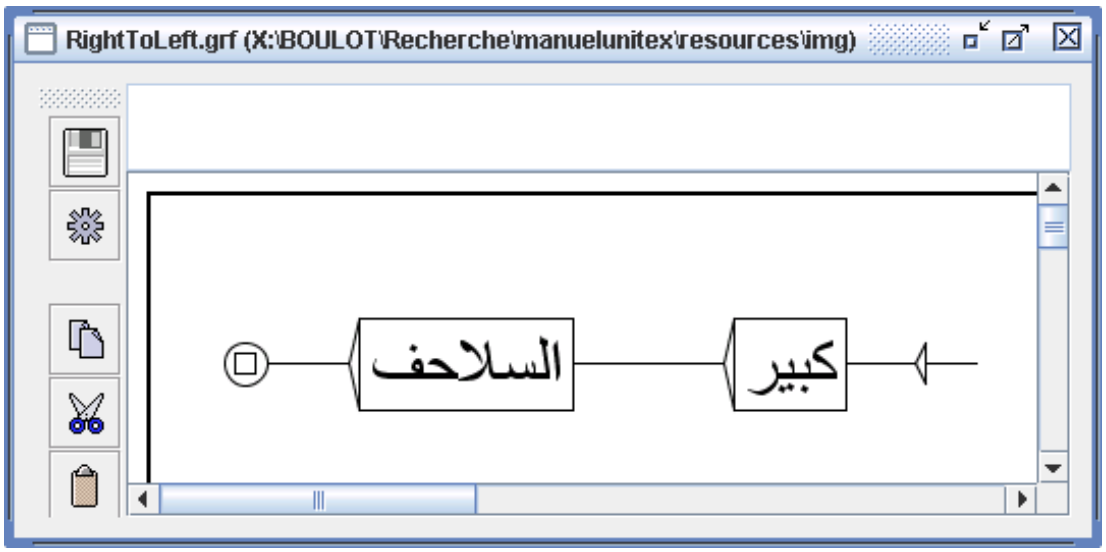


FIGURE 5.39 – Graphe se lisant de droite à gauche

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".

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, une première méthode consiste à exporter votre graphe vers un format d'image : PNG, JPEG ou SVG. Pour cela, allez dans le menu "FSGraph" et cliquez sur "Export as image". Choisissez ensuite le type de fichier. Vous obtiendrez ainsi une image prête à être intégrée dans un document ou à être éditée avec un logiciel de retouche d'images. Afin de rendre l'image plus lisse, vous pouvez activer l'antialiasing pour le graphe qui vous intéresse. Contraitement au JPEG, le format PNG utilise une compression sans perte de qualité, donc le PNG donne toujours un meilleurs résultat que le JPEG. Et contrairement au PNG et au JPEG, qui sont des fomats bitmap, le format SVG est un format vectoriel, ce qui permet souvent un meilleur résultat. A l'aide du logiciel Inkscape, il est également possible de convertir le fichier SVG en EPS ou en PDF, avec des lignes de commandes de ce type :

```
Inkscape -z -E graph.eps graph.svg

Inkscape -z -A graph.pdf graph.svg
```

La seconde méthode consiste à faire une capture d'écran :

Sous Windows :

Appuyez 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` peut 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.

Image vectorielle

Si vous préférez une image vectorielle, vous pouvez exporter votre graphe vers le format SVG, qui est utilisable avec des logiciels comme Inkscape ([24]). Il permet d'obtenir des sorties PostScript utilisables dans des documents LaTeX.

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 plusieurs types de graphes qui correspondent aux utilisations suivantes : flexion automatique de dictionnaires, prétraitement des textes, normalisation des automates de textes, graphes dictionnaires, recherche de motifs, levée d'ambiguïtés et génération automatique de graphes. 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.

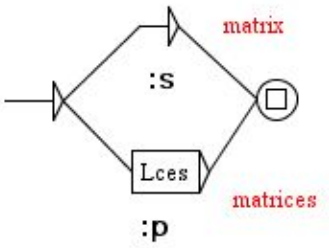


FIGURE 6.1 – Exemple de grammaire de flexion

Les chemins peuvent contenir des opérateurs et des lettres. Les opérateurs possibles sont représentés par les caractères `L`, `R`, `C`, `D`, `U`, `P` et `W`. Les lettres qui ne sont pas des opérateurs

sont des caractères. 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 cependant possible 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. 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).

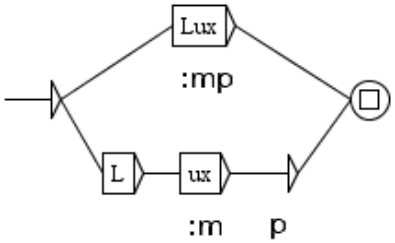


FIGURE 6.2 – Deux chemins équivalents dans une grammaire de flexion

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

Pour plus de détails, voir section 3.5.

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 ;
- si l'on travaille en mode caractère par caractère, il est possible d'utiliser le symbole spécial <L> qui reconnaît une lettre, telle que définie dans le fichier alphabet ;
- il est impossible de faire référence aux dictionnaires ;

- il est impossible d'utiliser les filtres morphologiques ;
- il est impossible d'utiliser le mode morphologique ;
- il est impossible d'utiliser des contextes.

Les figures 2.10 (page 34) et 2.11 (page 37) 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.

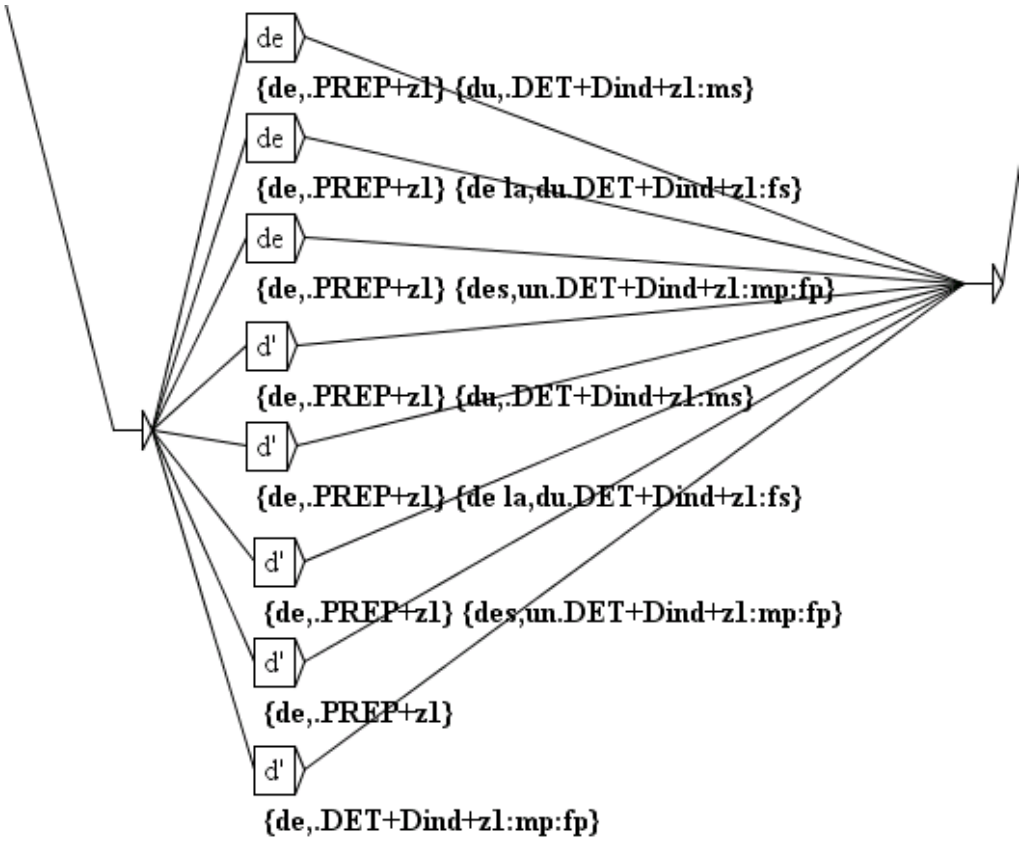


FIGURE 6.3 – Extrait du graphe de normalisation utilisé pour le français

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, également 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 5.2.2). 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 6.10.4).

Les graphes syntaxiques peuvent utiliser des contextes (voir section 6.3).

Les graphes syntaxiques peuvent utiliser des filtres morphologiques (voir section 4.7).

Les graphes syntaxiques peuvent utiliser le mode morphologique (voir section 6.4).

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 179.

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 9.

6.2 Compilation d'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).

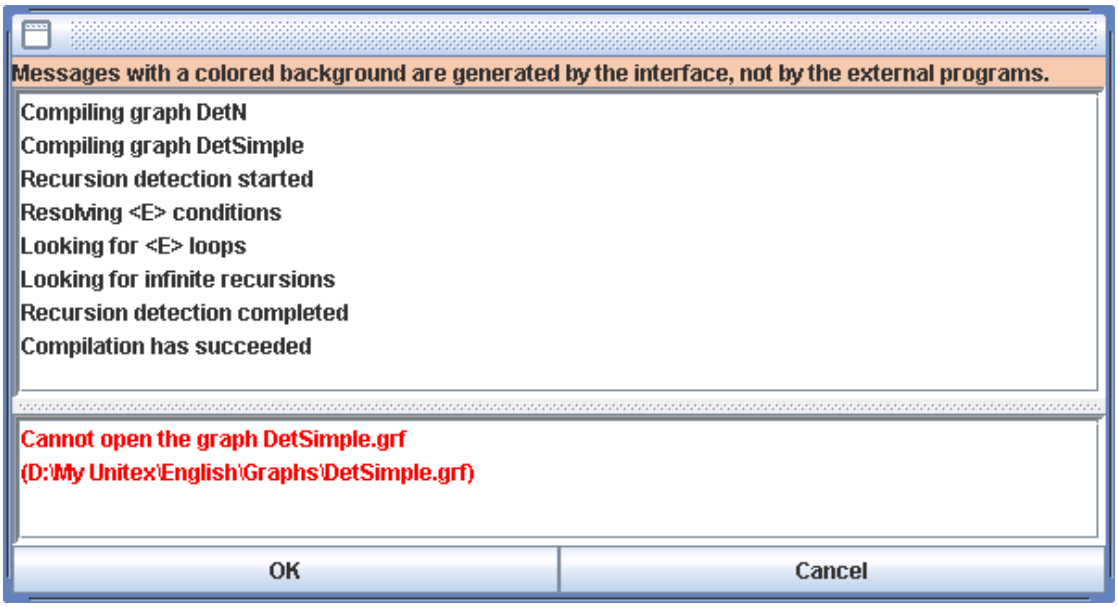


FIGURE 6.4 – Fenêtre de compilation

Si le graphe fait appel à des sous-graphes, ceux-ci sont automatiquement compilés. Le résultat est un fichier `.fst2` fichier 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 fini

Le format FST2 conserve l'architecture en sous-graphes des grammaires, ce qui les différencie des stricts transducteurs finis. Le programme `Flatten` permet de transformer une grammaire FST2 en un transducteur fini quand cela est possible, et d'en construire une approximation dans le cas contraire. 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.

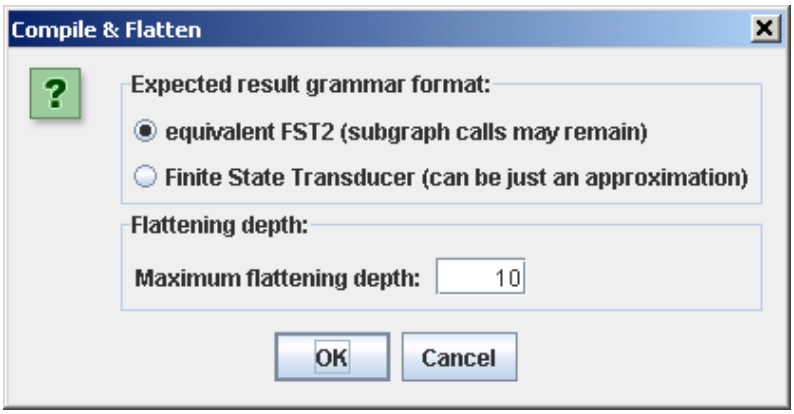


FIGURE 6.5 – Configuration de l'approximation d'une grammaire

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 Transducer", les appels aux sous-graphes seront ignorés (remplacés par `<E>`) au-delà de la profondeur maximale. Cette option garantit ainsi l'obtention d'un transducteur fini, 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 fini. 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 fini ou une grammaire FST2, et dans le cas d'un transducteur, s'il est équivalent à la grammaire d'origine (voir figure 6.6).

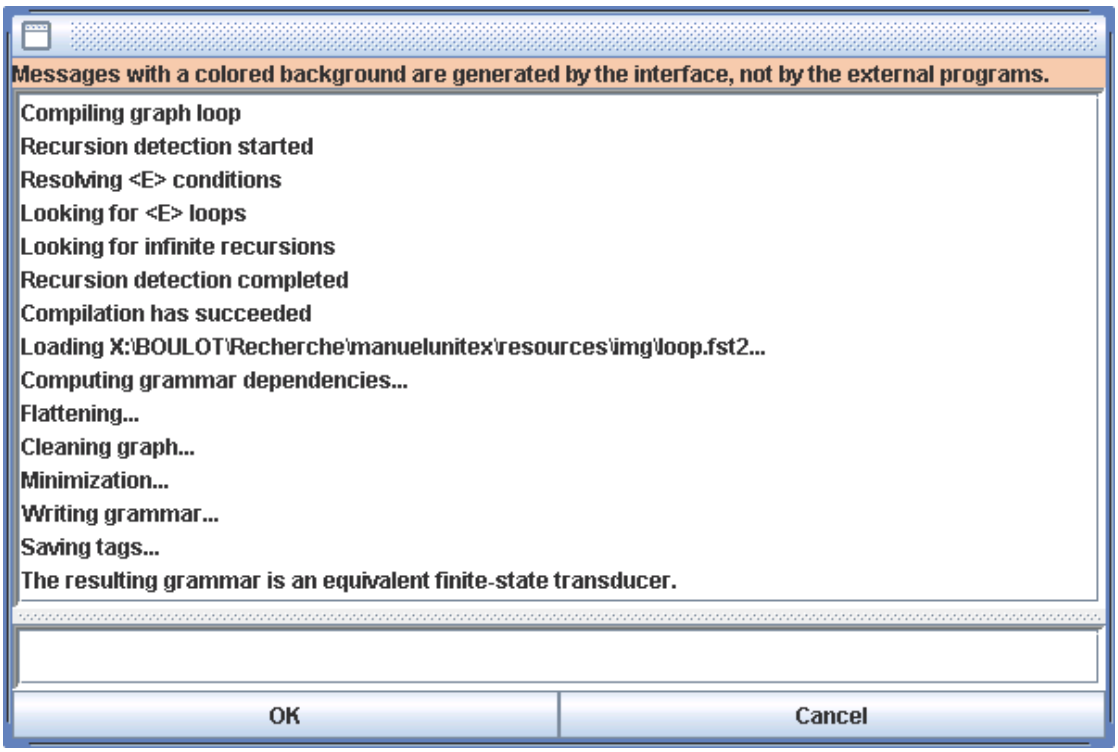


FIGURE 6.6 – Résultat de l'approximation d'une grammaire

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 vides, car les programmes d'Unitex ne pourraient jamais terminer l'exploration de telles grammaires. Une boucle vide peut faire entrer le programme `Locate` dans une boucle infinie. Les boucles vides peuvent être dues à des transitions étiquetées par le mot vide ou à certains appels de sous-graphes récursifs.

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

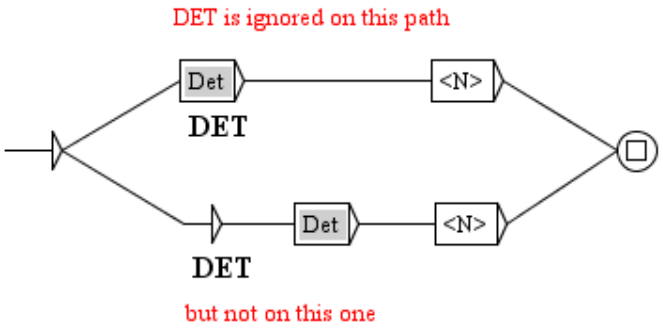


FIGURE 6.7 – Comment associer une sortie à un appel de sous-graphe

d'une sortie. Ainsi, la transition par le mot vide de la figure 6.8 ne sera pas supprimée et provoquera une boucle vide.

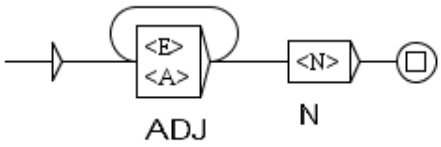


FIGURE 6.8 – Boucle vide due à une transition par le mot vide avec sortie

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` reconnait epsilon, on a une boucle vide qu'Unitex ne peut pas éliminer.

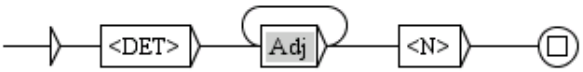


FIGURE 6.9 – Boucle vide due à un appel à un sous-graphe reconnaissant epsilon

La troisième possibilité de boucle vide vient de certains appels récursifs à des sous-graphes. 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 autre qu'epsilon 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

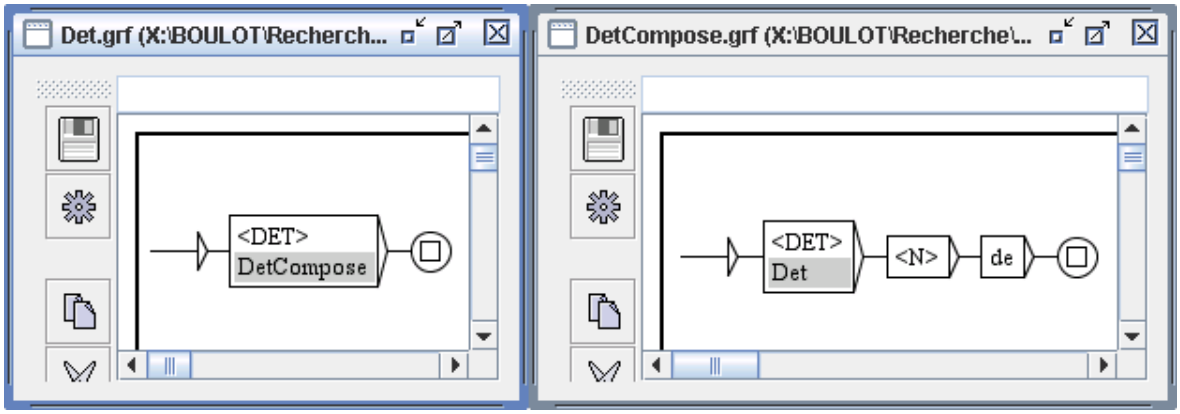


FIGURE 6.10 – Boucle vide causée par deux graphes s'appelant l'un l'autre

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.

6.2.4 Intervalle pour le nombre de répétitions

Pour reconnaître des séquences de tokens dans laquelle un motif apparaît une fois, plusieurs fois ou jamais, on peut associer un intervalle d'entiers à une boîte. Cela fixe les limites du nombre de fois que le motif apparait. Le motif doit être décrit dans une boite unique. Si on associe un intervalle [m,M] à une boîte contenant <A> (figure 6.11), le chemin reconnaitra des séquences avec au moins m adjectifs consécutifs et pas plus de M.

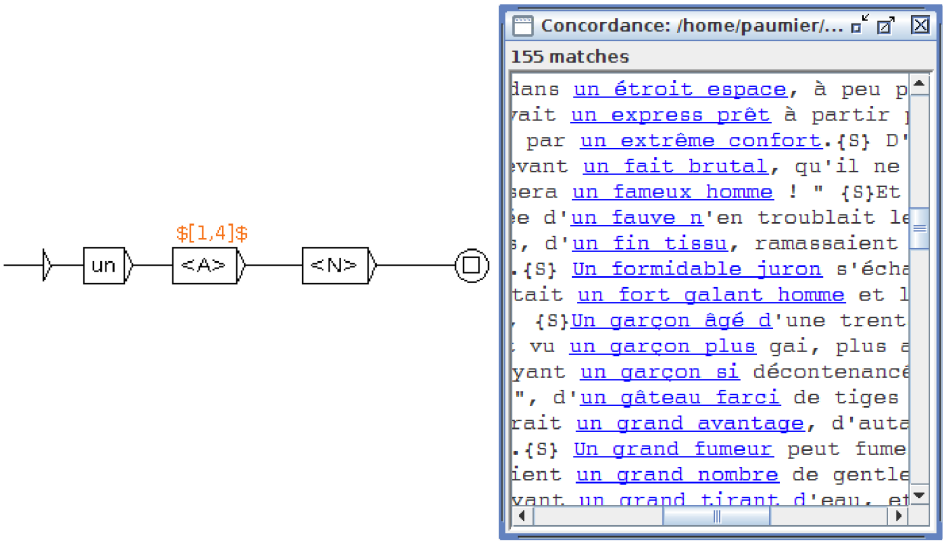


FIGURE 6.11 – Utilisation d'un intervalle pour reconnaître plusieurs tokens consécutifs

On attache un intervalle en insérant `$[m,M]$` dans la sortie de la boite, juste après le caractère "`/`", et selon les règles :

- `[m,M]` = au moins m motifs consécutifs et pas de plus de M
- `[,M]` = de 0 à M
- `[m,]` = au moins m

La boite ne doit pas être connectée à elle-même par une boucle directe. Un intervalle est compatible avec une sortie au sens habituel. Par exemple, pour insérer sous la boite de la figure 6.11 la sortie `<ADJ position='antéposé'>`, saisissez dans le champ texte : `<A>/$[1,4]$/<ADJ position='antéposé'>`.

6.2.5 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 6.12 montre le message obtenu lorsqu'on tente de compiler le graphe `Det` de la figure 6.10.

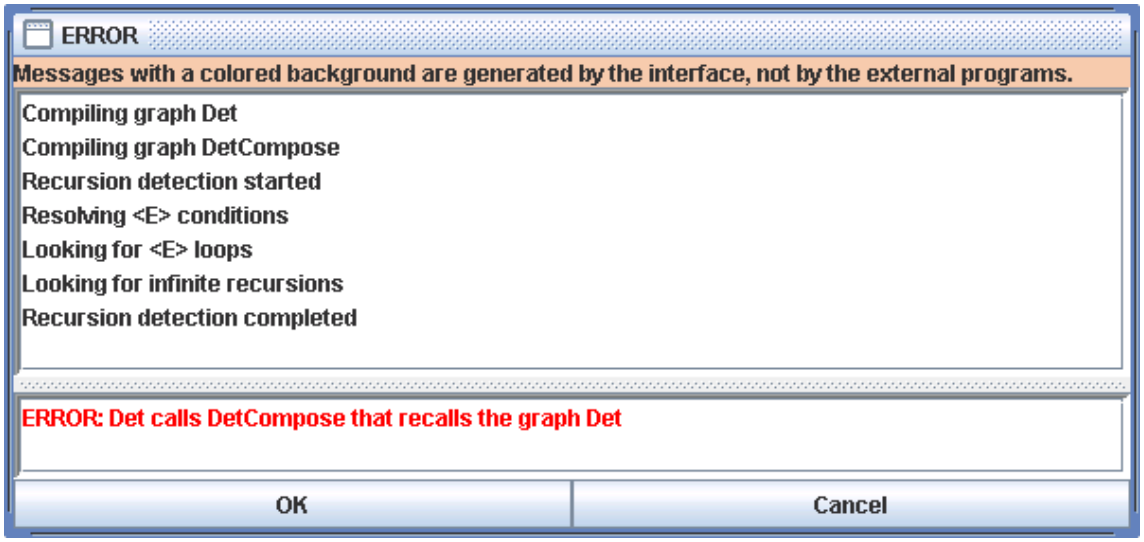


FIGURE 6.12 – Message d'erreur obtenu en compilant le graphe `Det`

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 Contextes

Les graphes d'Unitex sont des grammaires algébriques. Elles sont également appelées grammaires hors-contexte, car lorsque l'on souhaite reconnaître une séquence A, on ne tient pas compte du contexte dans lequel A apparaît. Par exemple, il est impossible de rechercher avec un graphe normal toutes les occurrences du mot `president`, sauf celles qui sont suivies par `of the republic`.

Il est toutefois possible de tenir compte du contexte dans les graphes syntaxiques. Dans ce cas, les graphes ne sont plus des grammaires algébriques, mais des grammaires contextuelles qui n'ont pas les mêmes propriétés théoriques. .

6.3.1 Contextes droits

On définit un contexte droit en délimitant une zone du graphe avec des boîtes contenant `$[` and `$]`, représentant respectivement les début et fin de contexte qui sont représentés dans le graphe par des crochets verts. Le début et la fin d'un contexte doivent apparaître dans le même graphe.

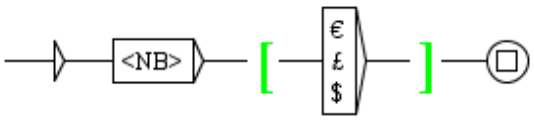


FIGURE 6.13 – Utilisation d'un contexte droit

La figure 6.13 montre un exemple simple de contexte. Ce graphe reconnaît tous les nombres qui sont suivis par l'euro, la livre ou le dollar, mais sans que le symbole d'unité n'apparaisse dans les occurrences trouvées, c'est-à-dire dans la concordance.

Les contextes s'interprètent de la façon suivante. Supposons que l'on rencontre un début de contexte lors de l'application d'une grammaire à un texte, et notons pos la position courante dans le texte à cet instant. Le programme `Locate` va ensuite chercher à reconnaître l'expression décrite dans le contexte. S'il échoue, il n'y aura pas de match. S'il réussit, c'est- à-dire s'il peut atteindre la fin du contexte, le programme reviendra à la position pos dans le texte et continuera l'exploration de la grammaire à partir de la fin du contexte.

Les poids (section 5.2.4) dans les contextes droits sont ignorés.

On peut également définir des contextes droits négatifs, en utilisant `$![` comme début de contexte. La figure 6.14 montre un graphe reconnaissant des nombres qui ne sont pas suivis par `th`. La différence avec les contextes positifs est que lorsque `Locate` essaie de reconnaître l'expression décrite dans le contexte, le fait d'atteindre la fin du contexte est considéré comme un échec, car cela signifie que l'on a reconnu une séquence interdite. À l'inverse, si

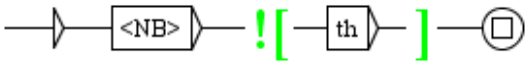


FIGURE 6.14 – Utilisation d'un contexte négatif

la fin de contexte ne peut être atteinte, le programme `Locate` reviendra à la position *pos* dans le texte et continuera l'exploration de la grammaire à partir de la fin du contexte.

Les contextes peuvent être placés n'importe où dans le graphe, y compris au début. La figure 6.15 montre ainsi un graphe qui reconnaît un adjectif dans le contexte de quelque chose qui n'est pas un participe passé. Autrement dit, ce graphe reconnaît tous les adjectifs qui ne sont pas ambigus avec des participes passés.

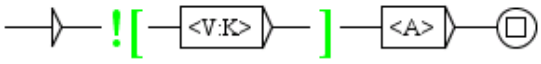


FIGURE 6.15 – Recherche d'un adjectif non ambigu avec un participe passé

Dans les graphes tels que celui de la figure 6.15, le contexte droit négatif ne vérifie pas nécessairement le même nombre de tokens que la boite qui le suit. Par exemple, avant que le graphe de la figure 6.16 ne reconnaisse `too`, le contexte droit négatif vérifie s'il apparait dans une expression telle que `too early` ou `too many`.

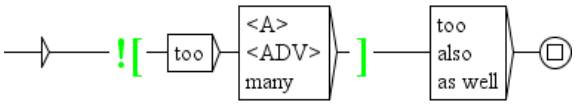


FIGURE 6.16 – Un contexte qui ne vérifie pas le même nombre de mots que la boite qui le suit

On peut formuler des requêtes complexes avec les contextes droits négatifs. Ainsi, la figure 6.17 montre un graphe qui reconnaît toutes les séquences de deux noms simples qui ne sont pas ambiguës avec des mots composés. En effet, le motif `<CDIC><<^([^ ]+ [^ ]+)$>>` reconnaît un mot composé contenant exactement un espace, et le motif `<N><<^([^ ]+)$>>` reconnaît un nom sans espace, c'est-à-dire un nom simple. Ainsi, dans la phrase *Black cats should like the town hall*, ce graphe reconnaîtra *Black cats*, mais pas *town hall*, qui est un mot composé.

Il est possible d'imbriquer des contextes. Par exemple, le graphe de la figure 6.18 reconnaît un nombre qui n'est pas suivi par un point, sauf si ce point est suivi par un nombre. Ainsi, dans le texte *5.0+7.=12*, ce graphe reconnaîtra *5*, *0* et *12*.

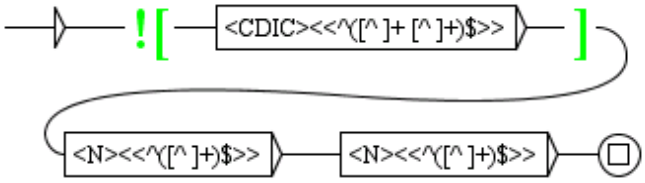


FIGURE 6.17 – Utilisation avancée des contextes

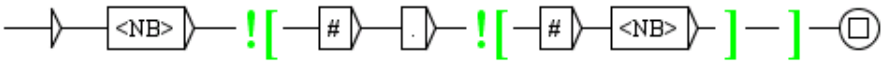


FIGURE 6.18 – Imbrication de contextes

Les sorties qui se trouvent dans des boîtes à l'intérieur d'un contexte sont ignorées. En revanche, il est possible d'utiliser une variable qui a été définie dans un contexte, comme c'est le cas sur la figure 6.19). Si l'on applique ce graphe en mode MERGE au texte *the cat is white*, on obtient en sortie :

```
the <pet name="cat" color="white"/> is white
```

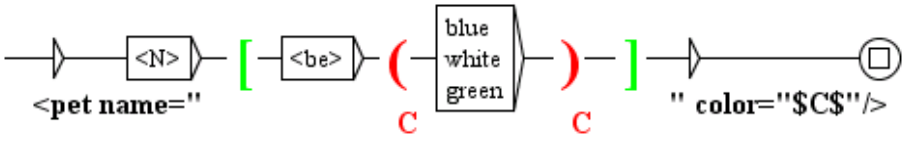


FIGURE 6.19 – Variable définie dans un contexte

6.3.2 Contextes gauches

Il est également possible de rechercher une expression X si elle se trouve seulement après une expression Y. Évidemment, il était déjà possible de le faire avec une grammaire semblable à celle de la figure 6.20. Cependant, avec ce type de grammaire, le contexte gauche est inclus dans la séquence reconnue, comme le montre la figure 6.21.

Pour éviter cela, on peut utiliser le symbole `$⋆` qui indique la fin du contexte gauche de l'expression qu'on désire reconnaître. Ce symbole est représenté par une étoile verte dans le graphe, comme le montre la figure 6.22. L'effet d'un tel contexte est d'utiliser une partie de la grammaire pour calculer la séquence reconnue, sans que cette partie ne figure dans le résultat (voir figure 6.23).

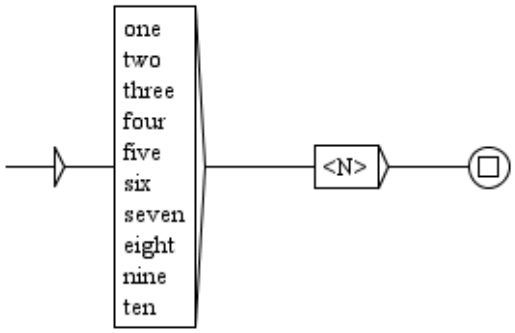


FIGURE 6.20 – Reconnaissance d'un nom précédé d'un déterminant numéral

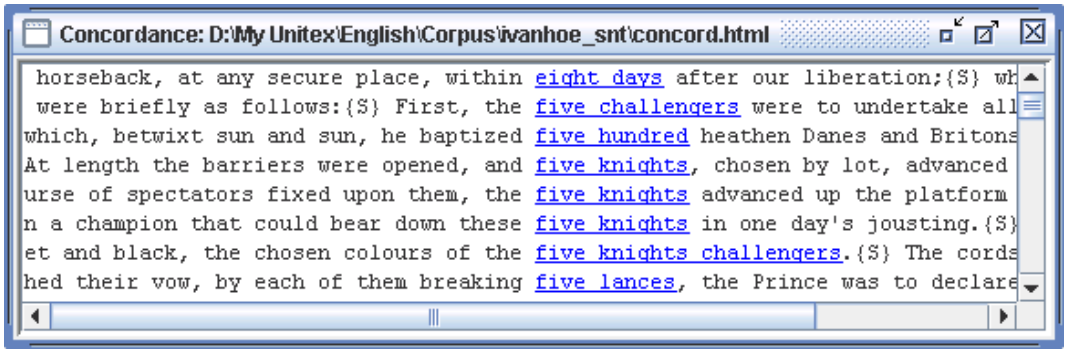


FIGURE 6.21 – Résultats de l'application de la grammaire de la figure 6.20

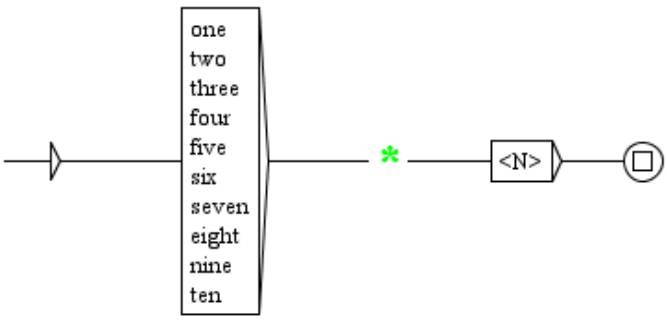


FIGURE 6.22 – Reconnaissance d'un nom après un contexte gauche

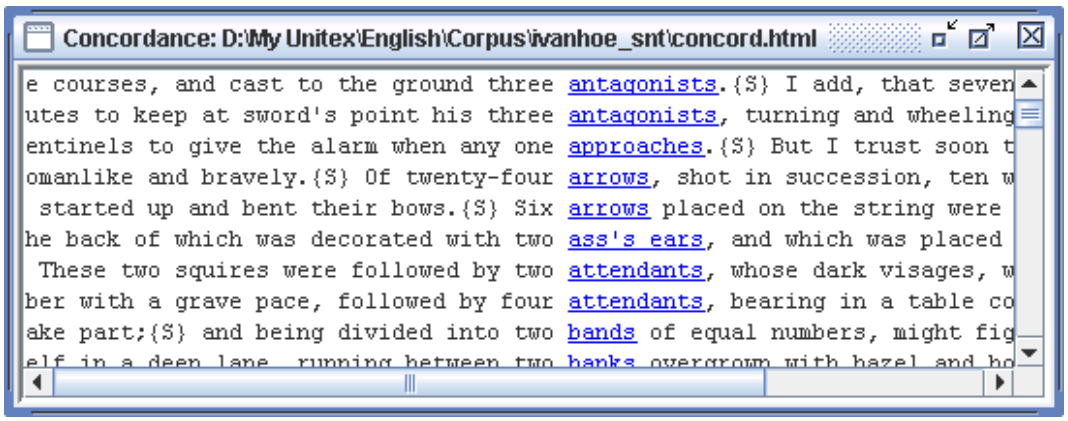


FIGURE 6.23 – Résultats de l'application de la grammaire de la figure 6.22

Toutes les sorties produites par un contexte gauche sont ignorées, comme on peut le voir dans la concordance de la figure 6.25, qui donne les résultats obtenus avec la grammaire de la figure 6.24.

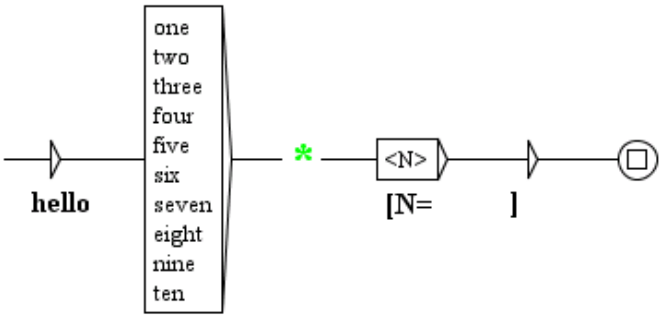


FIGURE 6.24 – Sorties ignorées dans un contexte gauche

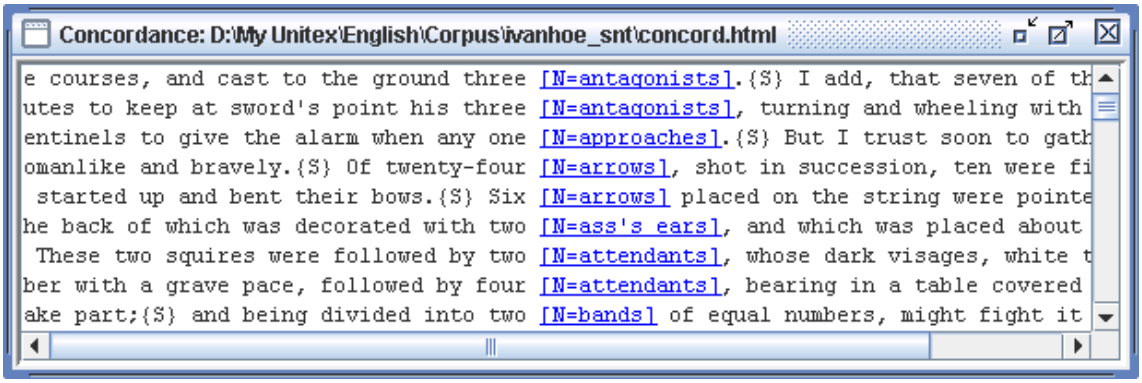


FIGURE 6.25 – Résultats de l'application de la grammaire de la figure 6.24

Toutefois, on peut mémoriser des informations avec des variables (voir section 6.7.5) et les utiliser en dehors du contexte gauche, comme le montrent la grammaire de la figure 6.26 et son résultat dans la figure 6.27.

On peut invoquer dans une grammaire un graphe qui contient des contextes gauches, mais cela nécessite d'être vigilant. Au moment où le contexte gauche est exclu de la séquence reconnue, toutes les séquences qui avaient été reconnues par des graphes appelants en sont exclues également, car la séquence qui sera finalement reconnue devra être contiguë. Les sorties correspondant aux séquences exclues sont ignorées elles aussi.

Ainsi, avec des contextes gauche et droit, on peut faire une distinction entre les motifs utilisés pour reconnaître des points du texte, et la délimitation des séquences à extraire dans

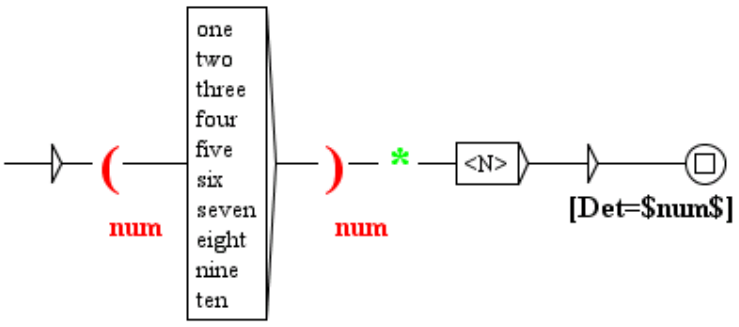


FIGURE 6.26 – Utilisation d'une variable dans un contexte gauche

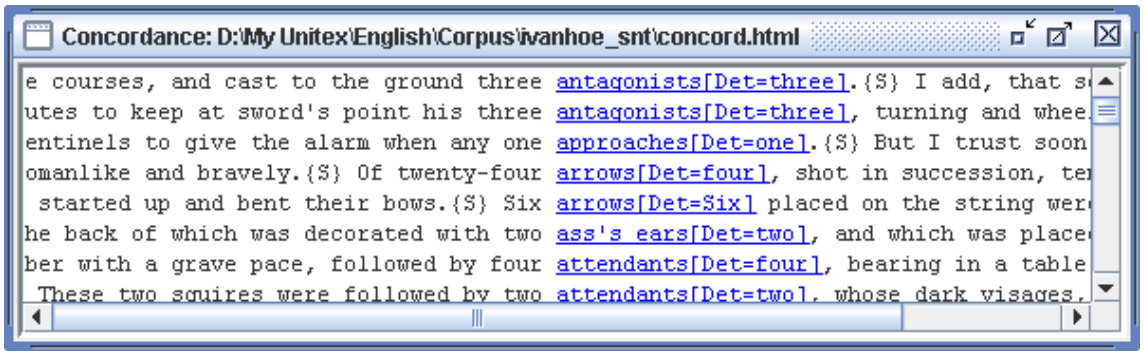


FIGURE 6.27 – Résultats de l'application de la grammaire de la figure 6.26

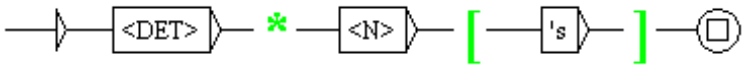


FIGURE 6.28 – Une grammaire avec des contextes gauche et droit

les résultats. Par exemple, la grammaire de la figure 6.28 cherche des expressions comme `the animal's`, mais extrait seulement les noms, comme on peut le voir figure 6.29.

Les poids (section 5.2.4) fonctionnent normalement dans les contextes gauches.

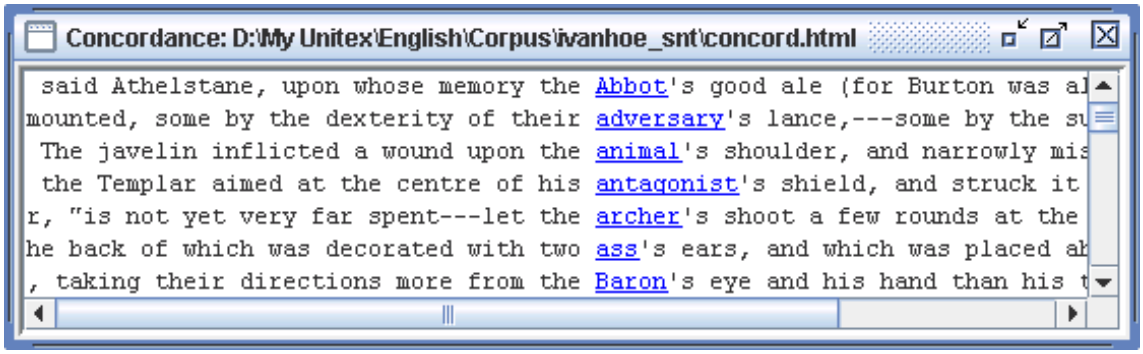


FIGURE 6.29 – Résultats de l'application de la grammaire de la figure 6.28

6.4 Le mode morphologique

6.4.1 Pourquoi ?

Comme Unitex fonctionne sur une version "tokenisée" du texte, il n'est pas possible de faire des requêtes qui entrent à l'intérieur des "tokens", sauf avec les filtres morphologiques (voir section 4.7), comme le montre la figure 6.30.

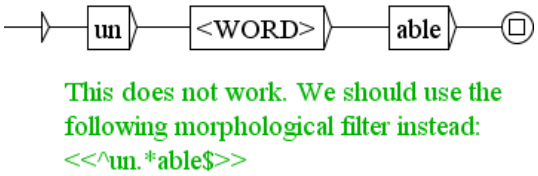


FIGURE 6.30 – Reconnaissance d'éléments morphologiques

Cependant, les filtres morphologiques ne permettent pas n'importe quelle requête, puisqu'ils ne peuvent pas faire référence aux informations contenues dans les dictionnaires. Ainsi, il est impossible de formuler de cette manière une requête comme "*un mot constitué du préfixe* `un` *suivi d'un adjectif en* `able`".

Pour surmonter cette difficulté, nous introduisons un mode morphologique dans le programme `Locate`. Il consiste à délimiter une partie de votre grammaire avec les symboles `$<` et `$>`. Dans cette zone, les données sont reconnues lettre par lettre, comme le montre la figure 6.31.

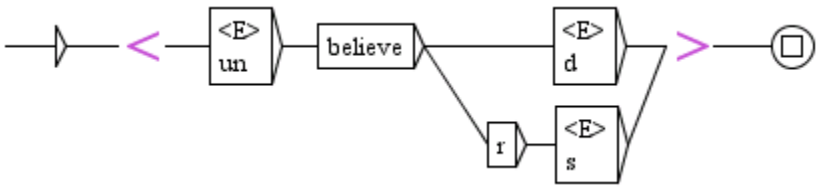


FIGURE 6.31 – Exemple de zone morphologique dans la grammaire

6.4.2 Les règles

Dans ce mode, le contenu du graphe n'est pas interprété de manière habituelle.

1. Il n'y a pas d'espace entre les boîtes. Ainsi, si on désire reconnaître un espace, on doit le rendre explicite avec `" "` (un espace entre guillemets).

2. On peut toujours utiliser des sous-graphes, mais la fin de la zone morphologique doit se trouver dans le même graphe que son début.

3. On peut utiliser des masques lexicaux qui nécessitent la consultation d'un dictionnaire — comme <DIC> , <be> ou <N:ms>, qui font référence aux informations contenues dans un dictionnaire —, du moment qu'il a été préalablement déclaré comme dictionnaire du mode morphologique (voir section 6.4.3).

4. On peut utiliser des masques lexicaux qui nécessitent la consultation d'un graphe-dictionnaire (section 3.8.3), du moment que le nom du graphe-dictionnaire contient l'option b. Cependant, cette possibilité ne fonctionne que pour les formes reconnues dans le texte par le graphe-dictionnaire pendant l'application initiale des dictionnaires (section 3.8), et non pour les formes qui n'apparaissent dans le texte que comme des parties de tokens.

5. On peut utiliser des filtres morphologiques (section 4.7). Cependant, les filtres morphologiques employés seuls ou sur <TOKEN> ne s'appliqueront seulement qu'au caractère courant. Par conséquent, les filtres comme <<[1-9][0-9]>> qui sont conçus pour reconnaître plus d'un caractère ne reconnaîtront jamais rien. En fait, dans le mode morphologique, les filtres morphologiques ne sont utiles que pour exprimer des négations comme <<[^aeiouy]>> (n'importe quel caractère qui n'est pas une voyelle).

6. Les contextes gauches et droits au sens de la section 6.3 sont interdits.

7. On peut utiliser des sorties.

8. <LETTER> reconnaît n'importe quelle lettre définie dans le fichier alphabet.

9. <LOWER> reconnaît n'importe quelle minuscule définie dans le fichier alphabet.

10. <UPPER> reconnaît n'importe quelle majuscule définie dans le fichier alphabet.

11. <DIC> reconnaît n'importe quel mot présent dans un dictionnaire du mode morphologique, mais les méta-symboles #, <FIRST>, <NB>, <SDIC> et<CDIC> sont interdits.

12. Si on atteint la fin de la zone sans être à la fin du token, la reconnaissance échoue. Par exemple, si le texte contient enabled, on ne peut pas reconnaître seulement enable.

Les anciens codes correspondant à <LETTER>, <LOWER> et <UPPER> étaient respectivement <MOT>, <MIN> et <MAJ>. Ils restent opérationnels afin de conserver la compatibilité descendante du système avec les graphes existants. Même s'il n'est pas prévu de supprimer ces codes, on recommande de les éviter dans les graphes conçus pour fonctionner avec les versions plus récentes[1], pour ne pas faire augmenter inutilement le nombre de masques lexicaux en usage.

6.4.3 Dictionnaires du mode morphologique

Dans le mode morphologique, on peut faire des requêtes qui utilisent les dictionnaires. Par exemple, la grammaire de la figure 6.32 cherche les mots constitués du préfixe un suivi d'un adjectif.

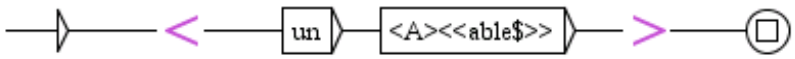


FIGURE 6.32 – Reconnaissance des mots constitués de *un* et d'un adjectif en *able*

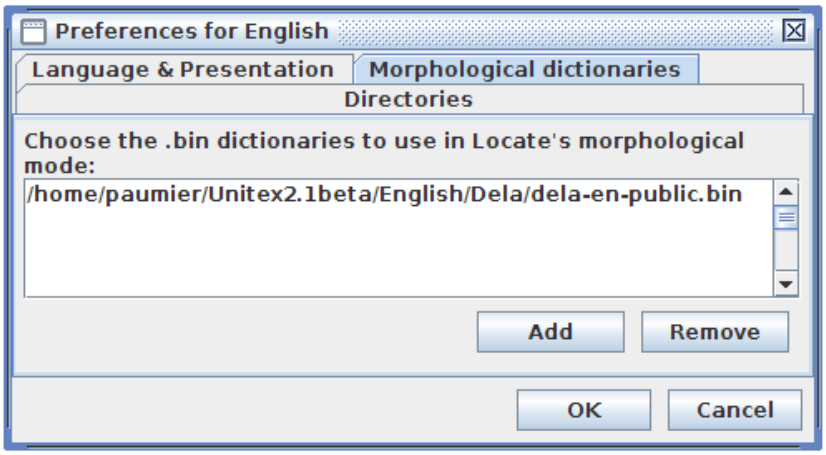


FIGURE 6.33 – Déclaration des dictionnaires du mode morphologique

Pour pouvoir reconnaître le mot `unaware` avec cette grammaire, le système doit savoir que `aware` est un adjectif. Le masque lexical `<A>` nécessite la consultation d'un dictionnaire. Mais `aware` peut ne pas être présent dans le texte, de sorte qu'on ne peut pas compter sur les dictionnaires du texte [2]. C'est la raison pour laquelle on doit définir une liste de dictionnaires à consulter en mode morphologique. Pour ce faire, on va dans "Info>Preferences>Morphological-mode dictionaries" (figure 6.33). On peut définir autant de dictionnaires du mode morphologique qu'on veut, mais ils doivent être au format `.bin`. Ceci fait, on peut appliquer la grammaire. Pour spécifier qu'un graphe-dictionnaire doit être consulté lorsqu'on est en mode morphologique, on utilise l'option `b` ou `z` (section 3.8.3, Exporter les entrées produites comme dictionnaire du mode morphologique).

6.4.4 Variables de dictionnaire

On peut affecter à des variables des informations issues des dictionnaires du mode morphologique. Ces variables sont appelées variables de dictionnaire ou variables morphologiques. L'initialisation d'une variable de ce type doit être associée à une boite contenant un motif qui fait référence à des informations contenues dans un dictionnaire du mode morphologique, à l'exception du motif `<DIC>`. On met `$xxx$` en sortie de la boîte, où `xxx` est

1. À partir de la version 3.1bêta, révision 4072 du 2 octobre 2015.
2. Les dictionnaires du texte sont compilés pendant l'application initiale des dictionnaires (section 3.8), non pas pendant la recherche de motifs.

un nom correct de variable (cf. section 5.2.5). Ceci affecte à une variable dénommée `xxx` l'entrée de dictionnaire reconnue par le motif. Dans la suite des chemins qui passent par la boite, on peut obtenir la forme fléchie, la forme canonique et les codes fournis par l'entrée avec `$xxx.INFLECTED$`, `$xxx.LEMMA$` et `$xxx.CODE$`, comme le montre la figure 6.34. On peut également utiliser les motifs suivants :

- `$xxx.CODE.GRAM$` : fournit seulement le premier code grammatical, censé être la catégorie grammaticale
- `$xxx.CODE.SEM$` : fournit tous les autres codes, séparés par des +, s'il en existe
- `$xxx.CODE.FLEX$` : fournit tous les codes flexionnels séparés par des `:`, s'il en existe
- `$xxx.CODE.ATTR=yyy$` renvoie la valeur d'une paire attribut-valeur contenue dans les codes sémantiques, c'est-à-dire la valeur `zzz` de l'attribut `yyy` s'il y figure un code sémantique de la forme `yyy=zzz`.

Les variables de dictionnaire peuvent être utilisées en dehors du mode morphologique, comme sur la figure 6.36. On peut effectuer des tests sur ces variables comme expliqué dans la section 6.7.5.

FIGURE 6.34 – Utilisation d'une variable de dictionnaire

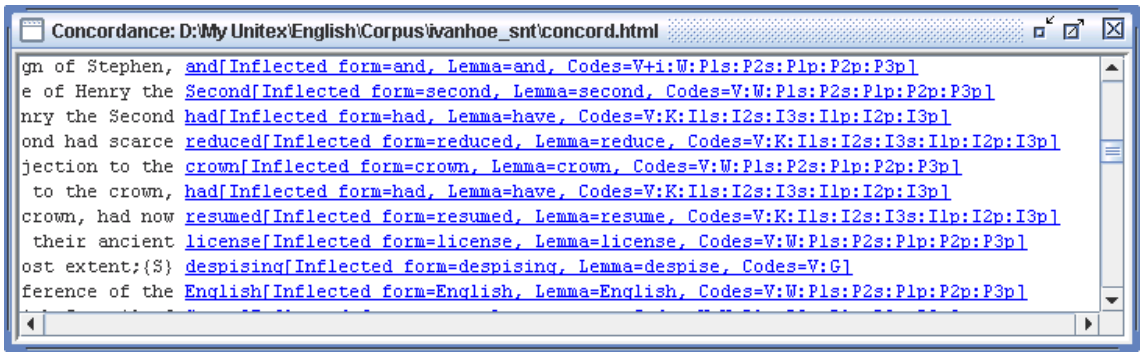


FIGURE 6.35 – Résultats de la grammaire de la figure 6.34 appliquée en mode in MERGE

Variables de dictionnaire dans LocateTfst

Pour les grammaires appliquées avec `LocateTfst` (cf. section 7.7), on dispose d'une possibilité supplémentaire. En dehors du mode morphologique, on peut mémoriser dans une

FIGURE 6.36 – Utilisation d'une variable de dictionnaire en mode normal

variable de dictionnaire une étiquette lexicale contenue dans l'automate du texte. Il suffit pour cela d'associer à la boîte une sortie de la forme `$:abc$` où `abc` est le nom de la variable. On peut ensuite l'utiliser comme variable de dictionnaire habituelle, de la façon décrite ci-dessus : on peut obtenir la forme fléchie, la forme canonique et les codes donnés dans l'entrée, sa catégorie grammaticale, ses codes sémantiques, ses codes flexionnels et la valeur `zzz` de l'attribut `yyy` s'il y figure un code sémantique de la forme `yyy=zzz`.

6.5 Exploration des chemins d'une grammaire

On peut générer la liste des chemins reconnus par une grammaire, par exemple pour vérifier qu'elle engendre correctement les formes attendues. Pour cela, on ouvre le graphe principal de la grammaire, on s'assure que la fenêtre du graphe est bien la fenêtre active (la fenêtre active possède une barre de titre bleue, tandis que les fenêtres inactives ont une barre de titre grise). On va ensuite dans le menu "FSGraph", puis dans le sous-menu "Tools", et on clique sur "Explore graph paths". La fenêtre de la figure 6.37 apparaît alors.

Le champ du haut contient le nom du graphe principal de la grammaire à explorer. Le deuxième champ indique le nom du fichier où le programme listera les chemins. Les options suivantes concernent la gestion des sorties des boites de la grammaire ainsi que le traitement des appels à des sous-graphes :

- "Outputs" : si "Ignore" est coché, les sorties sont ignorées ; avec "Split inputs and outputs", les sorties de toutes les boites d'un chemin sont affichées après les entrées des boites du même chemin (`a b c / A B C`) ; avec "Merge inputs and outputs", la sortie d'une boite est émise juste après l'entrée correspondante (`a/A b/B c/C`) ;

- "Explore subgraphs" : avec "Recursively", les appels aux sous-graphes sont explorés récursivement ; avec "Independently, printing names of called subgraphs", ils sont remplacés par le nom du sous-graphe, et les chemins reconnus par le sous-graphe sont explorés séparément, plus loin dans la liste.

Si l'option "Max sequences" est cochée, le nombre spécifié sera le nombre maximual de chemins générés. Si l'option n'est pas cochée, tous les chemins seront générés à condition qu'ils soient en nombre fini.

Avec "Flatten graphs", le programme remplace la grammaire par un transducteur fini ou si nécessaire par une approximation (voir section 6.2.2) avant de l'explorer.

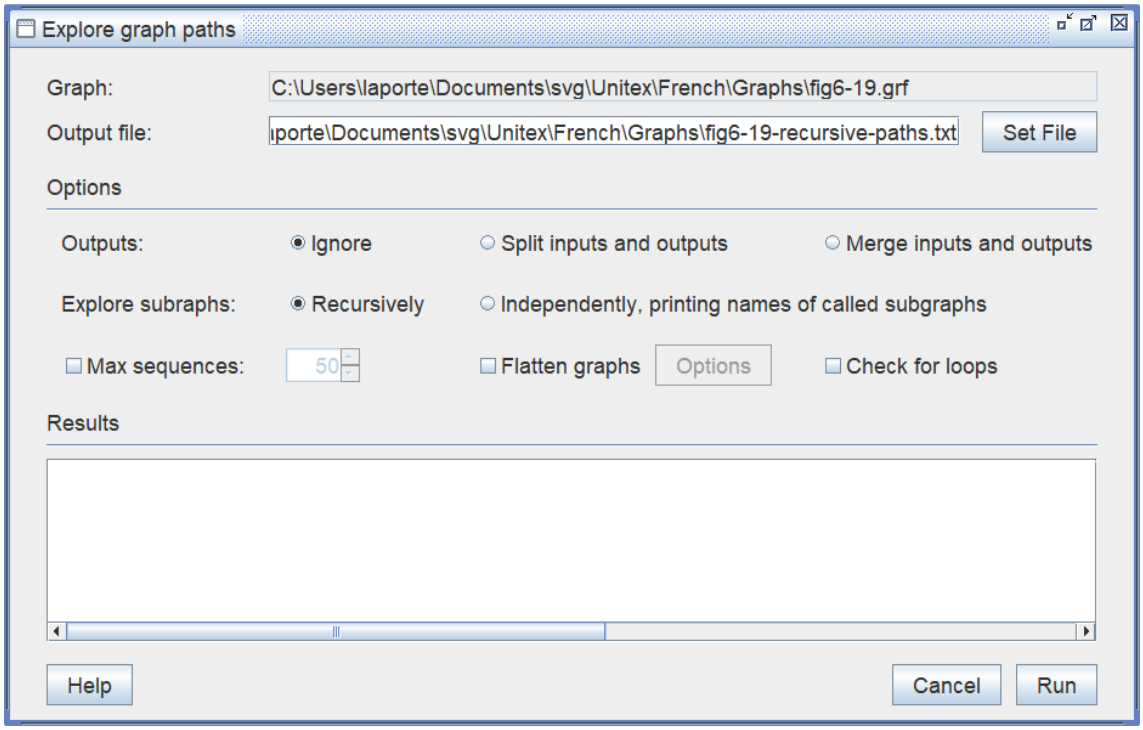


FIGURE 6.37 – Exploration des chemins d'une grammaire

Voici ce que l'on obtient pour le graphe de la figure 6.38 avec les paramètres par défaut (ignorer les sorties, pas de limite sur le nombre de 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
```

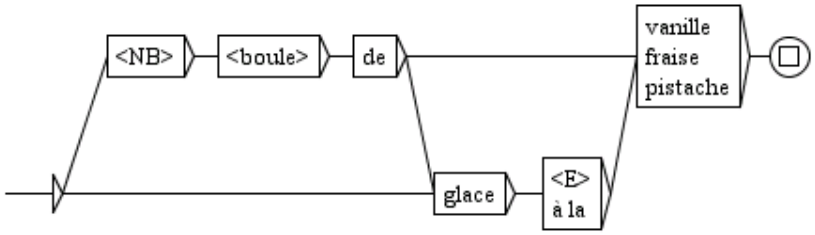


FIGURE 6.38 – Exemple de graphe

6.6 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 6.39.

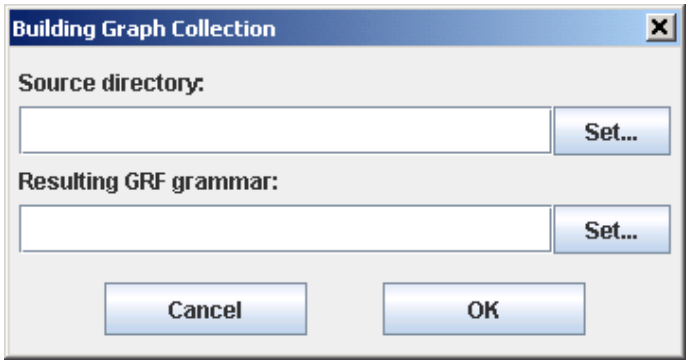


FIGURE 6.39 – Construction d'une collection de graphes

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 à 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 6.40, 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`).

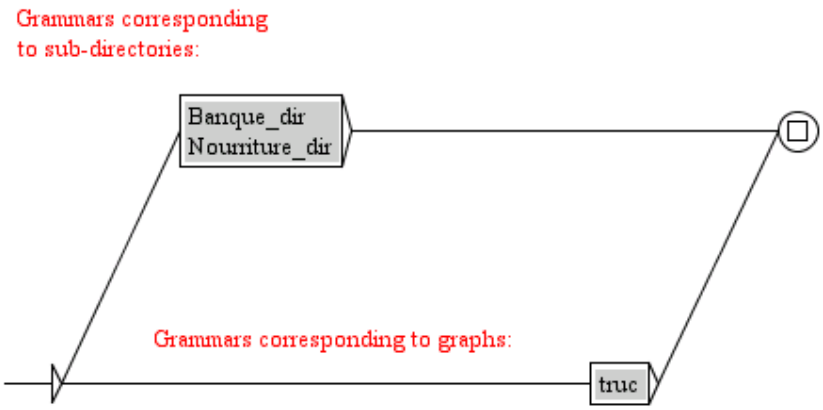


FIGURE 6.40 – Graphe principal d'une collection de graphes

6.7 Règles d'application des transducteurs

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

6.7.1 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 6.41 .

Si l'on applique ce transducteur au roman *Ivanhoe* by Sir Walter Scott en mode MERGE, on obtient la concordance de la figure 6.42

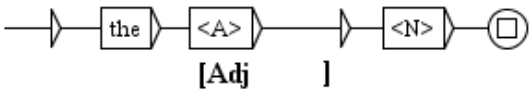


FIGURE 6.41 – Exemple de transducteur

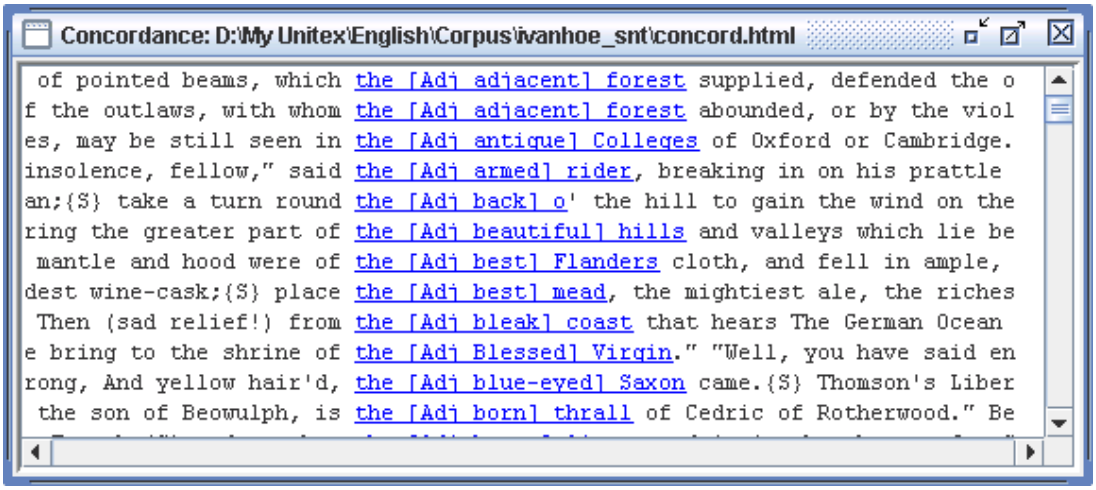


FIGURE 6.42 – Concordance obtenue en mode MERGE avec le transducteur de la figure 6.41

6.7.2 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.

6.7.3 Priorité à gauche

Lors de l'application d'une grammaire locale, les occurrences qui se chevauchent sont toutes indexées. Nous considérons, ici, de vrai chevauchements d'occurrence comme `abc` et `bcd`, et pas d'occurrences imbriquées comme `abc` et `bc`. Lors de la construction de la concordance, toutes ces occurrences sont présentées (voir figure 6.43).
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

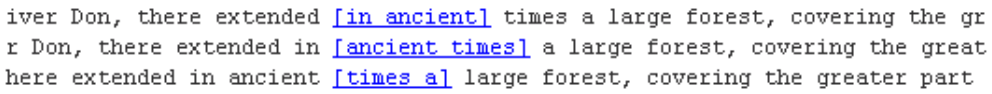


FIGURE 6.43 – Occurrences se chevauchant dans une concordance

`[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 6.10.4).

6.7.4 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.

6.7.5 Sorties à variables

Comme nous l'avons vu à la section 5.2.5, il est possible d'utiliser des variables d'entrée pour mémoriser le texte qui a été analysé par une grammaire. Ces variables peuvent être utilisées dans les graphes de prétraitement et dans les graphes syntaxiques.

On doit donner des noms aux variables qu'on utilise. 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 et la fin de la zone à stocker dans une variable d'entrée, soit on utilise le bouton avec les parenthèses rouges dans la barre d'icônes au-dessus du graphe (section 5.2.8), soit on crée deux boîtes, l'une contenant le nom de la variable encadré par les caractères `$` et `(` pour le début de la zone, et l'autre par `$` et `)` pour la fin. Pour utiliser une variable dans une sortie, on fait précéder et suivre son nom du caractère `$` (voir figure 6.44).

Les variables sont globales. Cela signifie qu'on peut définir une variable dans un graphe et l'appeler dans un autre, comme l'illustrent les graphes de la figure 6.44. Si on applique le graphe `TitleName` en mode MERGE au texte *Ivanhoe*, on obtient la concordance de la figure 6.45.

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 intervertir 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 6.46 appliqué en mode REPLACE au texte *Ivanhoe* donne la concordance de la figure 6.47.

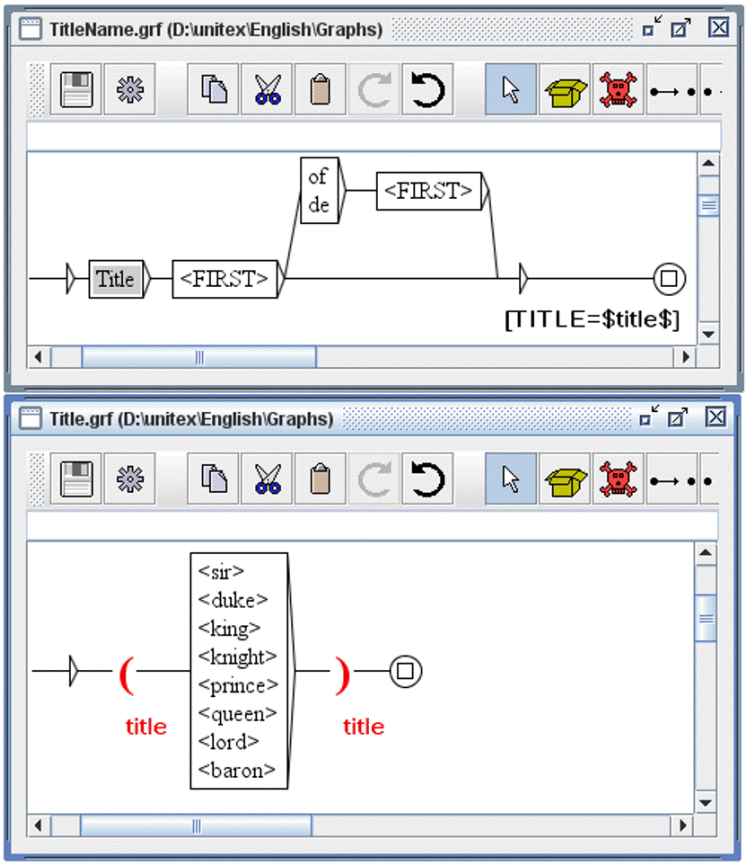


FIGURE 6.44 – Définition d'une variable d'entrée dans un sous-graphe

Si le début ou la fin d'une variable est mal défini (fin d'une variable avant son début, absence du début ou de la fin d'une variable), celle-ci sera ignorée lors des sorties. Consultez la section 6.10.2 pour d'autres options affectant le traitement d'erreurs concernant les variables.

Il n'y a aucune limite au nombre de variables utilisables.

Les variables d'entrées peuvent être imbriquées, et même se chevaucher comme le montre la figure 6.48.

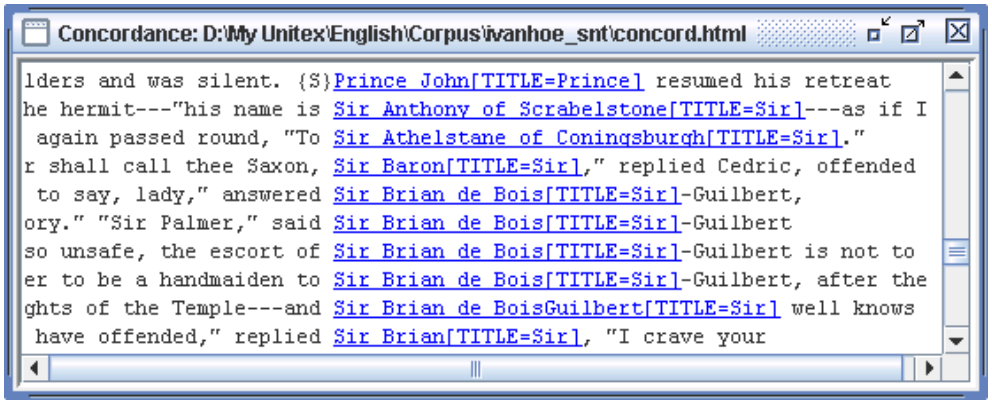


FIGURE 6.45 – Concordance obtenue par l'application du graphe `TitleName` de la fig. 6.44

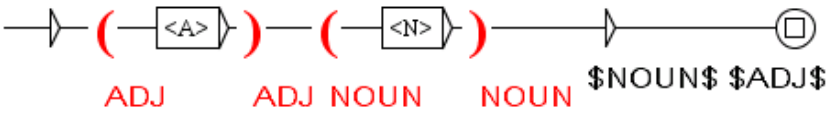


FIGURE 6.46 – Interversion de mots grâce à deux variables d'entrée

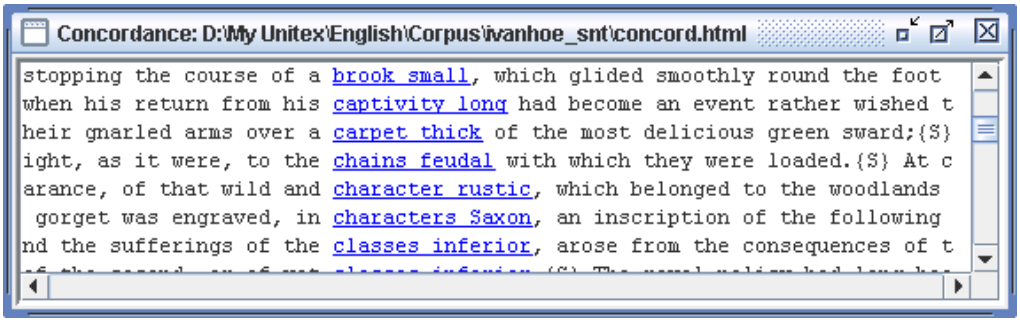


FIGURE 6.47 – Résultat de l'application du transducteur de la figure 6.46

6.8 Variables de sortie

Les variables d'entrée sont déclarées soit avec les parenthèses rouges de la barre d'icônes, soit avec `$xxx(` et `$xxx)`, et mémorisent des portions du texte d'entrée. Il est aussi possible de mémoriser des parties des sorties produites par une grammaire. Cela met en jeu des variables de sortie. Ces variables sont déclarées soit avec l'icône des parenthèses bleues dans la barre d'icônes au-dessus du graphe (section 5.2.8), soit avec `$|xxx(` et `$|xxx)`. Elles apparaissent en bleu (voir figure 6.49). Cette grammaire appliquée en mode MERGE au texte *Ivanhoe* produit la concordance visible sur la figure 6.50.

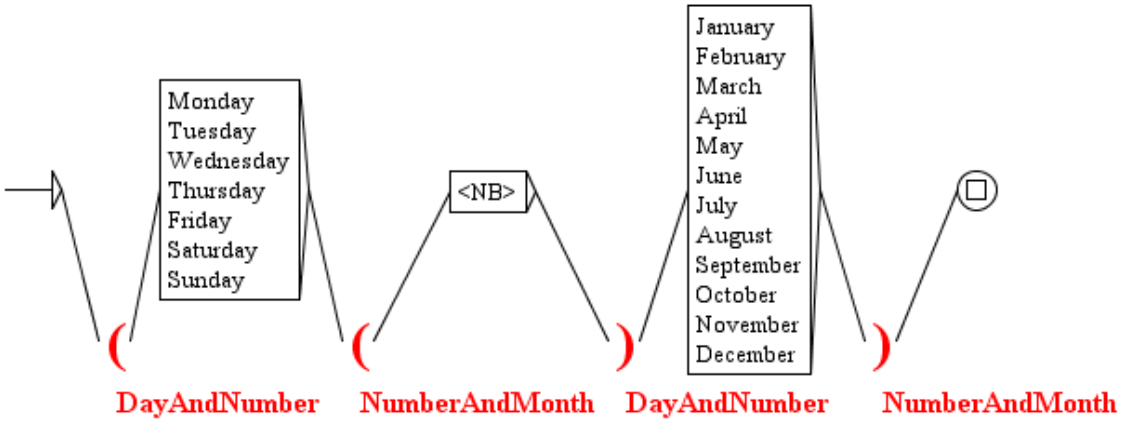


FIGURE 6.48 – Chevauchement de variables d'entrée

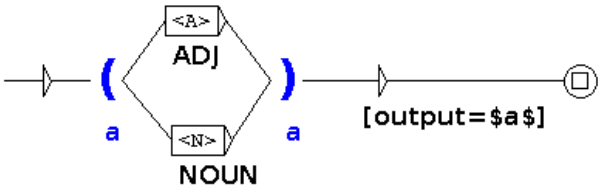


FIGURE 6.49 – Variables de sortie

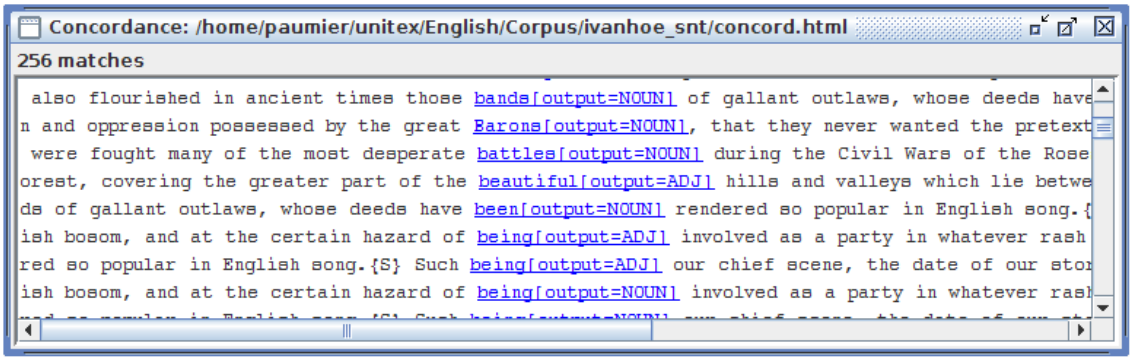


FIGURE 6.50 – Concordances obtenues avec la grammaire de la figure 6.49

Au moment où une variable de sortie est initialisée, les séquences de sortie du transducteur ne sont pas émises dans la sortie correspondant à l'occurrence courante, elles sont seulement mémorisées dans la variable de sortie créée par cette opération. Par exemple, les sorties ADJ et NOUN de la figure figure 6.49 n'ont pas été insérées à gauche du texte d'entrée dans la figure 6.50. Par ailleurs, les sorties sont traitées avant d'être mémorisées : si la sortie d'une

boite contient une chaine comme `$A.LEMMA$`, la variable de sortie ne contiendra en fait pas cette chaîne mais le lemme associé à la variable `A`.

Les variables de sortie mémorisent seulement des sorties effectivement produites par la grammaire. Ainsi, même en mode MERGE, les variables de sortie ne mémorisent jamais le texte d'entrée (figures 6.49 et 6.50).

Quand une boite redéfinit une variable qui avait déjà été définie, la nouvelle valeur écrase l'ancienne. Ainsi, si la variable est définie dans une boucle, la valeur de la variable juste après la boucle dépend du dernier passage dans la boucle.

6.9 Opérations sur les variables

6.9.1 Tests sur les variables

Il est possible de tester si une variable est définie ou non, afin d'interrompre la reconnaissance courante si la condition n'est pas vérifiée. Ceci se fait en insérant la séquence `$xxx.SET$` dans la sortie d'une boîte. Ainsi, si une variable dénommée `xxx` a été définie, cette séquence est ignorée et la reconnaissance continue, sinon, la reconnaissance s'arrête et le programme repart en arrière. Ceci fonctionne sur les variables d'entrée, les variables de sortie et les variables de dictionnaire. De façon similaire, on peut vérifier qu'une variable n'est pas définie en utilisant `$xxx.UNSET$`. La figure 6.51 montre un graphe qui utilise ce type de test. La figure 6.52 montre les résultats obtenus par ce graphe en mode MERGE.

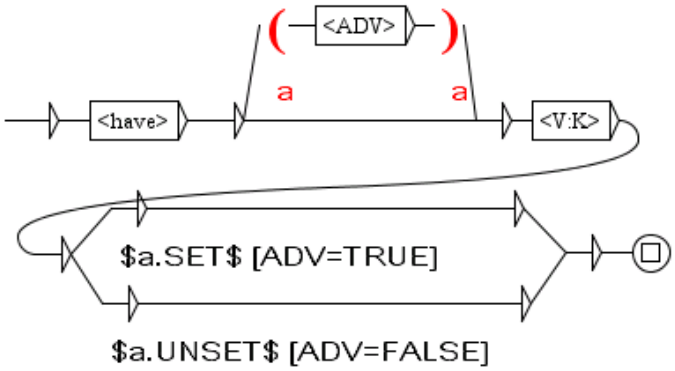


FIGURE 6.51 – Test d'une variable

6.9.2 Comparaison de variables

Il est également possible de comparer tout type de variable (d'entrée, de sortie, ou de dictionnaire) avec une constante ou une autre variable. Ceci se fait en insérant dans la sortie d'une boîte une séquence respectant la syntaxe suivante :

```
$abc.EQUAL=xyz$
```

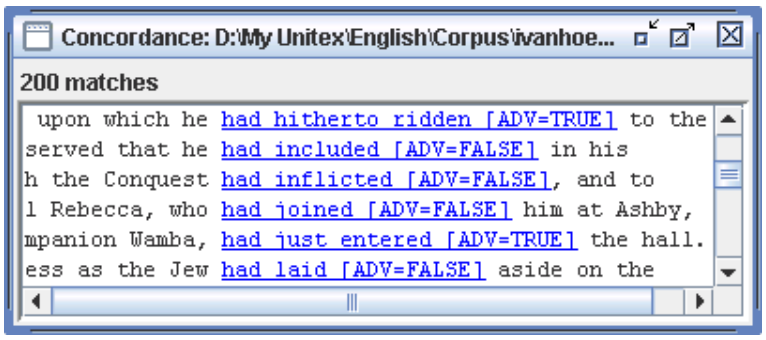


FIGURE 6.52 – Résultats d'un test de variable

Cela agit comme un interrupteur qui permet de bloquer l'exploration de grammaire si la valeur de la variable `abc` est différente de la valeur de la variable `xyz`. Remarquons que pour les variables de dictionnaire, c'est la forme fléchie telle qu'elle existe dans le dictionnaire (attention aux variantes de casse !) qui est utilisée pour le test. Si vous désirez comparer la variable `abc` à la constante `JKL`, utilisez le test suivant :

```
$abc.EQUAL=#JKL$
```

on peut également tester si le contenu est différent avec `UNEQUAL`.

Si vous désirez comparer des variables en ignorant les variantes de casse, vous pouvez utiliser les tests suivants :

```
$abc.EQUALcC=xyz$
```
ou
```
$abc.UNEQUALcC=xyz$
```

6.9.3 Recherche d'un code sémantique dans une variable de dictionnaire

On peut chercher dans une variable de dictionnaire (section 6.4.4) un "code sémantique" au sens de la section 3.1.1. Pour cela, on insère dans la sortie d'une boite une séquence respectant la syntaxe suivante :

```
$abc.EQ=Conc$
```

Ce test agit comme un interrupteur qui permet de bloquer l'exploration de la grammaire si `Conc` ne figure pas parmi les "codes sémantiques" de la variable de dictionnaire `abc`. On peut chercher un seul code à la fois dans une variable. Pour vérifier plusieurs codes, on met plusieurs boites en série.

Cette fonctionnalité est utilisée pour de grandes grammaires de graphes-dictionnaires morphologiques, en vue de dissocier dans des boites distinctes la vérification d'un code grammatical et de "codes sémantiques" qui viennent ensuite, comme dans [79], page 486. On teste

le code grammatical avec un masque lexical, puis on fait de même pour les codes sémantiques en les cherchant dans la variable de dictionnaire correspondante. Cette dissociation peut accélérer l'application des graphes si :

- tous les graphes sont invoqués directement ou indirectement depuis un même graphe principal,
- le graphe principal est compilé et transformé en transducteur fini (voir section 6.2.2),
- la boite qui contient le masque lexical est commune à plus de chemins que celles qui cherchent les codes sémantiques dans la variable de dictionnaire [3].

6.10 Application des graphes aux textes

Cette section concerne uniquement les graphes syntaxiques.

6.10.1 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 6.53.
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` format (Unicode Compiled Graphs). Si votre graphe est au format `.grf`, Unitex le compilera automatiquement avant de lancer la recherche. Si vous cliquez sur "Activate debug mode", la concordance sera affichée dans une fenêtre dans laquelle vous trouverez l'automate et, pour chaque séquence reconnue, la liste des états du chemin qui la reconnaît. Cette fenêtre est décrite en détails à la section 6.10.7.

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.

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

3. De cette façon, le masque lexical provoque une consultation des dictionnaires du mode morphologique qui n'est effectuée qu'une fois avant plusieurs recherches de codes sémantiques. Si on vérifie le code grammatical et un code sémantique par un même masque lexical, ces masques deviennent plus nombreux dans l'ensemble de la grammaire et ils provoquent plus de consultations des dictionnaires.

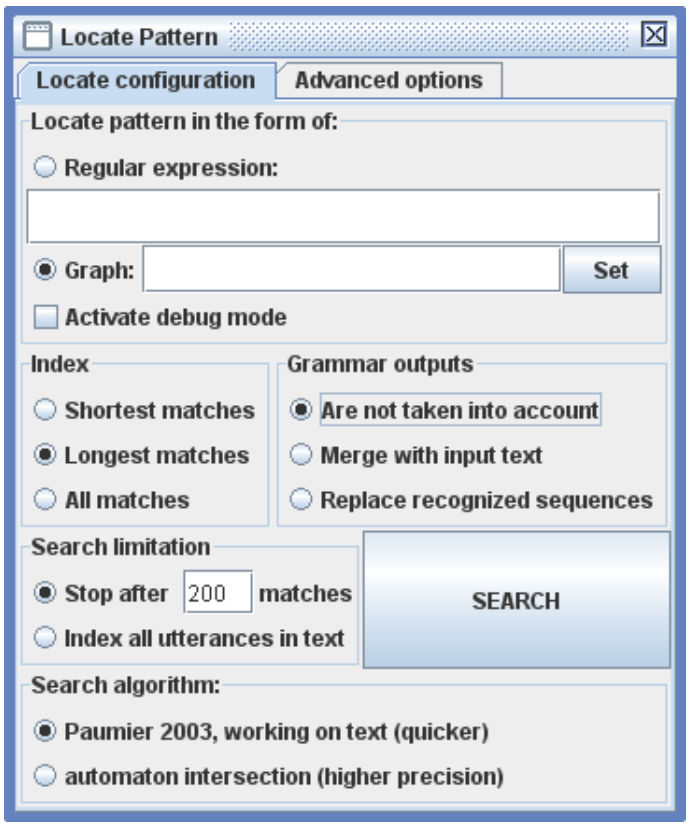


FIGURE 6.53 – Fenêtre de recherche d'expressions

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.

Dans le cadre "Search algorithm", vous pouvez spécifier si vous voulez effectuer la recherche sur le texte en utilisant le programme Locate ou sur l'automate du texte avec LocateTfst. Par défaut, la recherche est faite avec le programme Locate, comme Unitex l'a toujours fait jusqu'à maintenant. Si vous désirez utiliser LocateTfst, lisez la section 7.7.

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

6.10.2 Options de recherche avancées

Si vous sélectionnez l'onglet "Advanced options", vous voyez le cadre de la figure 6.54.

L'option "Ambiguous output policy" est illustrée par le graphe de la figure 6.55. Lorsqu'un déterminant est suivi par un mot pouvant être un nom ou un adjectif, il peut produire deux sorties distinctes pour la même séquence d'entrée (le transducteur est dit ambigu).

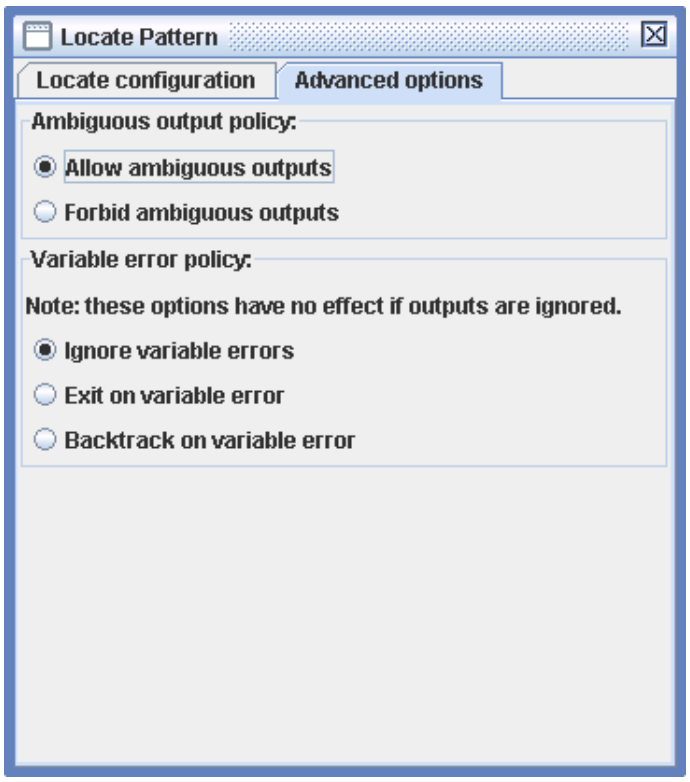


FIGURE 6.54 – Options de recherche avancées

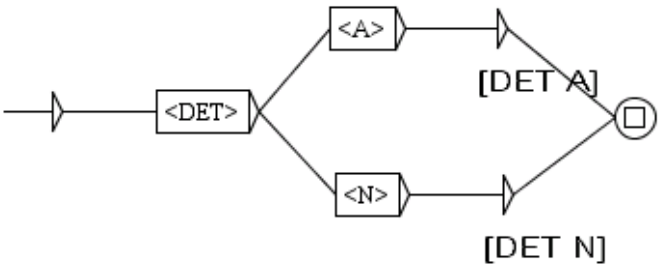


FIGURE 6.55 – Graphe avec des sorties ambiguës

Si nous appliquons ce graphe sur le texte *Ivanhoe* avec l'option "Allow ambiguous outputs" (celle par défaut), nous obtenons la concordance de la figure 6.56. Comme vous pouvez le constater, deux sorties sont produites pour la séquence *the noble*.

A l'opposé, avec l'option "Forbid ambiguous outputs", nous obtenons la concordance de la figure 6.57, avec seulement une sortie choisie arbitrairement pour la séquence *the noble*.

L'option "Variable error policy" permet de définir le comportement de `Locate/LocateTfst`

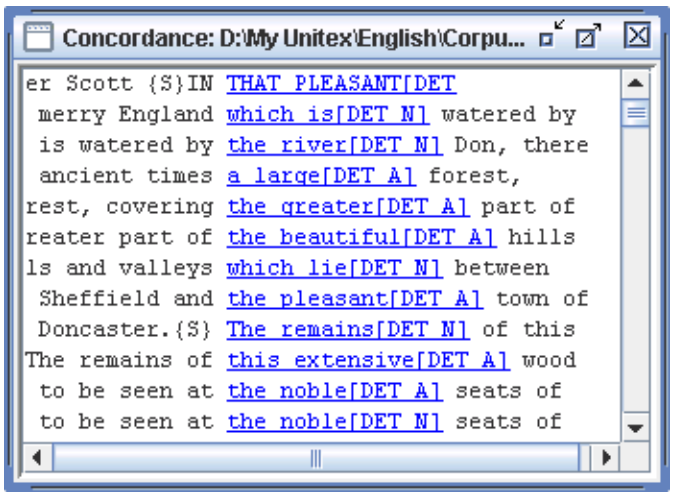


FIGURE 6.56 – Sorties ambiguës pour *the noble*

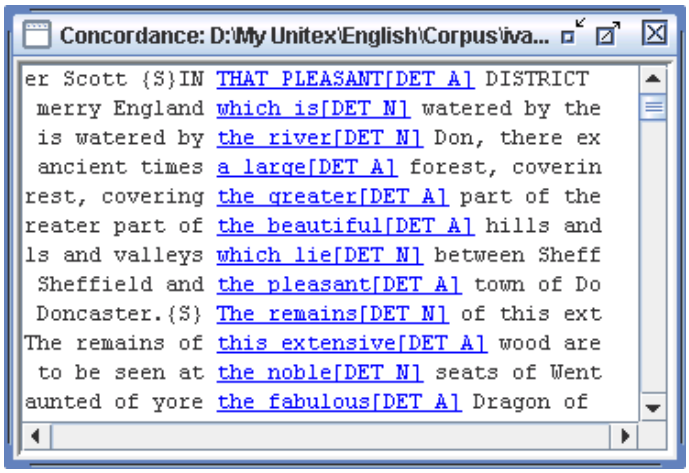


FIGURE 6.57 – Sortie unique *the noble*

lorsqu'ils rencontrent une sortie contenant une variable mal définie. Remarquons que ce paramètre n'a aucun effet si les sorties sont ignorées. Considérons par exemple le graphe de la figure 6.58.

Avec l'option "Ignore variable errors", *A* est ignorée, comme si son contenu était vide, comme le montre la figure 6.59.

Avec l'option "Exit on variable error", `Locate/LocateTfst` émettent un message d'erreur, comme le montre la figure 6.60.

Avec l'option "Backtrack on variable error" , `Locate/LocateTfst` arrête l'exploration du chemin courant de la grammaire. Ainsi, les variables jouent le rôle d'interrupteurs qui coupent

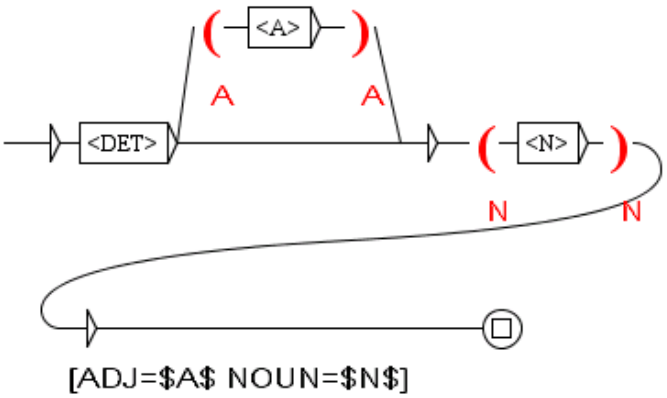


FIGURE 6.58 – Une variable *A* qui peut être indéfinie

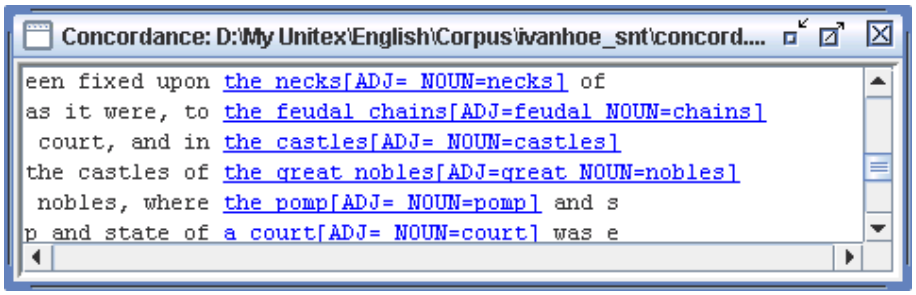


FIGURE 6.59 – La variable *A* peut être indéfinie

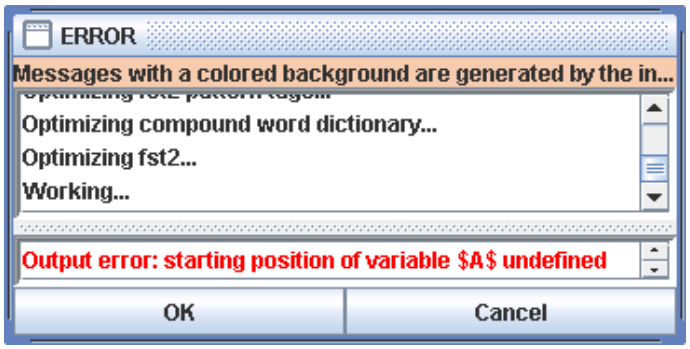


FIGURE 6.60 – Sortie à cause d'une variable erronée

les chemins lorsqu' elles sont indéfinies. Par exemple, l'application de la grammaire 6.58 produit seulement des sorties contenant des adjectifs , comme le montre la figure 6.61.

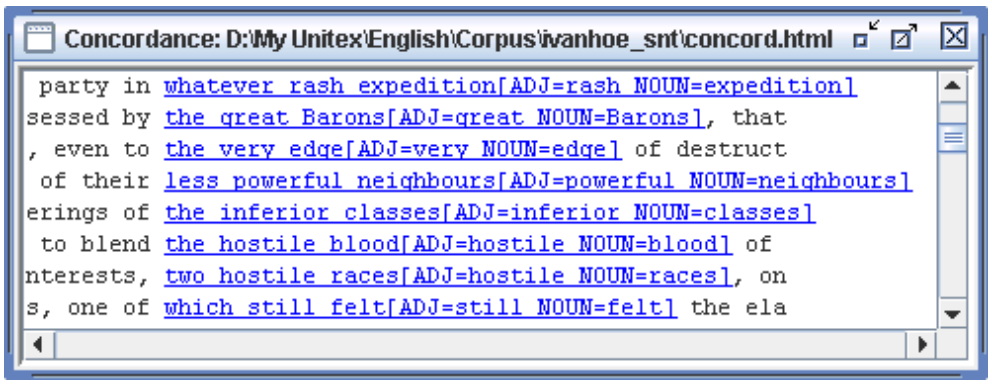


FIGURE 6.61 – Marche arrière en cas de variable erronée

6.10.3 Concordance

Le résultat de la recherche est un fichier d'index contenant les positions de toutes les occurrences trouvées. La fenêtre de la figure 6.62 vous propose de construire une concordance, de modifier le texte ou de comparer le résultat de la recherche à la recherche précédente sur le même 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.

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 icônifiée et que le texte n'est pas trop long pour être affiché, vous verrez apparaître la séquence sélectionnée (voir figure 6.63).

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

6.10.4 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 6.62. Ce fichier doit porter l'extension `.txt`.

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

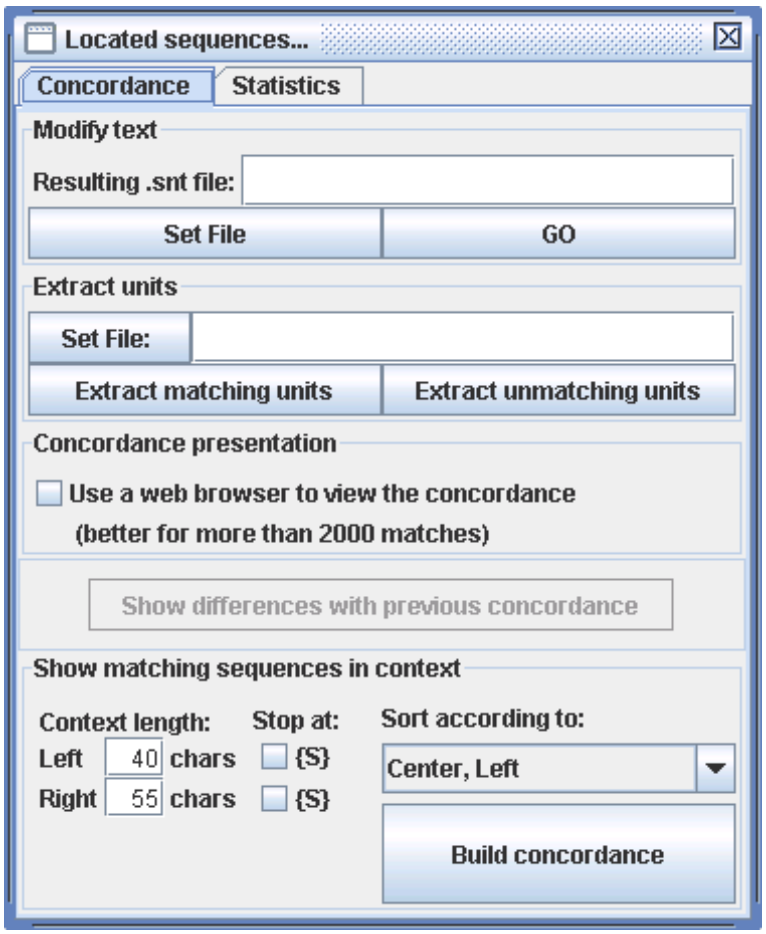


FIGURE 6.62 – Configuration de l'affichage des occurrences trouvées

bouton "GO" pour lancer la modification du texte. Les règles de priorités appliquées lors de cette opération sont détaillées à la section 6.7.

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.

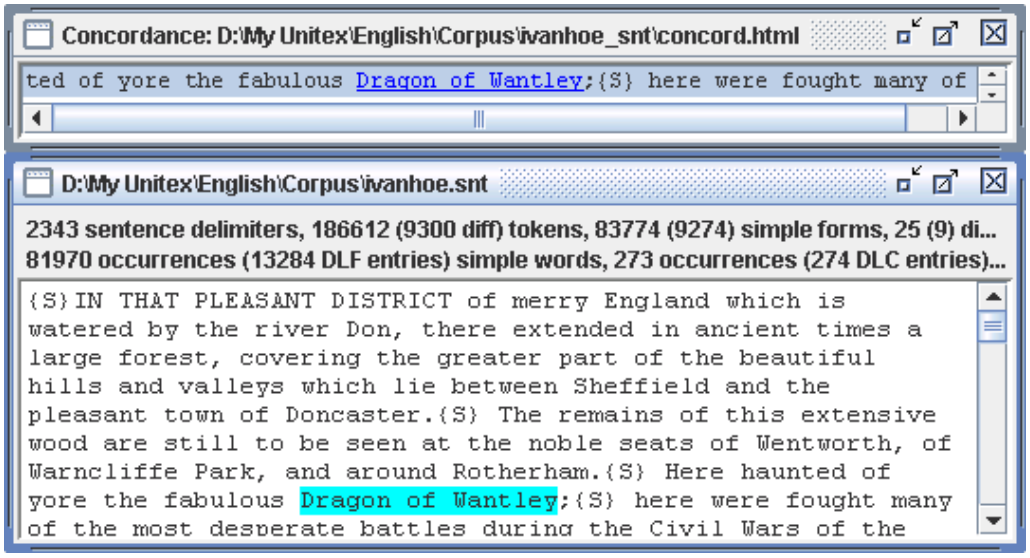


FIGURE 6.63 – Sélection d'une occurrence dans le texte

6.10.5 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 6.62). 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.

6.10.6 Comparaison de concordances

L'option "Show differences with previous concordance" permet de comparer la concordance qui vient d'être calculée avec la concordance précédente, si elle existe. Pour cela, le programme `ConcorDiff` construit les deux concordances dans l'ordre du texte, puis compare leurs lignes. Le résultat est une page HTML qui montre alternativement les lignes des deux concordances, laissant une ligne vide quand un match n'apparaît que dans une seule des deux concordances (figure 6.64).

Les lignes de la concordance antérieure sont grisées et celles de la concordance courante restent sur fond blanc. Dans chaque ligne, seules les séquences reconnues sont colorées. On peut cliquer dessus pour ouvrir le texte à cette position.

Le bleu indique qu'une séquence est commune aux deux concordances. Le rouge indique qu'une séquence reconnue est commune aux deux concordances, mais avec des extensions différentes, c'est-à-dire que les deux séquences reconnues se chevauchent partiellement. Le vert signale qu'une séquence n'apparait que dans une seule concordance.

S'il n'existe pas de concordance antérieure, le bouton est désactivé.

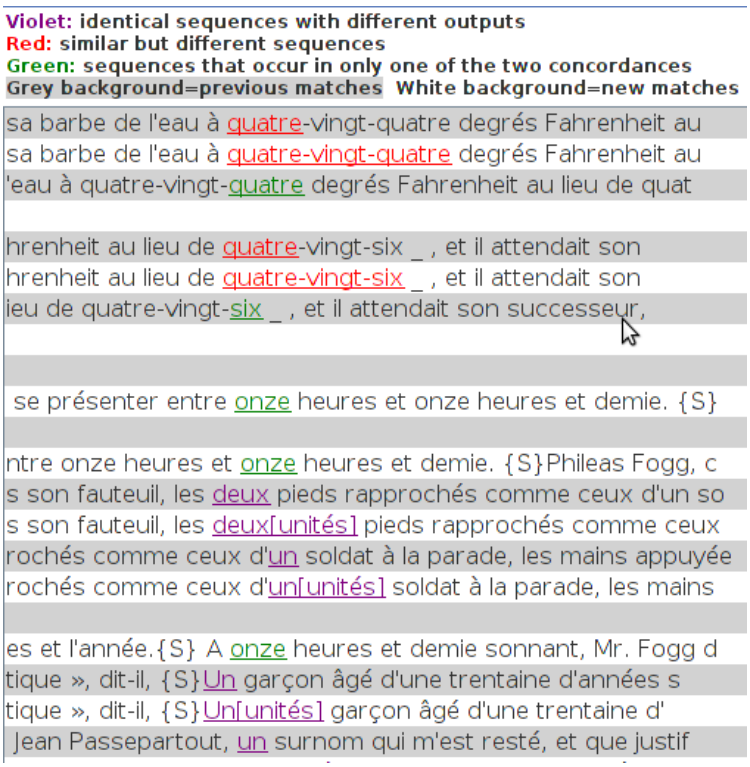


FIGURE 6.64 – Exemple de comparaison de concordances

6.10.7 Mode Debug

Lorsqu'on applique un graphe à un texte avec le menu `Locate` dans la fenêtre de la figure 6.53, si le mode debug est activé dans le champ "Locate pattern in the form of", la concordance est affichée dans une fenêtre spéciale (voir figure 6.65), divisée en trois parties :

En haut à droite, se trouve la fenêtre de concordance. Elle est identique à la fenêtre habituelle dans laquelle les séquences reconnues apparaissent en bleu.

En bas à droite se trouve le graphe utilisé par `Locate`.

A gauche, il y a un tableau divisé en trois colonnes : "Tag", "Output" et "Matched". Chaque token de la séquence reconnue apparaît dans la colonne "Matched", la colonne "Tag" indique le contenu de la boîte de l'automate qui l'a reconnue, et si elle possède une sortie, elle apparaît dans la colonne "Output".

Pour chaque séquence reconnue de la concordance, si on clique dessus, le tableau est mis à jour. Si on clique sur une ligne du tableau, le système colore la boîte correspondante dans le graphe. On peut ainsi voir pour chaque occurrence reconnue dans le texte quel chemin de l'automate la reconnait. Le nombre en rouge au-dessus d'une boîte indique le nombre de séquences du texte pour lesquelles cette boîte a reconnu un token.

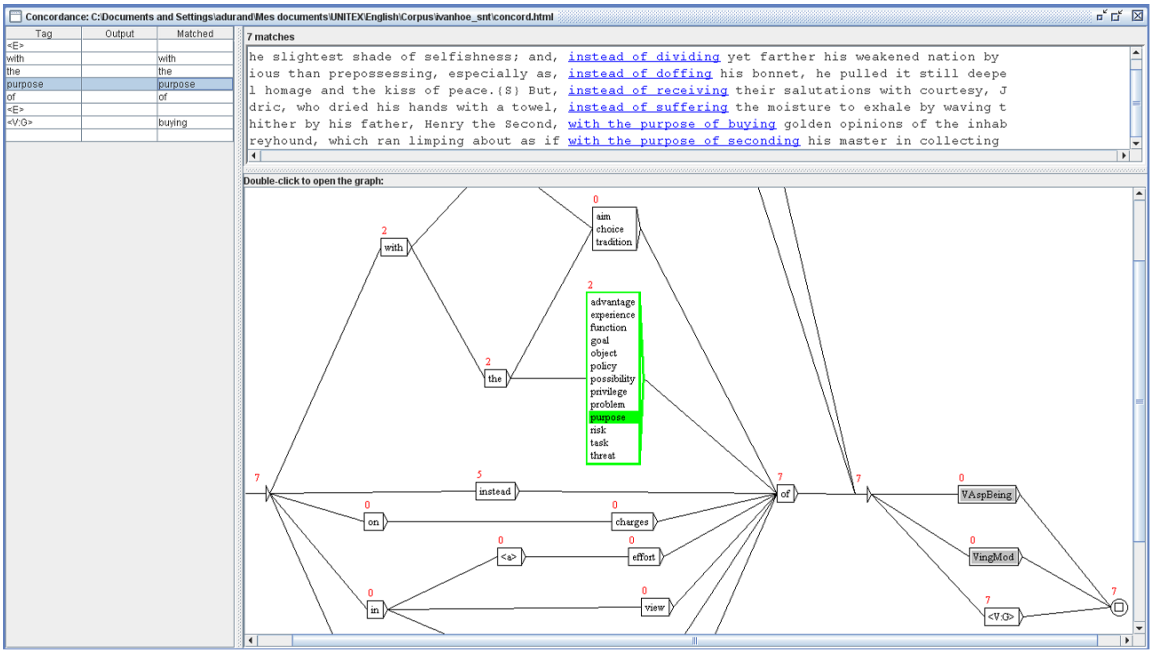


FIGURE 6.65 – La fenêtre de concordance en mode debug

Quand on applique un graphe en mode debug avec le menu `Text>Locate Pattern`, le système le compile en un fichier `fst2` dans un formal spécial de mode debug, qui n'est pas compatible avec CasSys. Voir la section 12.2.1 pour résoudre ce problème.

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. Depuis la version 2.1, il est possible d'effectuer des recherches de motifs sur l'automate du texte (voir section 7.7).

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 suivants et précédents.

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 illustré 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.

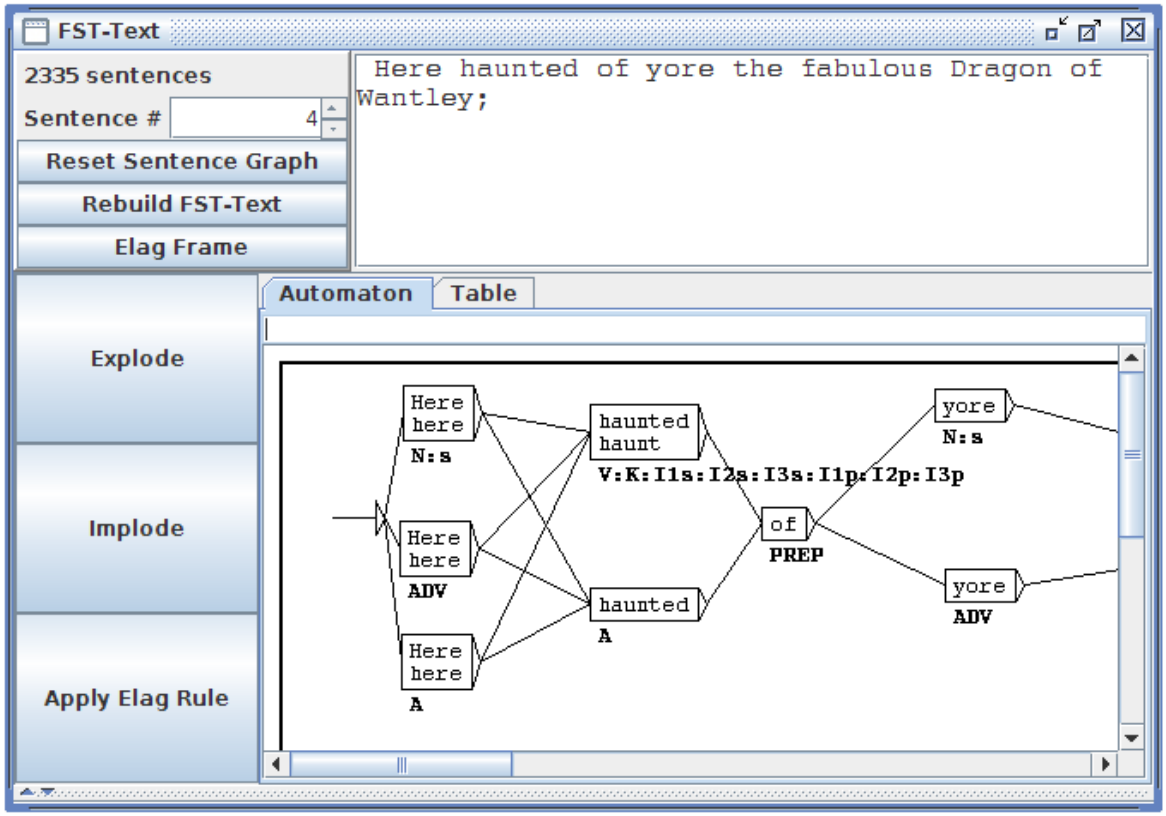


FIGURE 7.1 – Exemple d'automate de phrase

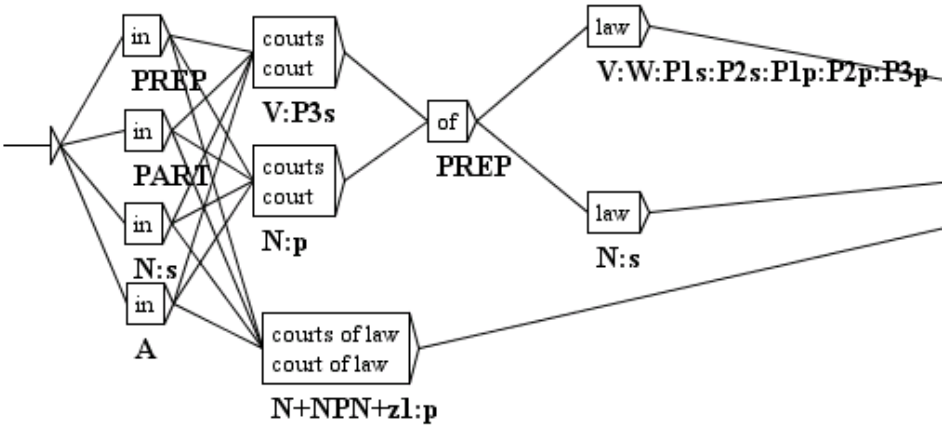


FIGURE 7.2 – Concurrence entre un mot composé et une combinaison de mots simples

7.2 Construction

Pour construire l'automate d'un texte, vous devez ouvrir ce texte, puis cliquer dans le menu "Text" sur "Construct FST-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.

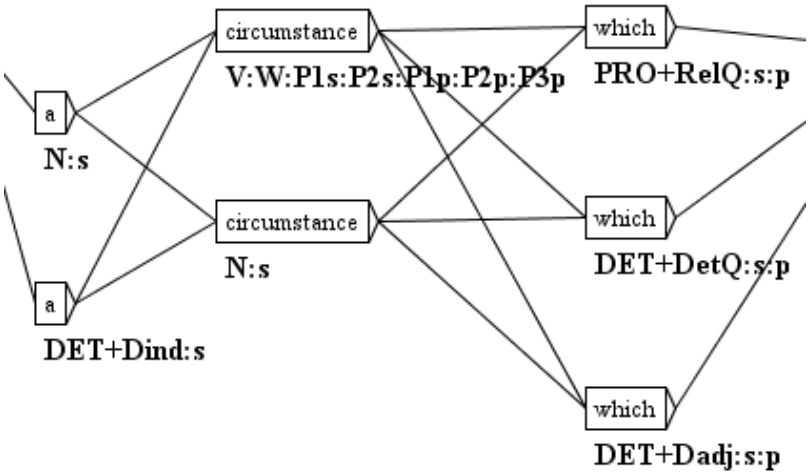


FIGURE 7.3 – Double entrée pour `which` en tant que déterminant

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 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 (*e.g.* `{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 1re ligne contient la forme fléchie trouvée dans le texte, et la 2e ligne contient la forme canonique si elle est différente. Les autres informations sont codées sous la boîte (voir section 7.5.1).

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

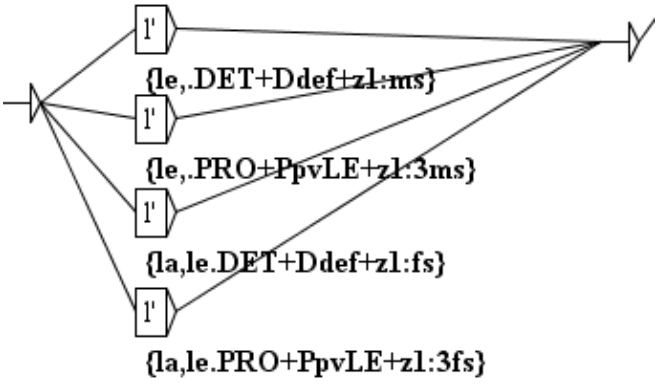


FIGURE 7.4 – Normalisation de la séquence `l'`

Si l'on applique cette grammaire à une phrase française contenant la séquence `l'`, 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 `l'` ont été appliquées, ce qui a ajouté quatre étiquettes dans l'automate. Ces étiquettes ne sont

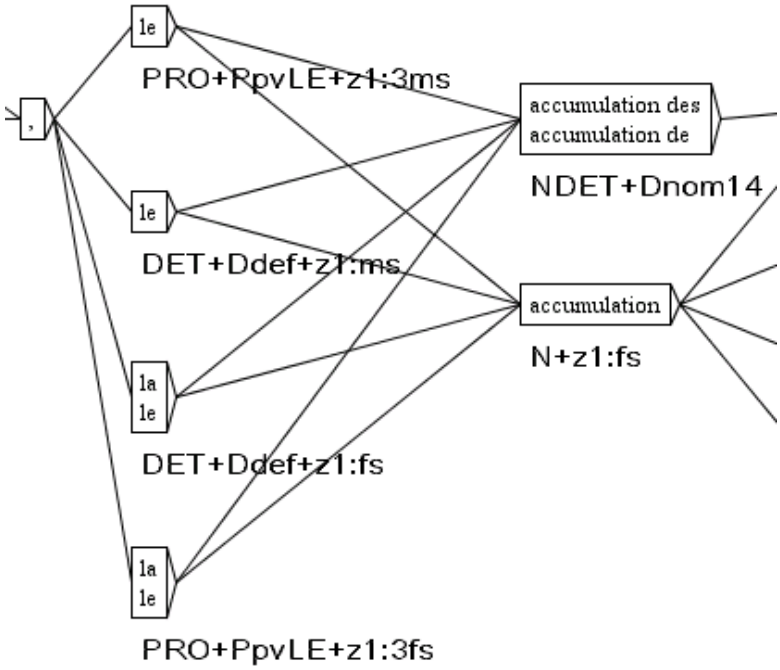


FIGURE 7.5 – Automate normalisé avec la grammaire de la figure 7.4

pas concurrentes avec les deux chemins préexistants pour la séquence `l'` grâce à l'heuristique "keep best paths" (voir section 7.2.4). La normalisation à la construction de l'automate du texte permet d'ajouter des chemins à l'automate, pas d'en supprimer. La suppression des chemins est partiellement faite par l'heuristique "keep best paths" si elle est sélectionnée. Pour aller plus loin, vous devez utiliser les fonctionnalités de désambïguisation du système ELAG

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 sous-jacente, il est nécessaire de l'introduire dans l'automate du texte, en parallèle avec 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.

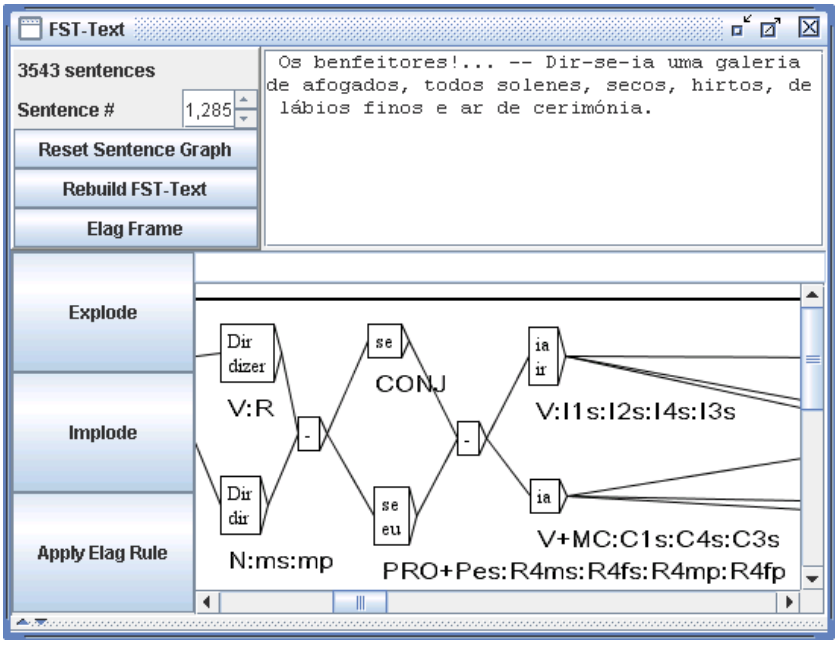


FIGURE 7.6 – Automate de phrase non normalisé

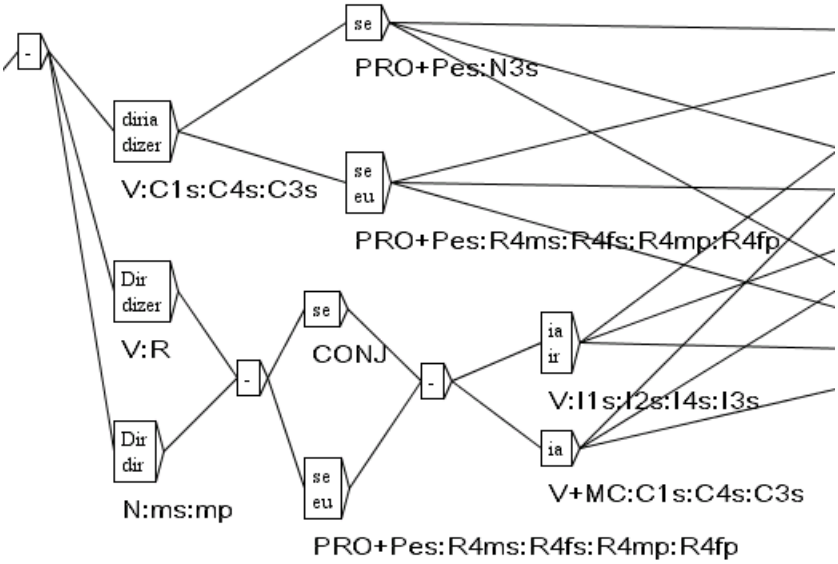


FIGURE 7.7 – Automate de phrase normalisé

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, 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`.

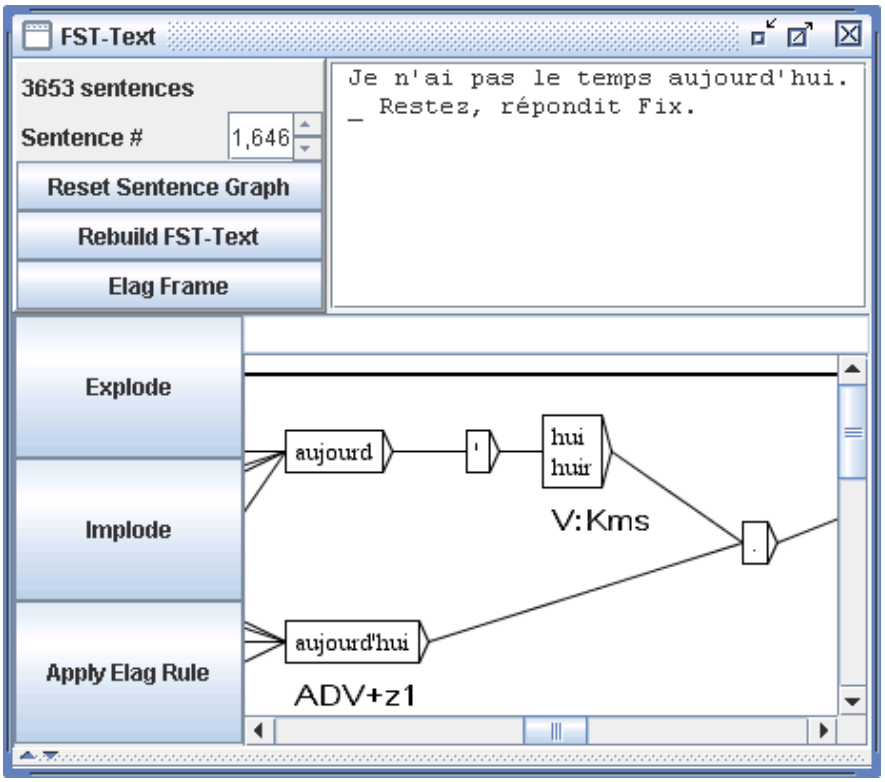


FIGURE 7.8 – Ambiguïté due à une séquence contenant un mot inconnu

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.

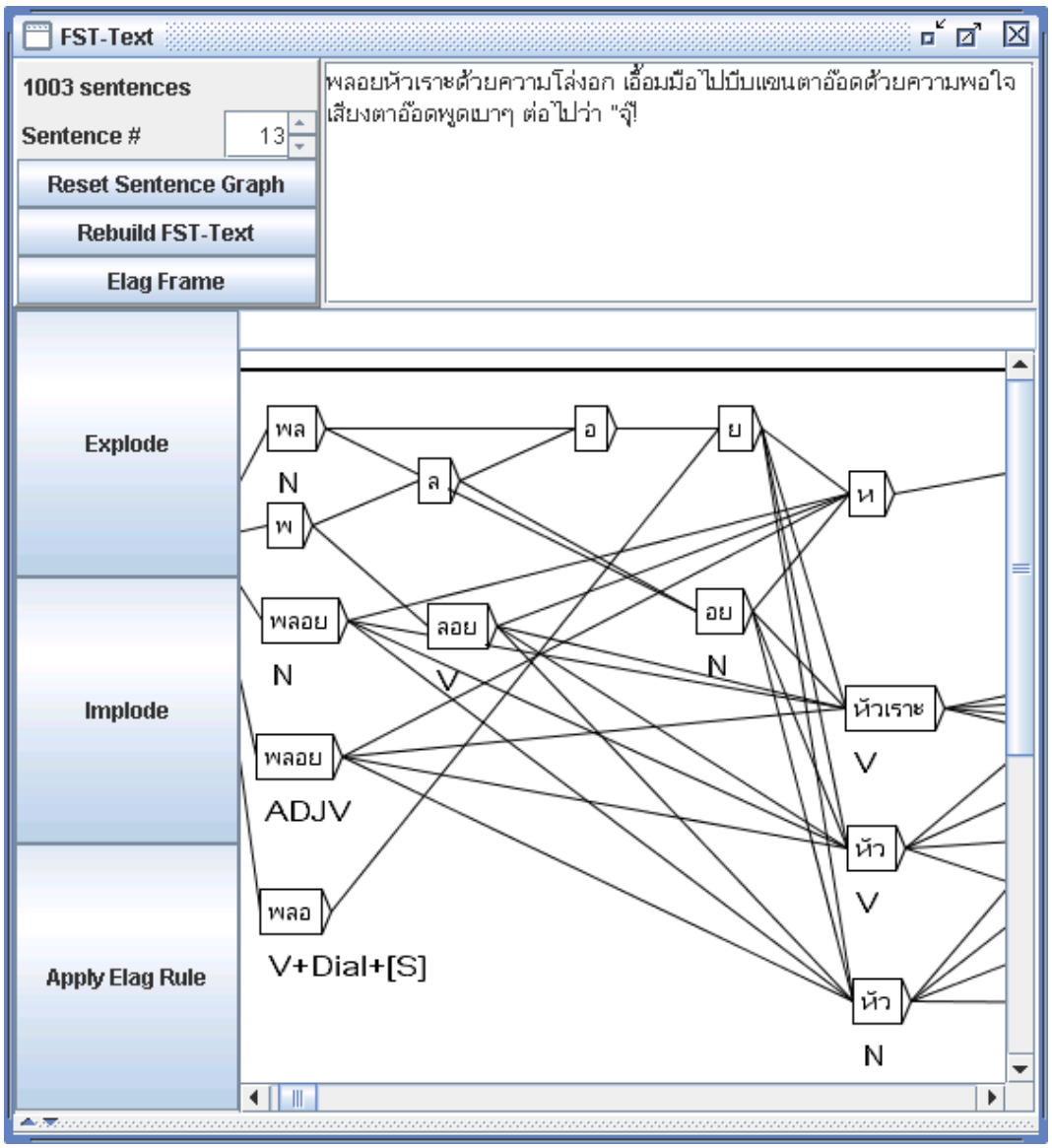


FIGURE 7.9 – Automate d'une phrase thaï

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.

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` suivi d'une apostrophe et de `hui`, car `aujourd` est un mot inconnu, ce

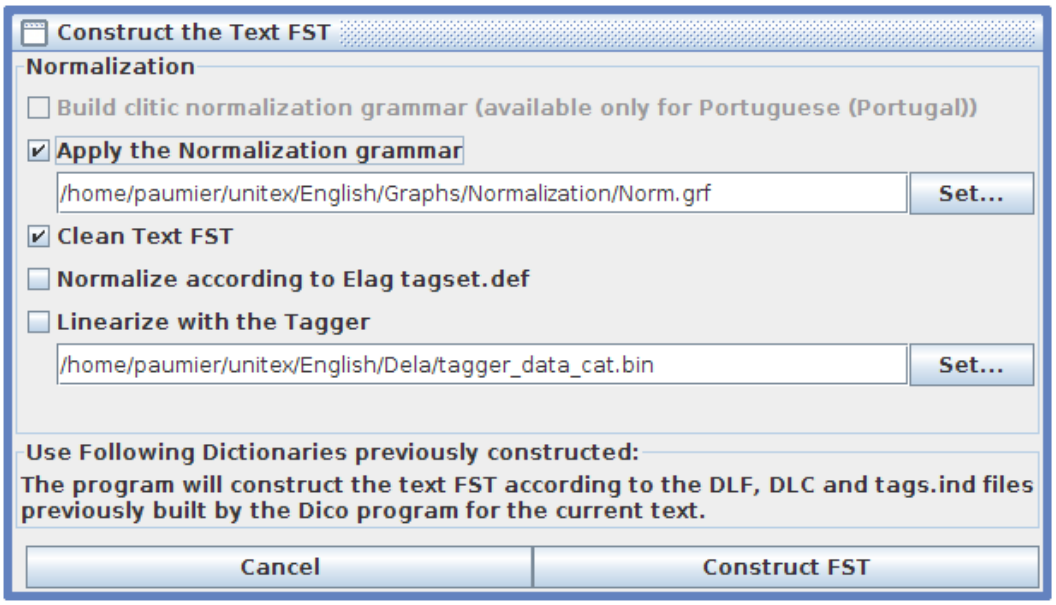


FIGURE 7.10 – Configuration de la construction de l'automate du texte

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 [6] et [64].

7.3.1 Grammaires de levée d'ambiguïtés

Les grammaires manipulées par ELAG ont une syntaxe particulière. Elles comportent deux parties, que nous appelerons partie *si* et *alors*. La partie *si* d'une grammaire ELAG se divise en deux zones délimitées par des boîtes contenant le symbole <!>. La partie *alors* est divisée de la même façon au moyen du symbole <=>. 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* reconnait 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

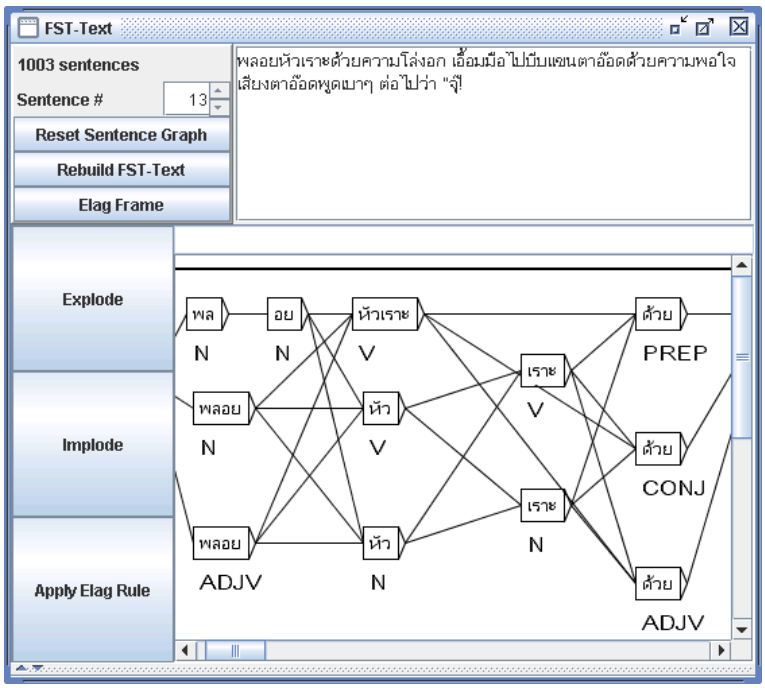


FIGURE 7.11 – Automate de la figure 7.9 après nettoyage

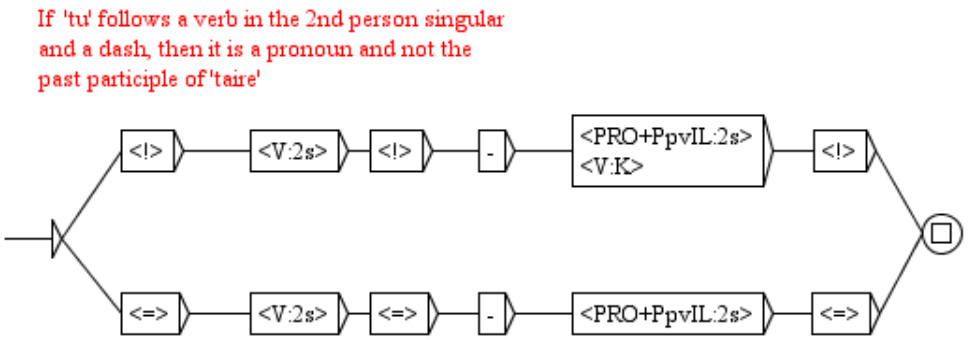


FIGURE 7.12 – Exemple de grammaire ELAG `elag-tu.grf`

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 `<!>` dans la partie *si*, et par le deuxième symbole `<=>` dans la partie *alors*. Ces

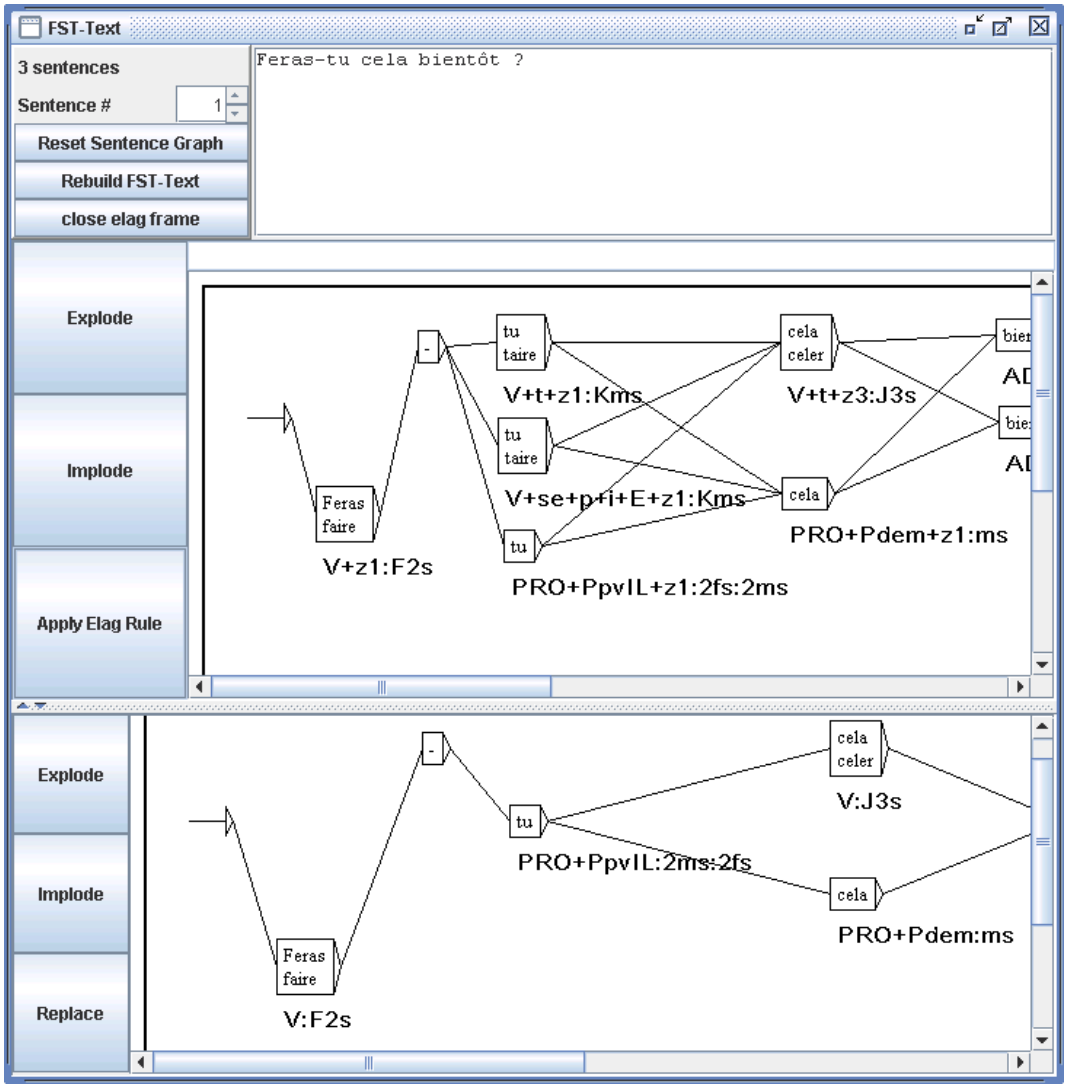


FIGURE 7.13 – Résultat de l'application de la grammaire de la figure 7.12

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.

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

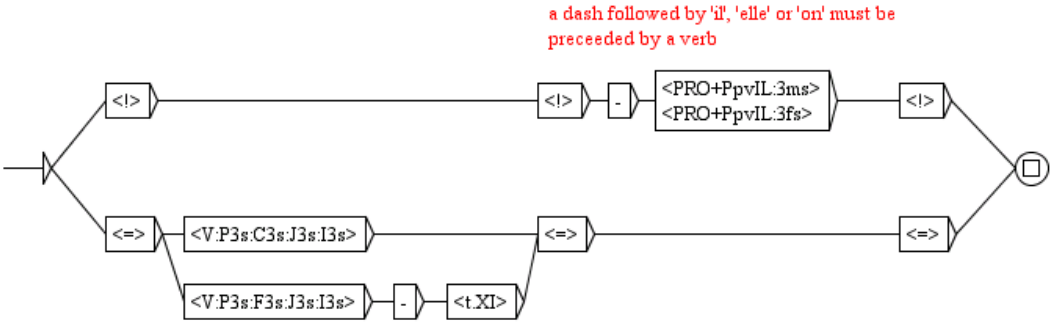


FIGURE 7.14 – Utilisation du point de sychronisation

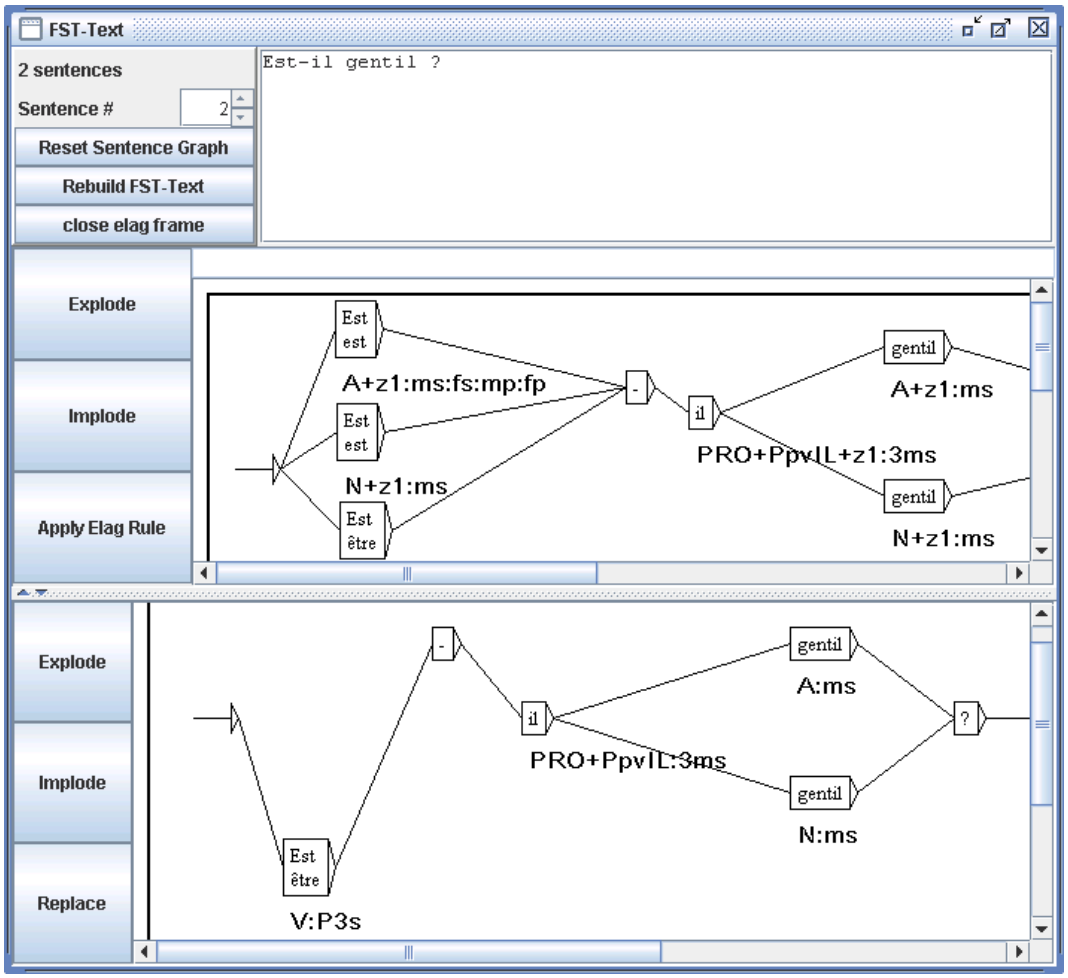


FIGURE 7.15 – Résultat de l'application de la grammaire de la figure 7.14

le menu "Text", qui fait apparaître la fenêtre de la figure 7.16.

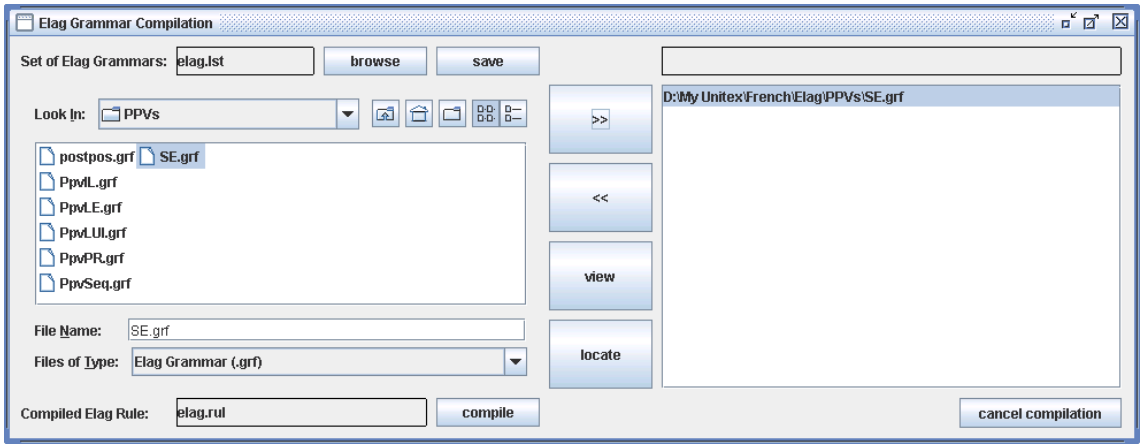


FIGURE 7.16 – Fenêtre de compilation des grammaires ELAG

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` par défaut.

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 *si* 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 "Apply Elag Rule". Une boîte de dialogue apparaîtra pour vous demander le nom du fichier `.rul` à utiliser (voir figure 7.17). Comme le fichier par défaut est bien `elag.rul`, cliquez simplement sur "OK". Cela lancera le programme `Elag` qui va effectuer la levée d'ambiguïtés.

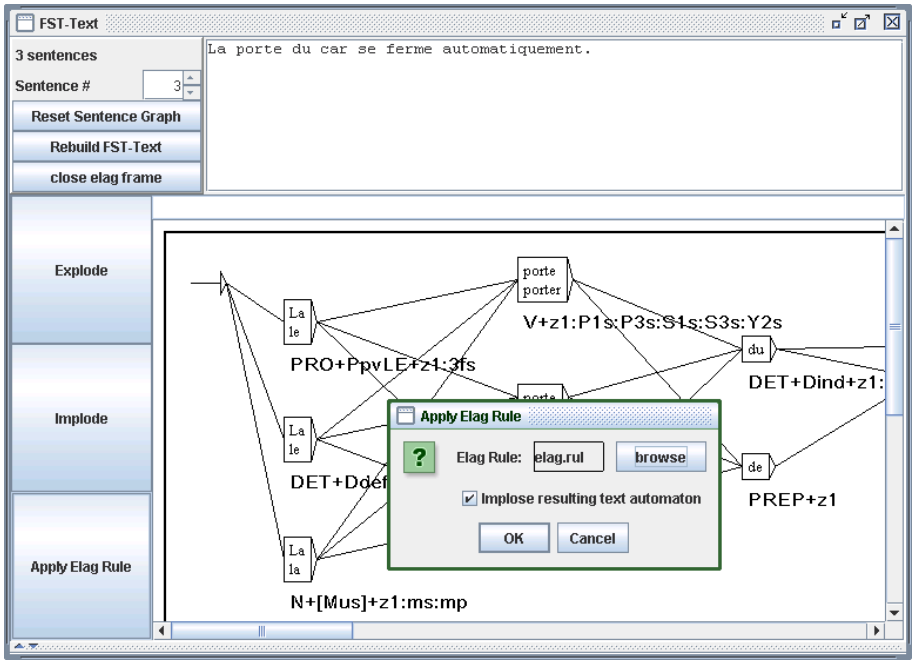


FIGURE 7.17 – Fenêtre de l'automate du texte

Une fois le programme terminé, vous pouvez consulter l'automate résultat en cliquant sur le bouton "Open Elag Frame" button. Comme on le voit sur la figure 7.18, la fenêtre est séparée en deux : 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 factorisées [1] 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.

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.

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

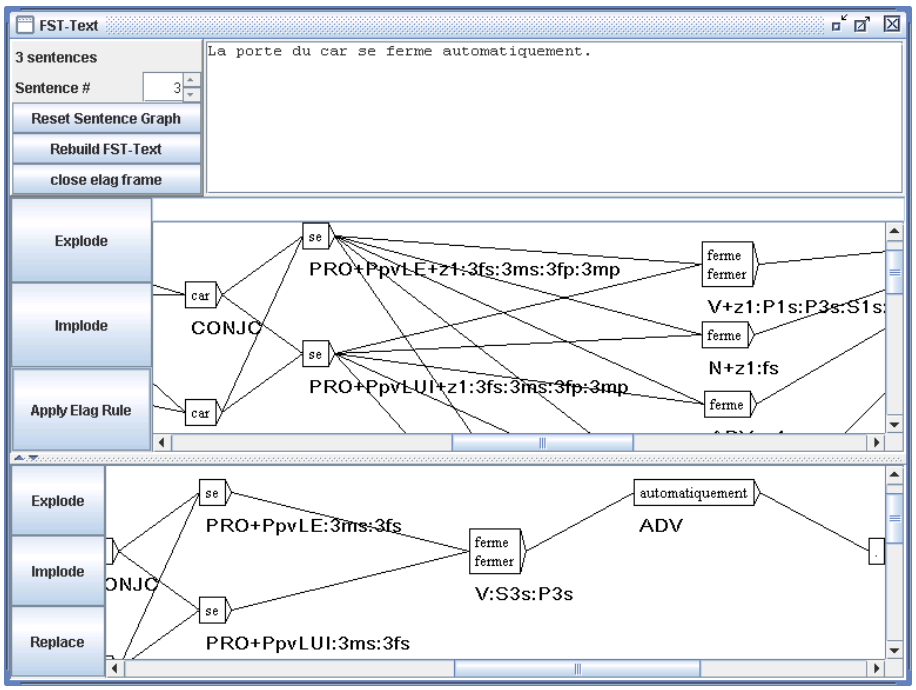


FIGURE 7.18 – Fenêtre de l'automate du texte séparée en deux

Pour modifier le nom de l'ensemble, cliquez sur le bouton "Browse". Dans la boîte de dialogue 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 `.rul`).

Vous pouvez maintenant appliquer votre ensemble de grammaires. Comme expliqué plus haut, cliquez sur le bouton "Apply Elag Rule" 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 séletionnez votre ensemble. L'automate résultat est identique à celui qui aurait été obtenu en appliquant successivement chacune des grammaires.

7.3.5 Fenêtre de traitement d'ELAG

Lors de la désambiguïsation, le programme `Elag` est lancé dans une fenêtre de traitement 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.

Évaluation du taux d'ambiguïté

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 :

$$\textit{taux d'ambiguïtés} = exp^{\frac{log(nombredechemins)}{longueurdutexte}}$$

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 traitement d'ELAG.

7.3.6 Description du jeu d'étiquettes

Les programmes `Elag` and `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` qui se trouve dans votre répertoire personnel, dans le sous-répertoire de la langue choisie

`tagset.def` file

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

```
NAME french

POS ADV
.
```

```
POS PRO
flex:
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                                 # en y
PronQ                                 # ou qui que quoi
Dnom                                  # rien
.

POS A ## adjectifs
flex:
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
flex:
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   # eusse dusse puisse fusse (-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 (`french`, dans l'exemple). La suite du fichier est constituée de sections `POS` (pour Part-Of-Speech, partie du discours), 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 :

- `flex` : 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 pour les pronoms mais pas pour les 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 or 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
  ```

mais cet attribut n'est pas forcément présent pour tous les mots.

- `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 (dénoté 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 décrites 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`.

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 comme invalide ou non, lors du chargement de l'automate du texte.

Ce sont des codes facultatifs, qui sont indépendants des autres codes, comme par exemple l'attribut de niveau de langue (`z1`, `z2` or `z3`). De la même manière que pour les codes flexionnels, il est également possible de nier un attribut flexionnel en écrivant le caractère `!` juste

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.*

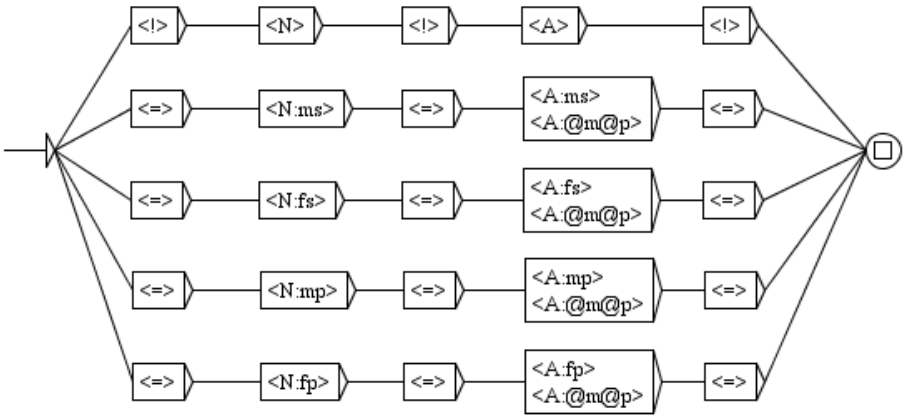


FIGURE 7.19 – Grammaire ELAG vérifiant l'accord en genre et en nombre entre un nom et l'adjectif qui le suit

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 `gauche`,[3].

Tous les codes qui ne sont pas déclarés dans le fichier `tagset.def` sont ignorés par ELAG. Si une entrée de dictionnaire contient un tel code, ELAG produira un avetissement 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 programme 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 intégrées dans la description du jeu d'étiquettes. Celle-ci sont donc automatiquement éliminées lors du chargement de l'automate du texte, ce qui réduit son taux d'ambiguïtés.

Afin de bien distinguer les effets liés au jeu d'étiquettes de ceux 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`.

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

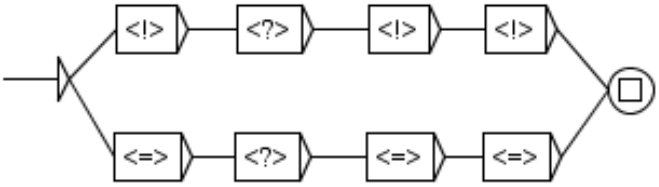


FIGURE 7.20 – Grammaire ELAG n'exprimant aucune contrainte

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 interprétations lexicales 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 *then* 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.

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>`

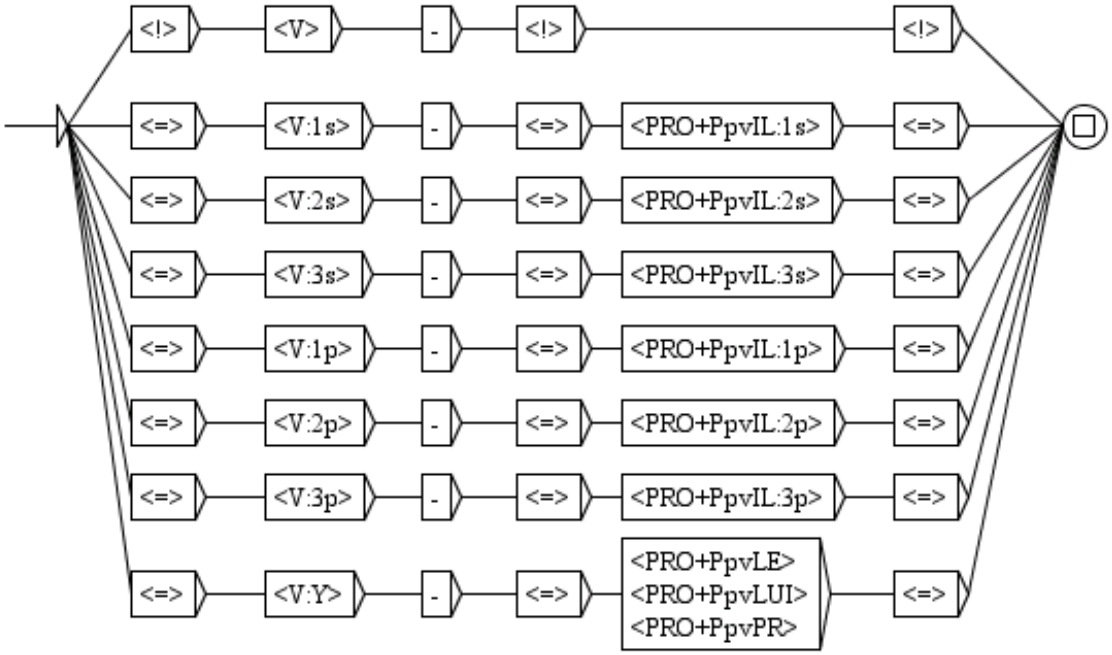


FIGURE 7.21 – Grammaire ELAG vérifiant l'accord entre verbe et pronom

et `<je.PRO>` par le symbole `<PRO+PpvI1:1s>`. 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 pas tous ces traits, le programme considérera en vain des étiquettes non existantes telles `<je.PRO:3p>`, `<je.PRO+PronQ>` etc. en vain.

7.4 Linéarisation de l'automate du texte avec le taggeur

Par défaut, l'automate du texte contient de nombreux chemins étiquetés en raison de l'ambiguïté lexicale. Le processus de linéarisation consiste à choisir un chemin unique, une séquence d'étiquettes avec une étiquette par token et de supprimer les autres. Le résultat est un automate du texte avec un seul chemin (voir section 7.6 pour convertir un automate linéaire en un texte linéaire). Le choix du chemin dépend de son score. Le chemin avec le meilleur score est choisi et les autres supprimés Le score d'un chemin est calculé par un modèle statistique entrainé sur un corpus annoté. Ce modèle utilise des fichiers de données du taggeur produites par le programme TrainingTagger (vour section 14.44). Par exemple, vous pouvez voir figure 7.23, l'automate du texte original sur la phrase *Les insectes nuisibles envahissent la maison*. L'automate du texte après linéarisation est celui de la figure 7.24.

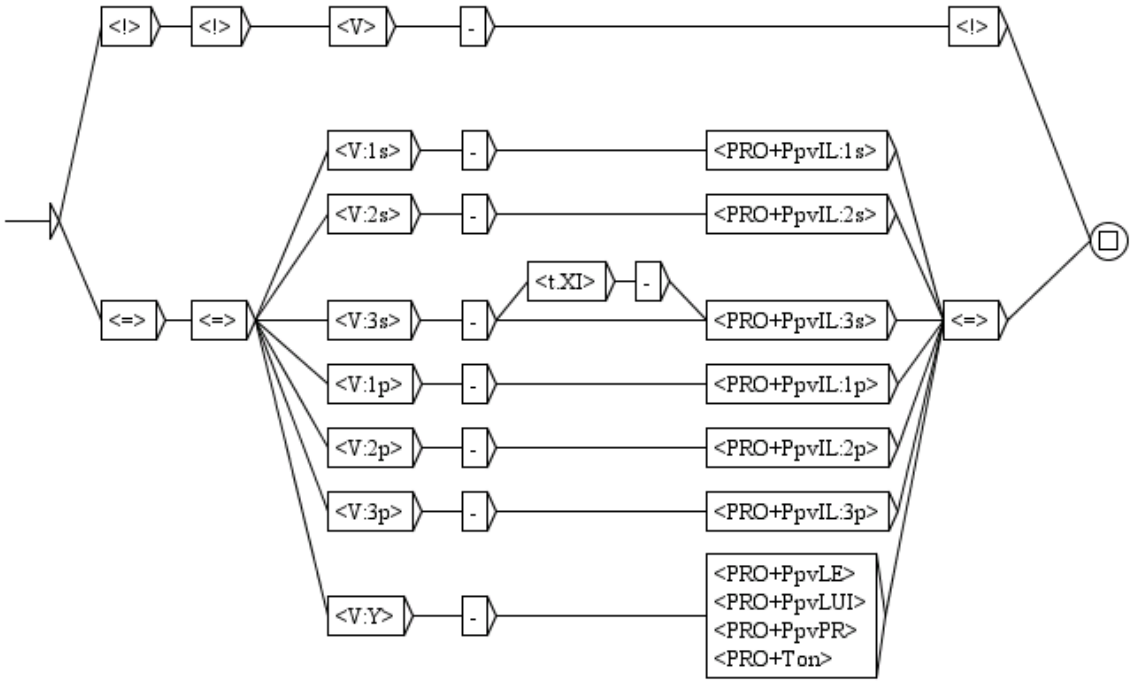


FIGURE 7.22 – Grammaire ELAG optimisée vérifiant l'accord entre verbe et pronom

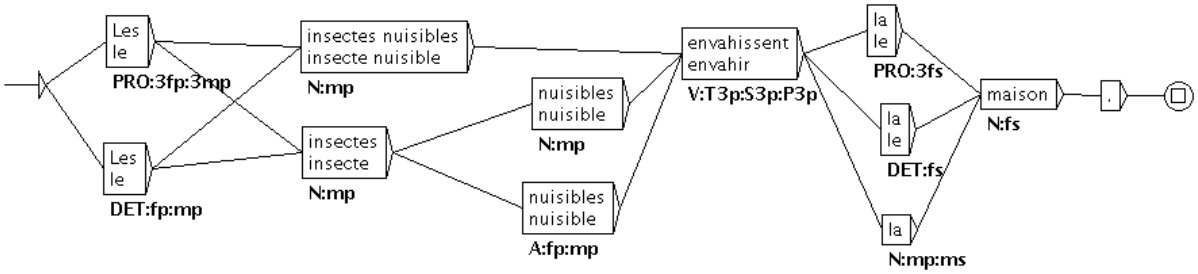


FIGURE 7.23 – Automate du texte de *Les insectes nuisibles envahissent la maison.*

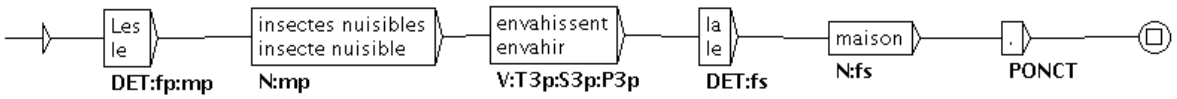


FIGURE 7.24 – Automate du texte linéarisé

7.4.1 Compatibilité du jeu d'étiquettes

Le jeu d'étiquettes du taggeur est identique à celui du corpus d'entrainement ou en est une variante (voir ci-dessous). Toutefois, pour utiliser le taggeur sur l'automate du texte, on doit faire attention au jeu d'étiquettes et à la morphologie. Le jeu d'étiquettes du modèle doit être identique à celui de l'automate du texte. Par exemple, si le modèle statistique a été calculé avec les étiquettes `DET` pour les mots `the`, l'étiquette correspondante dans le texte doit être `DET`. Unitex dispose d'une fonctionnalité qui permet de changer la forme des mots du texte, par exemple pour normaliser `doesn't` en `does not`. Appliquer des graphes de remplacement ou de normalisation peut entrainer des modifications de la forme des mots. Si un tel traitement a été appliqué au texte, il doit avoir été appliqué également au corpus d'entrainement. Si ces règles ne sont pas respectées, le taggeur pourrait être incapable de trouver le chemin souhaité dans l'automate du texte.

Le programme TrainingTagger produit deux variantes de taggeur. Le premier supprime des transitions sur la base de codes gramaticaux, sémantiques, syntaxiques et flexionnels (par exemple, `the.DET+Ddef:s` au lieu de `the.DET+Ddef:p`). Le second supprime les transitions sur la base de codes gramaticaux, sémantiques et syntaxiques (`that.DET+Ddem` au lieu de `that.PRO+Pdem`). Ce traitement accélère l'entrainement et les informations flexionnelles ne sont plus nécessaires pour toutes les applications.

7.4.2 Utilisation du Tagger

Pour linéariser l'automate du texte, vous devez choisir l'option "Linearize with the Tagger" dans la fenêtre de configuration pour construire l'automate du texte (cf. figure 7.25). Avec cette option, le programme linéarise chaque phrase de l'automate. Vous devez également sélectionner le fichier de données du taggeur (avec l'extension ".bin") en cliquant sur le bouton "Set". Le fichier de données du taggeur suffixé par "morph" est la première variante (avec les codes flexionnels) et celle suffixée par "cat" est la seconde (sans codes flexionnels). Si vous utilisez les données de type "morph", vous devez également cliquer sur "Normalize accordind to Elag tagset.def" (pour plus de détails, voir section 14.38 au sujet du programme `Tagger`).

Par exemple, l'automate du texte, de la figure 7.24, est la sortie de la linéarisation de l'automate du texte de la figure 7.23 avec la version "cat". La linéarisation de l'automate avec la version "morph" se trouve figure 7.26.

7.4.3 Création d'un nouveau taggeur

Pour créer un nouveau taggeur pour votre langue, vous devez lancer le programme TrainingTagger sur votre propre corpus annoté. Le format du corpus annoté est décrit dans 15.10.1. Comme nous le signalions à la section 7.4.1, vous devez faire attention au jeu d'étiquettes et à la morphologie. Avant de calculer un modèle statistique, vous devez décider quels dictionnaires et graphes de normalisation vous utiliserez pour construire l'automate

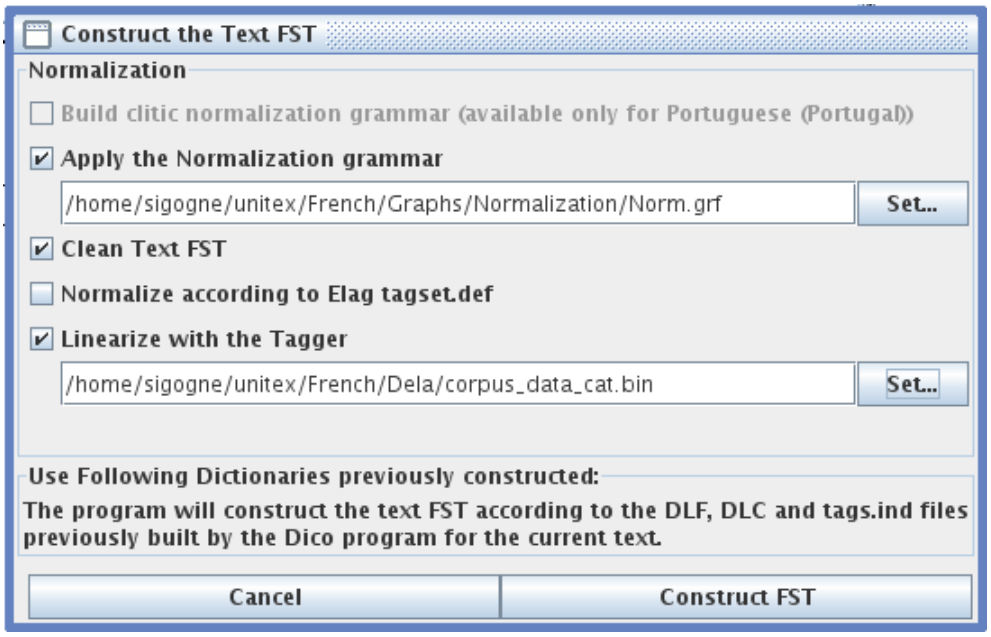


FIGURE 7.25 – Configuration de la linéarisation de l'automate du texte

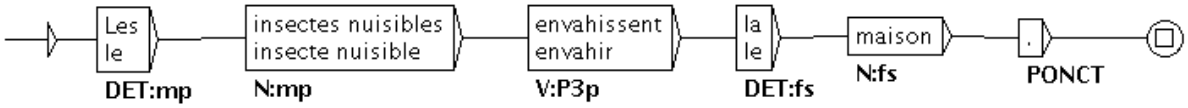


FIGURE 7.26 – L'automate du texte linéarisé avec les données de type "morph"

du texte. Puis, vous devrez modifier le corpus annoté si la forme des mots ou le jeu d'étiquettes ne sont pas identiques. Par exemple, si le graphe de normalisation transforme le mot `jusqu'` en `jusque`, le mot correspondant dans le corpus annoté doit être `jusque`.

Un taggeur pour le français est fourni avec Unitex. Il a été créé avec un corpus annoté composé d'étiquettes dépourvues de codes sémantiques et syntaxiques.

7.5 Manipulation de l'automate du texte

7.5.1 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`.

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 fléchie 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 7.27 montre le graphe obtenu pour la première phrase *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 fléchie par une minuscule.

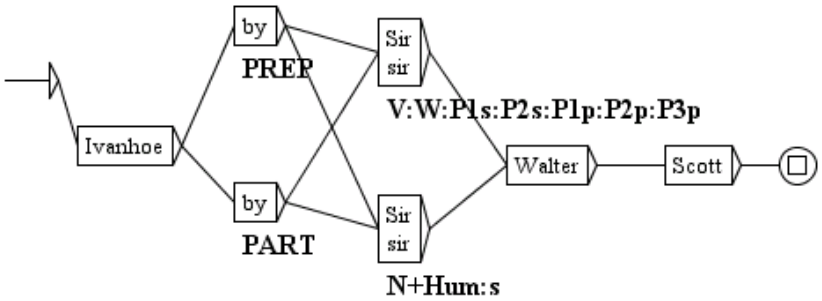


FIGURE 7.27 – Automate de la première phrase *Ivanhoe*

7.5.2 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). On peut modifier ou supprimer des boîtes ou des transitions (voir figure 7.28).

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, mais cette opération ne modifie pas l'automate global du texte. On peut donc annuler les modifications faites manuellement et réinitialiser l'automate de cette phrase en cliquant sur le bouton "Reset Sentence Graph".

Lorsqu'on sélectionne une phrase, si un graphe modifié existe pour cette phrase, Unitex l'affiche.

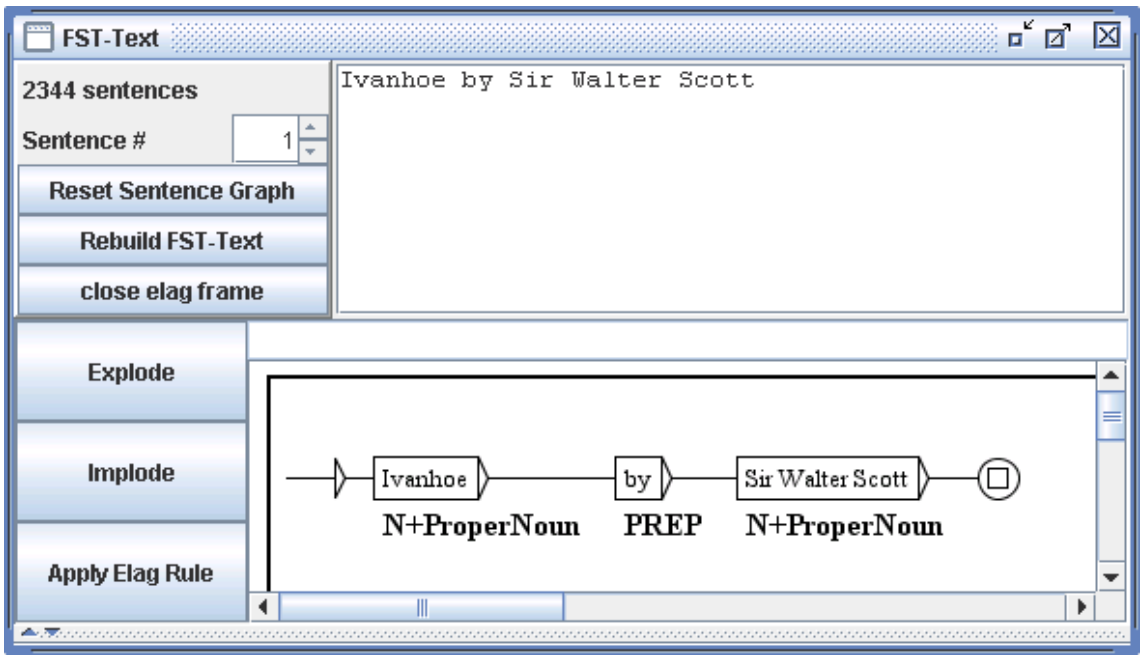


FIGURE 7.28 – Automate de phrase modifié

Après avoir édité des automates de phrases, on peut sauvegarder ses modifications manuelles dans l'automate global du texte. Pour cela, on clique 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.

Lors de la construction de l'automate d'un texte (7.2), toutes les modifications manuelles sauvegardées dans l'automate du texte sont annulées.

Levée manuelle des ambiguïtés

L'automate du texte peut contenir de nombreux chemins étiquetés en raison de l'ambiguïté lexicale. Vous pouvez soit lever les ambiguités avec des grammaires ELAG, soit sélectionner manuellement les chemins corrects pour l'un ou tous les graphes de l'automate de phrase. Vous devez pour cela effectuer un clic droit sur la boîte que vous voulez garder lorsque plusieurs boîtes avec différentes étiquettes sont proposées. Les bords de la boîte sélectionnée deviendront plus gras tandis que les autres boîtes apparaîtront grisées (voir figure 7.29).

Vous pouvez alors cliquer sur le bouton "Remove greyed states" pour ne garder que les boîtes sélectionnées (figure 7.30).

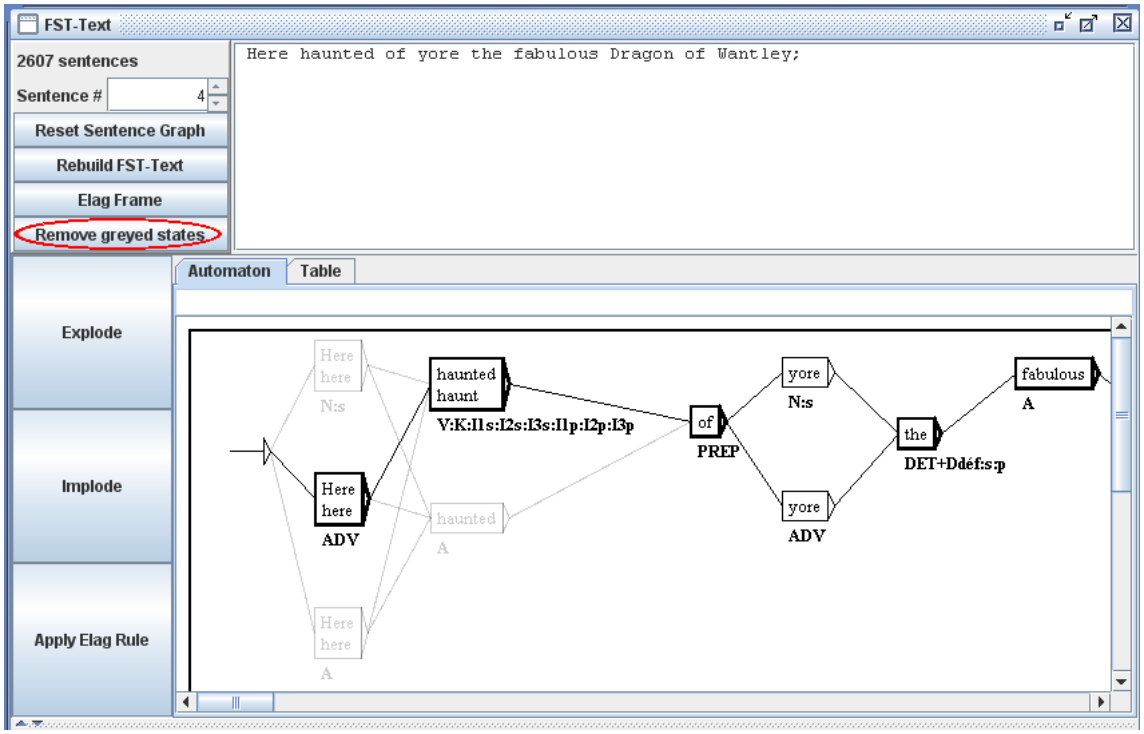


FIGURE 7.29 – Levée manuelle d'ambiguités dans l'automate de phrases

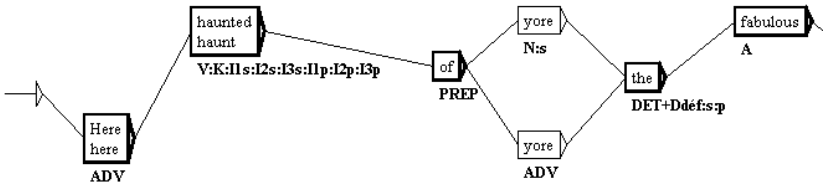


FIGURE 7.30 – Suppression de boîtes ambiguës dans l'automate de phrase

7.5.3 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".

7.6 Convertir l'automate du texte en texte linéaire

Si l'automate du texte ne contient plus la moindre ambiguïté, il est possible de construire un fichier texte correspondant à l'unique chemin représenté par cet automate. Pour cela, allez dans le menu "Text" et cliquez sur "Convert FST-Text to Text...". La fenêtre de la figure 7.31 vous permet alors de définir le fichier texte de sortie.

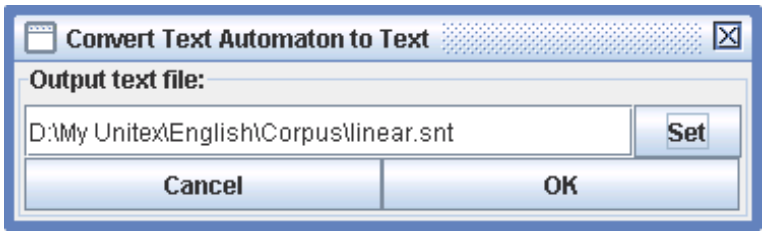


FIGURE 7.31 – Choix du fichier de sortie pour la linéarisation de l'automate du texte

Si l'automate n'est pas complètement linéaire, un message d'erreur vous indiquera le numéro de la première phrase contenant une ambiguïté. Sinon, le programme `Tfst2Unambig` construira le fichier de sortie selon les principes suivants :

- le fichier de sortie contient une ligne par phrase ;
- toutes les phrases sauf la dernière sont terminées par `{S}` ;
- pour chaque boîte, le programme écrit son contenu suivi par un espace.

NOTE : la gestion des espaces est entièrement laissée à l'utilisateur. Ainsi, si le texte d'origine est celui de l'automate de phrase de la figure 7.32, le texte produit sera :

```
2 3 {cats,cat.N+Anl:p} {are,be.V:P2s:P1p:P2p:P3p} {white,white.A} .
```

7.7 Recherche de motifs dans l'automate du texte

Le programme `LocateTfst` d'Unitex peut effectuer des recherches sur l'automate du texte. Les principaux avantages sont que vous pouvez :

- bénéficier de la suppression de l'ambiguité ;
- bénéficier de l'application de grammaire de normalisation (voir ci-dessous) ;
- travailler à plusieurs niveaux morphologiques (mots composés, mots simples morphèmes). C'est particulièrement intéressant car vous pouvez facilement manipuler les langues agglutinantes comme le coréen (pour le coréen, voir section 7.9).

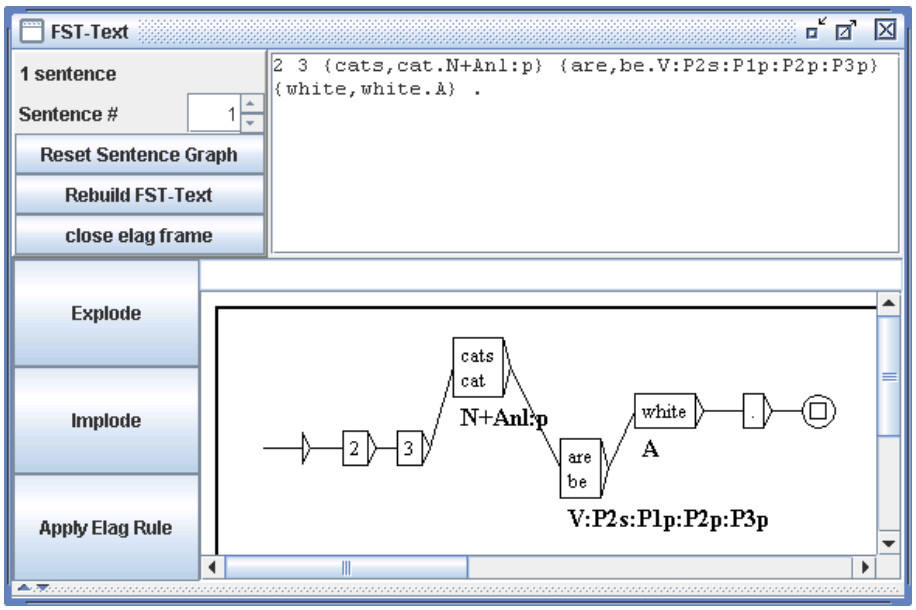


FIGURE 7.32 – Exemple d'automate de texte linéaire

Les règles sont très proches de celles qui s'appliquent lors des recherches avec `Locate`. Voici les différences :

- vous ne pouvez pas mémoriser des séquences dans des variables, comme avec `Locate` (voir figure 6.19, page 139)
- vous ne pouvez pas reconnaître des choses qui ne sont pas l'automate du texte : si l'automate du texte contient seulement l'étiquette d'un mot composé, mais pas des mots simples qu'il renferme, vous ne pourrez pas reconnaître les mots simples. Par exemple, dans la phrase de l'automate de la figure 7.33, il est impossible reconnaître `soixante` ou `huit`, puisque ces chemins n'existent pas.
- les séquences reconnues peuvent être différentes de celles apparaissant dans les concordances. En fait, l'automate du texte peut contenir des étiquettes qui ne correspondent pas au texte brut d'entrée, en particulier lorsqu'une grammaire de normalisation a été appliquée. Par exemple, si vous recherchez le motif `<le.DET>` dans l'automate du texte de *80jours*, vous obtenez 7703 matches, tandis que `Locate` n'en trouve que 5763. Ceci provient du fait que quelques mots ont été normalisés, comme `au` → `à le` ou `du` →`de le`. Ainsi, quand vous cherchez `<le.DET>`, `LocateTfst` reconnaît les étiquettes ajoutées à l'automate du texte par la grammaire de normalisation, alors que `Concord` utilise le fichier texte original pour produire la concordance, comme le montre la figure 7.34.
- `<TOKEN>` ne reconnaît pas les tokens tel que définis dans `tokens.txt`. Il reconnaît n'importe quelle étiquette de l'automate du texte. Les étiquettes reconnues peuvent

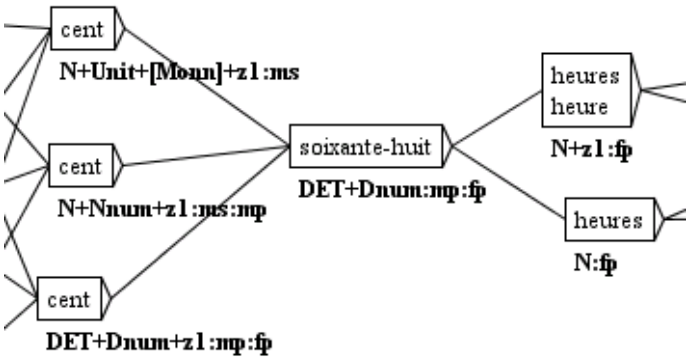


FIGURE 7.33 – Phrase de l'automate qui ne reconnaît pas le motif *huit*

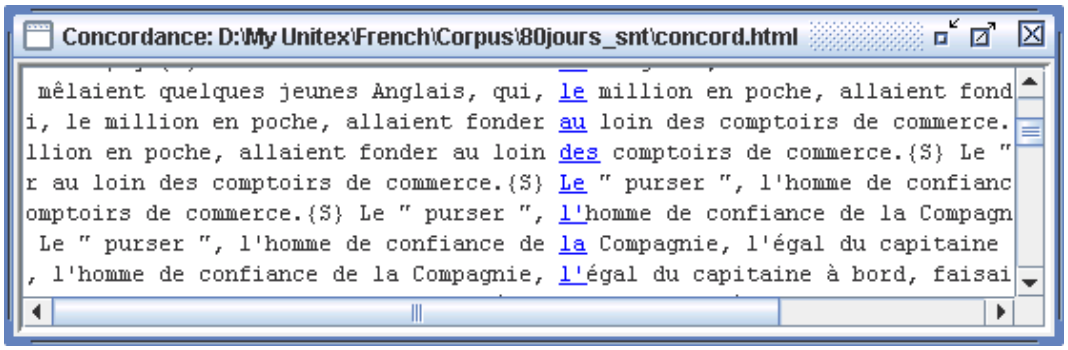


FIGURE 7.34 – Une concordance surprenante pour le motif `<le.DET>`

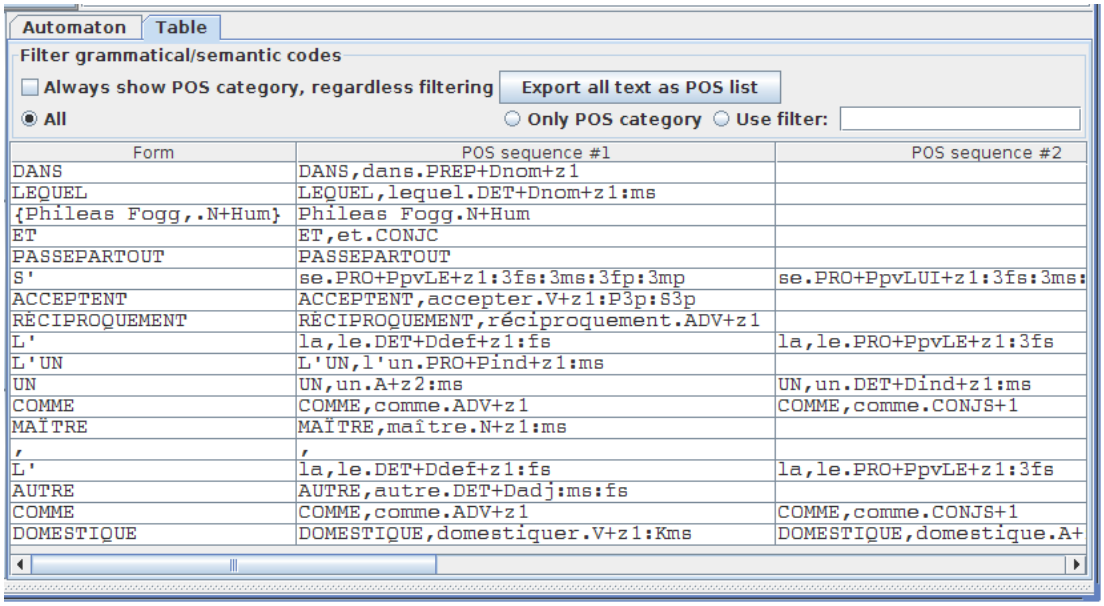
être plus longues que les tokens si ce sont des étiquettes de mots composés, ou même plus courtes, si l'automate comporte une analyse mophologique comme `un` comme le montre la figure 3.28, page 75.

- même sans entrer dans le mode morphologique, on peut définir des variables de dictionnaire (cf. section 6.4.4). Ensuite, on peut extraire de ces variables la forme fléchie, la forme canonique et les codes des entrées lexicales correspondantes, leur catégorie grammaticale, leurs codes sémantiques, leurs codes flexionnels et la valeur `zzz` de l'attribut `yyy` s'il y figure un code de la forme `yyy=zzz`.

7.8 Affichage de la Table

Les automates de phrases peuvent être affichées sous forme de tableau. Pour ce faire, il vous suffit de sélectionner l'onglet "Tableau" dans la zone automate de texte. Vous verrez alors un tableau comme indiqué sur la figure 7.35.

Ce tableau n'est pas tout à fait équivalent à l'automate de phrase, car il affiche seulement



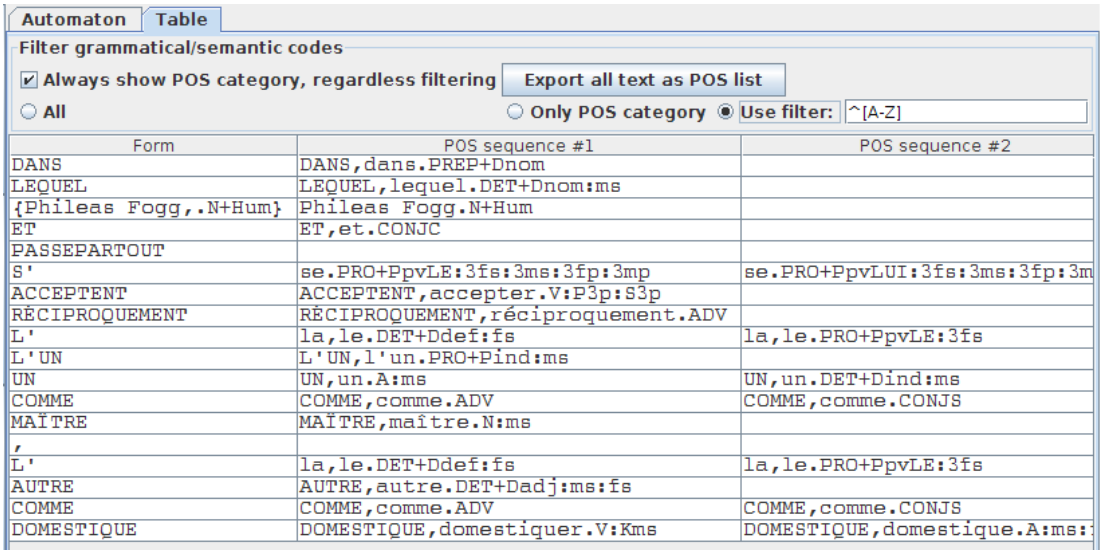
Form	POS sequence #1	POS sequence #2
DANS	DANS,dans.PREP+Dnom+z1	
LEQUEL	LEQUEL,lequel.DET+Dnom+z1:ms	
{Phileas Fogg,.N+Hum}	Phileas Fogg.N+Hum	
ET	ET,et.CONJC	
PASSEPARTOUT	PASSEPARTOUT	
S'	se.PRO+PpvLE+z1:3fs:3ms:3fp:3mp	se.PRO+PpvLUI+z1:3fs:3ms:
ACCEPTENT	ACCEPTENT,accepter.V+z1:P3p:S3p	
RÉCIPROQUEMENT	RÉCIPROQUEMENT,réciproquement.ADV+z1	
L'	la,le.DET+Ddef+z1:fs	la,le.PRO+PpvLE+z1:3fs
L'UN	L'UN,l'un.PRO+Pind+z1:ms	
UN	UN,un.A+z2:ms	UN,un.DET+Dind+z1:ms
COMME	COMME,comme.ADV+z1	COMME,comme.CONJS+1
MAÎTRE	MAÎTRE,maître.N+z1:ms	
,	,	
L'	la,le.DET+Ddef+z1:fs	la,le.PRO+PpvLE+z1:3fs
AUTRE	AUTRE,autre.DET+Dadj:ms:fs	
COMME	COMME,comme.ADV+z1	COMME,comme.CONJS+1
DOMESTIQUE	DOMESTIQUE,domestiquer.V+z1:Kms	DOMESTIQUE,domestique.A+

FIGURE 7.35 – Affichage d'un tableau

les POS possibles pour chaque mot simple ou composé. Il devrait être considéré comme une vue approximative et compacte des informations contenues dans l'automate. Vous pouvez également filtrer les codes grammaticaux / sémantiques à afficher. Choisissez "All" et vous verrez tous les codes. Choisissez "Only POS category" les premiers codes (supposés représenter la catégorie de la POS) seront affichés. Si vous choisissez "Use filter" et écrivez une expression régulière X, les codes non reconnus par X seront supprimés. Toute expression rationnelle POSIX est acceptée en tant que filtre. Vérifiez "Always show POS category", and as said, the POS category will be kept even if not matched by the filter, if any. For instance, Figure 7.36 shows a filtering result, obtained with the filter `^[A-Z]` that matches any code starting with an uppercase letter, thus discarding codes like `z1`.

Le bouton "Export all text as POS list" peut être utilisé pour exporter ce tableau d'affichage de l'ensemble du texte automate dans un fichier texte, en utilisant un format particulier. Actuellement, cette fonctionnalité est expérimentale et peut être modifiée dans le futur. Voici un exemple de sortie :

```
(Je/N:ms:mp)|(Je/PRO/PpvIL:1fs:1ms) (suis/V:P1s)|(suis/V:Y2s:P2s:P1s)
M/N:mp:ms . Mdiba (de/DET/Dind:fp:mp:fs:ms)|(de/PREP)|(de/PREP/z1
+de la/DET/Dind/z1:fs)|(de/PREP/z1+des/DET/Dind/z1:mp:fp)|(de/PREP/z1
+du/DET/Dind/z1:ms)|(de la/DET/Dind/z1:fs)|(des/DET/Dind/z1:mp:fp)|
(du/DET/Dind/z1:ms) LG - ville/N:fs . {S}
```



Form	POS sequence #1	POS sequence #2
DANS	DANS,dans.PREP+Dnom	
LEQUEL	LEQUEL,lequel.DET+Dnom:ms	
{Phileas Fogg,.N+Hum}	Phileas Fogg.N+Hum	
ET	ET,et.CONJC	
PASSEPARTOUT		
S'	se.PRO+PpvLE:3fs:3ms:3fp:3mp	se.PRO+PpvLUI:3fs:3ms:3fp:3m
ACCEPTENT	ACCEPTENT,accepter.V:P3p:S3p	
RÉCIPROQUEMENT	RÉCIPROQUEMENT,réciproquement.ADV	
L'	la,le.DET+Ddef:fs	la,le.PRO+PpvLE:3fs
L'UN	L'UN,l'un.PRO+Pind:ms	
UN	UN,un.A:ms	UN,un.DET+Dind:ms
COMME	COMME,comme.ADV	COMME,comme.CONJS
MAÎTRE	MAÎTRE,maître.N:ms	
,		
L'	la,le.DET+Ddef:fs	la,le.PRO+PpvLE:3fs
AUTRE	AUTRE,autre.DET+Dadj:ms:fs	
COMME	COMME,comme.ADV	COMME,comme.CONJS
DOMESTIQUE	DOMESTIQUE,domestiquer.V:Kms	DOMESTIQUE,domestique.A:ms:

FIGURE 7.36 – Affichage d'une table filtrée

7.9 Le cas particulier du coréen

Le coréen est une langue agglutinante qui possède un système d'écriture spécifique : les mots sont formés de caractères représentant des syllabes et appelés Hangul, mais un caractère Hangul correspond à plusieurs caractères de l'alphabet JAMO. Par exemple, la figure 7.37 donne deux exemples de caractères Hangul suivis de leurs équivalents en alphabet Jamo.

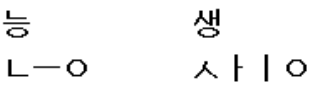


FIGURE 7.37 – Caractères et leurs équivalents en alphabet Jamo

En outre, tous les morphèmes ne correspondent pas à des caractères Hangul. Par exemple, la figure 7.38 montre un token (en vert) analysé comme une combinaison de deux éléments : un verbe et un modifieur. Le problème est que le modifieur n'est formé que d'un caractère Jamo qui se combine avec le dernier caractère Hangul du verbe pour donner le dernier caractère Hangul du mot entier (en vert).

Par conséquent, il est préférable pour les utilisateurs coréens d'écrire des grammaires avec un mélange de caractères Hangul et Jamo. Ainsi, une grammaire comme celle de la figure 7.39 reconnaîtra des séquences comme celles de la figure 7.40.

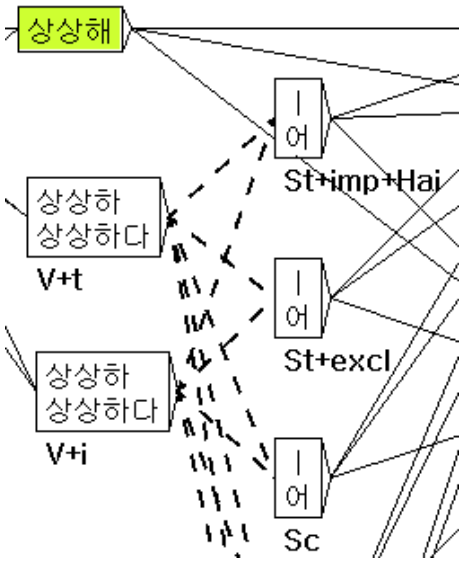


FIGURE 7.38 – Décomposition d'un caractère Hangul

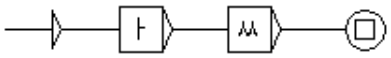


FIGURE 7.39 – Une grammaire avec deux lettres Jamo

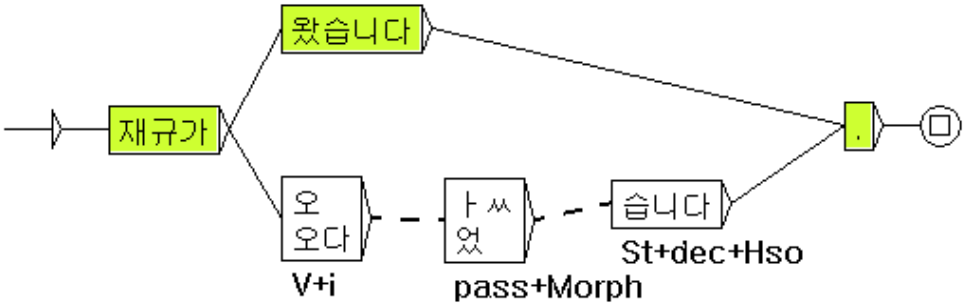


FIGURE 7.40 – Automate d'une phrase reconnue par la grammaire de la figure 7.39

En coréen, l'automate du texte affiche les tokens non étiquetés sur fond lavande [4].

REMARQUES :

1. Les lettres Jamo ne sont pas dans le fichier contenant l'alphabet coréen (`Alphabet.txt`).

4. En coréen, les versions avant la 3.1 bêta rév. 4272 affichent les tokens non étiquetés en vert, comme dans les figures 7.38 et 7.40.

NE LES AJOUTEZ PAS A CE FICHIER, parce que cela occasionnerait des disfonctionnements des programmes.

2. Ce fichier alphabet contient les équivalences entre certains caractères chinois et des Hangul. Dans la pratique, si la grammaire contient un caractère chinois qui possède un Hangul comme équivalent, il reconnaît celui-ci dans l'automate du texte. Par exemple la grammaire de la figure 7.41 reconnaît la phrase de la figure 7.40, parce que l'alphabet contient un équivalent pour ce caractère, comme le montre la figure 7.42.

FIGURE 7.41 – Une grammaire avec un caractère chinois

FIGURE 7.42 – Extrait du fichier contenant l'alphabet coréen

Chapitre 8

Automate de Séquences

La construction de grammaires locales peut être un long processus durant lequel le linguiste répète de nombreuses fois les mêmes opérations. La finalité du programme `Seq2Grf` est de produire rapidement et automatiquement des grammaires locales.

Ce programme peut être utilisé en ligne de commande ou en cliquant sur "Construct Sequences Automaton" dans le menu "Text". L'utilisation de la commande `Seq2Grf` est décrite à la section 14.34.

Pour un document donné (`TEILite` ou des fichiers au format `txt` ou `SNT` quand ils sont prétraités pour cette tâche avec des marqueurs `{STOP}`) ce programme construit un unique automate qui reconnaît toutes les séquences contenues dans le document.

On doit porter une attention particulière à la construction de la liste de séquences qui doivent être reconnues par le graphe.

Ce chapitre présente les formats de fichiers supportés par le programme `Seq2Grf`, la construction de l'automate de séquences et l'utilisation de jokers.

8.1 Corpus de séquences

Nous appelons *corpus de séquences* ou *corpus qualifié* une liste de séquences d'un ou plusieurs mots que l'on veut reconnaître par une grammaire locale représentée par un seul graphe.

Le corpus de séquences est stocké dans un seul fichier qui peut avoir l'un des formats suivants :

- fichiers texte brut, dans lequel les séquences sont délimitées par des fins de lignes
- fichiers `SNT` déjà prétraités par ce menu : les séquences sont délimitées par `{STOP}`
- fichiers `TEILite` dont les séquences sont délimitées par un tag `xml` de la forme :
  `<seg type="sequence">example</seg>}`

Puisque le corpus contient des séquences spécifiques, il doit être fait à la main. Cela signifie que vous devez soit écrire toutes les séquences dans un fichier texte brut et les séparer par une fin de ligne (figure 8.1), soit insérer la balise `XML` spécifique dans un document existant `TEILite` (figure 8.3). Le prétraitement des documents `TXT` ou `XML` génère un fichier `SNT` qui est utilisé pour la construction de l'automate de séquences (figure 8.2). Ce fichier peut être utilisé comme une entrée. Le graphe produit ne reconnaîtra que les séquences qui sont correctement délimitées. La production de grammaires locales est automatique uniquement à partir d'un corpus de séquences bien définies. Si vous disposez d'un tel corpus, alors le gain de temps est considérable.

```
Tomorrow
this week
twice a month
as soon as possible
in the next few days
```

FIGURE 8.1 – TXT

```
Tomorrow{STOP}
this week{STOP}
twice a month{STOP}
as soon as possible{STOP}
in the next few days
```

FIGURE 8.2 – SNT

```
<?xml version="1.0" encoding="UTF-16LE"?>
<!DOCTYPE xml SYSTEM "teilite.dtd">
<TEI.2 lang="fr">
<teiHeader>
[...]
</teiHeader>
<text>
<body>
<p id="1">I am going to see three of them  <seg type="sequence">tomorrow</seg>.</p>
<p id="2">Here are suggestions of things to do <seg type="sequence">this week</seg> in London.</p>
<p id="3">These meetings will be held at least <seg type="sequence">twice a month</seg>.</p>
<p id="4">We will bring forward an amended proposal <seg type="sequence">as soon as possible</seg>.</p>
<p id="5">We will have to decide <seg type="sequence">in the next few days</seg> how we take all this together.</p>
</body>
</text>
</TEI.2>
```

FIGURE 8.3 – TEILite

8.2 Utilisation

Pour créer un automate de séquences, cliquez sur "Séquence Construct Automate" dans le menu "Text". Vous verrez alors apparaître la fenêtre de la figure 8.4.

Cette fenêtre vous permet de définir les paramètres pour produire un automate séquence. Vous devez suivre ces trois étapes :

- choisissez le corpus séquences : celui-ci peut être un fichier dont le format est l'un des trois formats décrits dans la section précédente. Le format de fichier est automatiquement détecté en fonction de l'extension de fichier.
- définissez les options spécifiques : "Apply the beautifying algorithm" placera chaque boîte de manière à ce que le graphe résultant soit le plus petit et le plus facile à lire que possible. "Exact case matching" mettra les tokens littéraux entre accolades dans le

graphe afin que ceui-ci ne reconnaisse pas des tokens avec les mêmes lettres mais avec des différences de casse.

Vous pouvez définir des options supplémentaires pour produire un graphe qui permet une reconnaissance approximative : vous pouvez fixer le nombre de jokers à utiliser pour produire de nouvelles séquences dérivées des séquences du corpus original, et choisir le joker approprié. Tous les détails sur l'utilisation des jokers se trouvent dans la section 8.3

- choisissez le répertoire où le graphe sera enregistré.

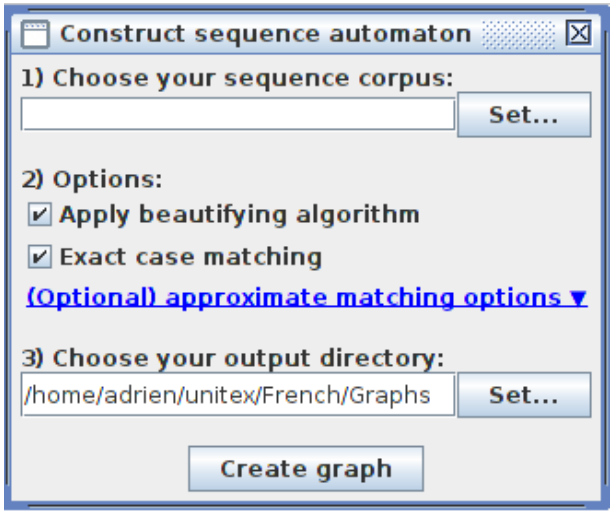


FIGURE 8.4 – Menu automate de séquences

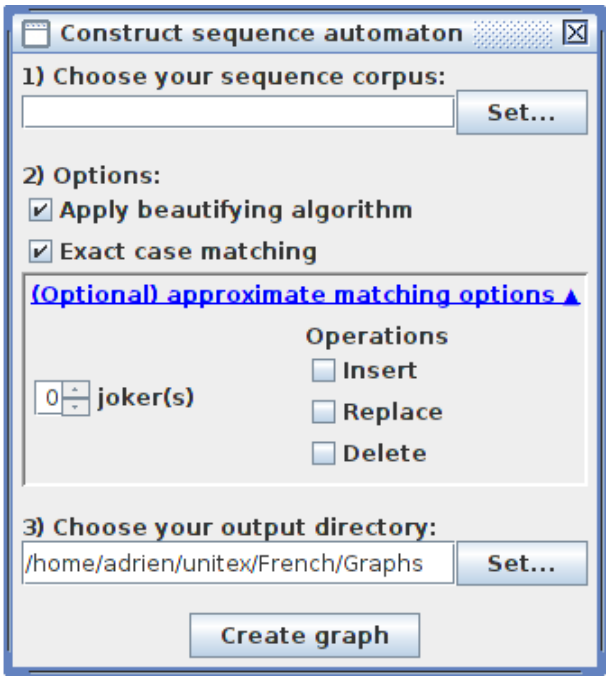


FIGURE 8.5 – Menu options de l'automate de séquences

Vous pouvez voir figures 8.6 et 8.7 les graphes sans jokers produits avec ou sans "beautify".

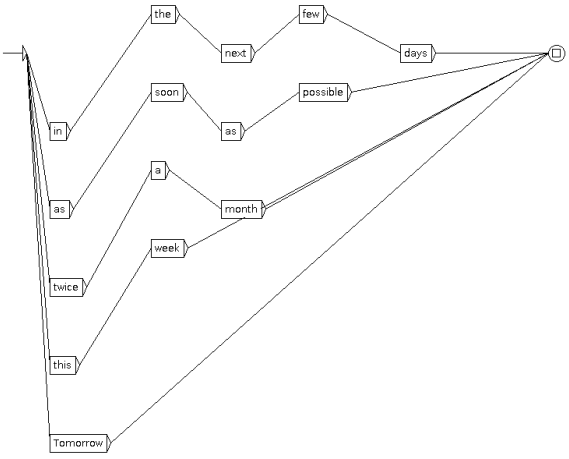


FIGURE 8.6 – Automate sans l'option "beautify"

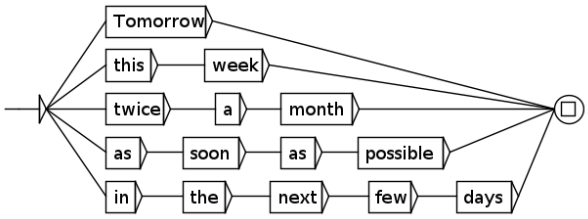


FIGURE 8.7 – Automate avec l'option "beautify"

8.3 Recherche par approximation

Lorsque vous effectuez un "Locate" sur un texte en utilisant un graphe produit avec le programme `Seq2Grf`, vous trouverez uniquement des séquences présentes dans le corpus de séquences original. Des séquences proches de celles du corpus original peuvent être présentes dans le texte et être ignorées parce qu'elles ne figurent pas dans ce corpus. Ces séquences devraient être incluses dans l'automate de séquences. Afin d'inclure ces séquences, vous devez appliquer les trois sortes de jokers et produire ainsi un graphe qui reconnaît toutes les séquences du corpus, et les nouvelles séquences . Chaque joker, permet d'appliquer une opération pour générer de nouvelles séquences.

- insertion : pour chaque séquence, ajouter à l'automate toutes les séquences où `<TOKEN>` a été inséré entre deux mots de la séquence originale.
- remplacement : pour chaque séquence, ajouter à l'automate toutes les séquences où i tokens ont été remplacés par `<TOKEN>`
- suppression : pour chaque séquence, ajouter à l'automate toutes les séquences où un token a été supprimé

Chacune de ces opérations peut être appliquée plusieurs fois aux séquences originales. L'application de cette grammaire à un texte permet d'introduire des approximations dans la recherche des séquences du texte.

Si les jokers sont utilisés, les graphes produits suivent les règles suivantes :

- les séquences originales et les séquences dérivées sont incluses dans l'automate,
- aucune séquence vide, ni une séquence composée uniquement de jokers ne seront ajoutées à ce graphe (de telles séquences peuvent être produites par des suppressions ou des remplacements sur des séquences courtes)

- pas d'insertion de jokers au début ou à la fin d'une séquence
- chaque token d'une séquence y compris le premier et le dernier peuvent être remplacés par un joker

Les graphes produits en utilisant des jokers contiennent de nombreuses séquences erronées et doivent être confrontées avec le corpus au moyen de `Locate` pour ne garder que les séquences pertinentes. Ces séquences peuvent être utilisées pour produire un nouveau graphe, que vous voudrez peut-être garder.

Le graphe de la figure 8.8 a été produit avec remplacement de 1 token et avec l'option "beautifying" activée. (cf. figure 8.2)

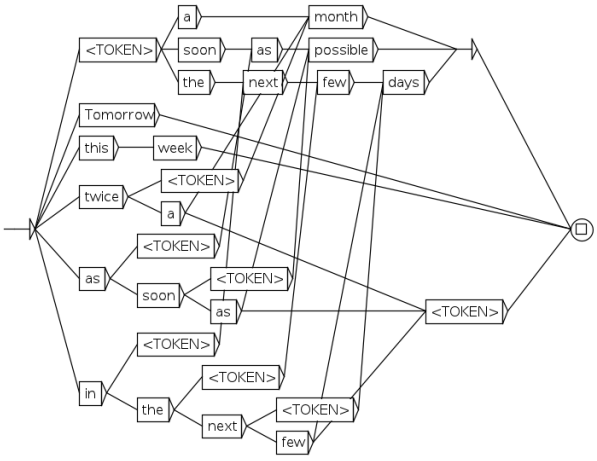


FIGURE 8.8 – Automate avec un remplacement permis

Chapitre 9

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.

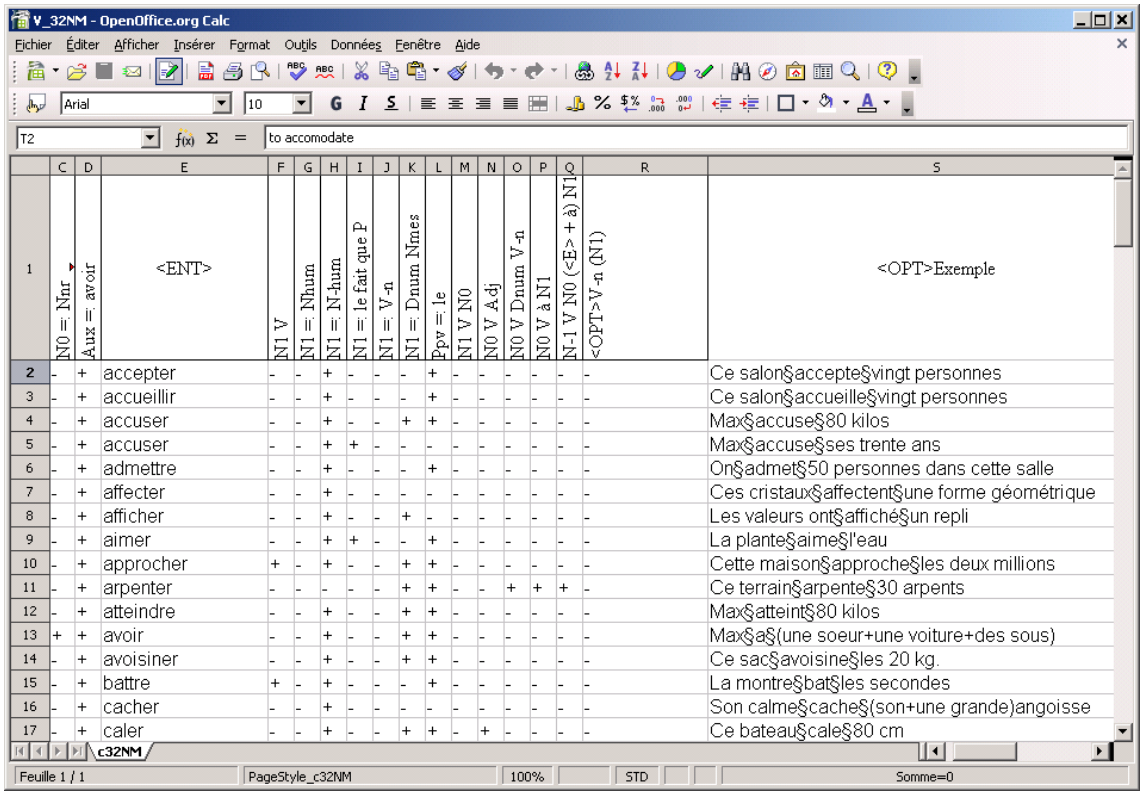
La première partie de ce chapitre présente le formalisme de ces tables. La seconde partie décrit les graphes paramétrés et le mécanisme de génération automatique de graphes à partir d'une table de lexique-grammaire.

9.1 Les tables de lexique-grammaire

Le lexique-grammaire est une méthodologie qui a été développée par Maurice Gross et son équipe du LADL ([9], [10], [39], [52], [50], [51], [49], [48], [45], [44], [43], [42], [41], [66], [87]) 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. Pour plus d'information consulter `http://infolingu.univ-mlv.fr`, où des tables du lexique-grammaire sont librement téléchargeables.

Ce type de description a aussi été utilisé pour les adjectifs ([69]), les noms prédicatifs ([34], [35], [33], [40], [84]), adverbes ([46], [71]), ou les expressions figées, dans de nombreuses langues ([14], [26], [27], [76], [77], [81], [91], [92], [93], [85], [82], [47]).

La figure 9.1 montre un exemple de table de lexique-grammaire. Cette table concerne les verbes admettant un complément numérique.



V_32NM - OpenOffice.org Calc

Fichier Éditer Afficher Insérer Format Outils Données Fenêtre Aide

Arial 10

T2 to accomodate

	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	N0 = Nnr	Aux = avoir	<ENT>	N1 V	N1 = Nhum	N1 = N-hum	N1 = le fait que P	N1 = V-n	N1 = Dnum Nmes	Ppv = le	N1 V N0	N0 V Adj	N0 V Dnum V-n	N0 V à N1	N-1 V N0 (<E> + à) N1	<OPT>V-n (N1)	<OPT>Exemple
2	-	+	accepter	-	-	+	-	-	-	+	-	-	-	-	-	-	Ce salon§accepte§vingt personnes
3	-	+	accueillir	-	-	+	-	-	-	+	-	-	-	-	-	-	Ce salon§accueille§vingt personnes
4	-	+	accuser	-	-	+	-	-	+	+	-	-	-	-	-	-	Max§accuse§80 kilos
5	-	+	accuser	-	-	+	+	-	-	-	-	-	-	-	-	-	Max§accuse§ses trente ans
6	-	+	admettre	-	-	+	-	-	-	+	-	-	-	-	-	-	On§admet§50 personnes dans cette salle
7	-	+	affecter	-	-	+	-	-	-	-	-	-	-	-	-	-	Ces cristaux§affectent§une forme géométrique
8	-	+	afficher	-	-	+	-	-	+	-	-	-	-	-	-	-	Les valeurs ont§affiché§un repli
9	-	+	aimer	-	-	+	+	-	-	+	-	-	-	-	-	-	La plante§aime§l'eau
10	-	+	approcher	+	-	+	-	-	+	+	-	-	-	-	-	-	Cette maison§approche§les deux millions
11	-	+	arpenter	-	-	-	-	-	+	+	-	-	+	+	+	-	Ce terrain§arpente§30 arpents
12	-	+	atteindre	-	-	+	-	-	+	+	-	-	-	-	-	-	Max§atteint§80 kilos
13	+	+	avoir	-	-	+	-	-	+	+	-	-	-	-	-	-	Max§a§(une soeur+une voiture+des sous)
14	-	+	avoisiner	-	-	+	-	-	+	+	-	-	-	-	-	-	Ce sac§avoisine§les 20 kg.
15	-	+	battre	+	-	+	-	-	-	+	-	-	-	-	-	-	La montre§bat§les secondes
16	-	+	cacher	-	-	+	-	-	-	-	-	-	-	-	-	-	Son calme§cache§(son+une grande)angoisse
17	-	+	caler	-	-	+	-	-	+	+	-	+	-	-	-	-	Ce bateau§cale§80 cm

c32NM

Feuille 1 / 1 PageStyle_c32NM 100% STD Somme=0

FIGURE 9.1 – Table de lexique-grammaire 32NM

9.2 Conversion d'une table en graphes

9.2.1 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.

9.2.2 Format de la table

Les tables de lexique-grammaire sont généralement codées à l'aide d'un tableur comme OpenOffice.org Calc ([75]). 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 9.2. Choisissez le codage "Unicode", sélectionnez la tabulation comme séparateur de colonnes, et ne précisez pas de délimiteur de texte.

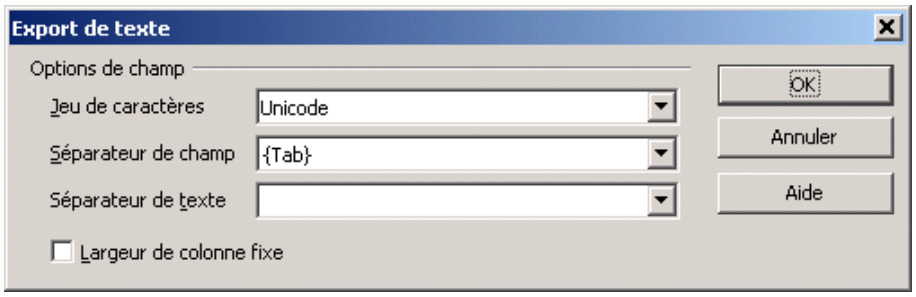


FIGURE 9.2 – Configuration de la sauvegarde d'une table avec OpenOffice.org Calc

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.

9.2.3 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 `@` 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 9.3 montre un exemple de graphe paramétré conçu pour être appliqué à la table de lexique-grammaire table 31H présentée sur la figure 9.4.

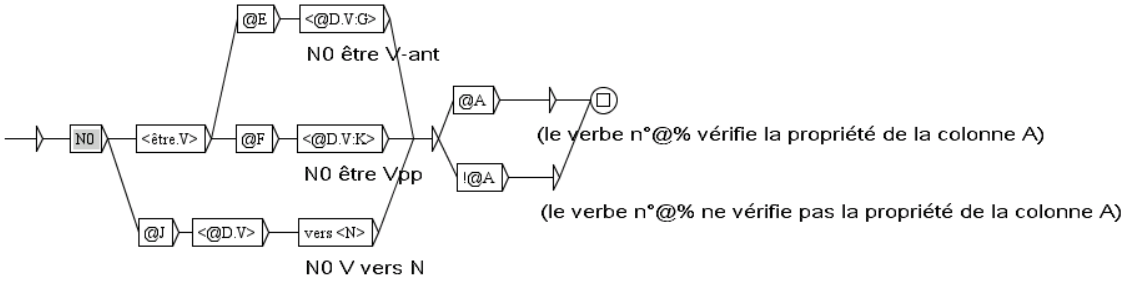
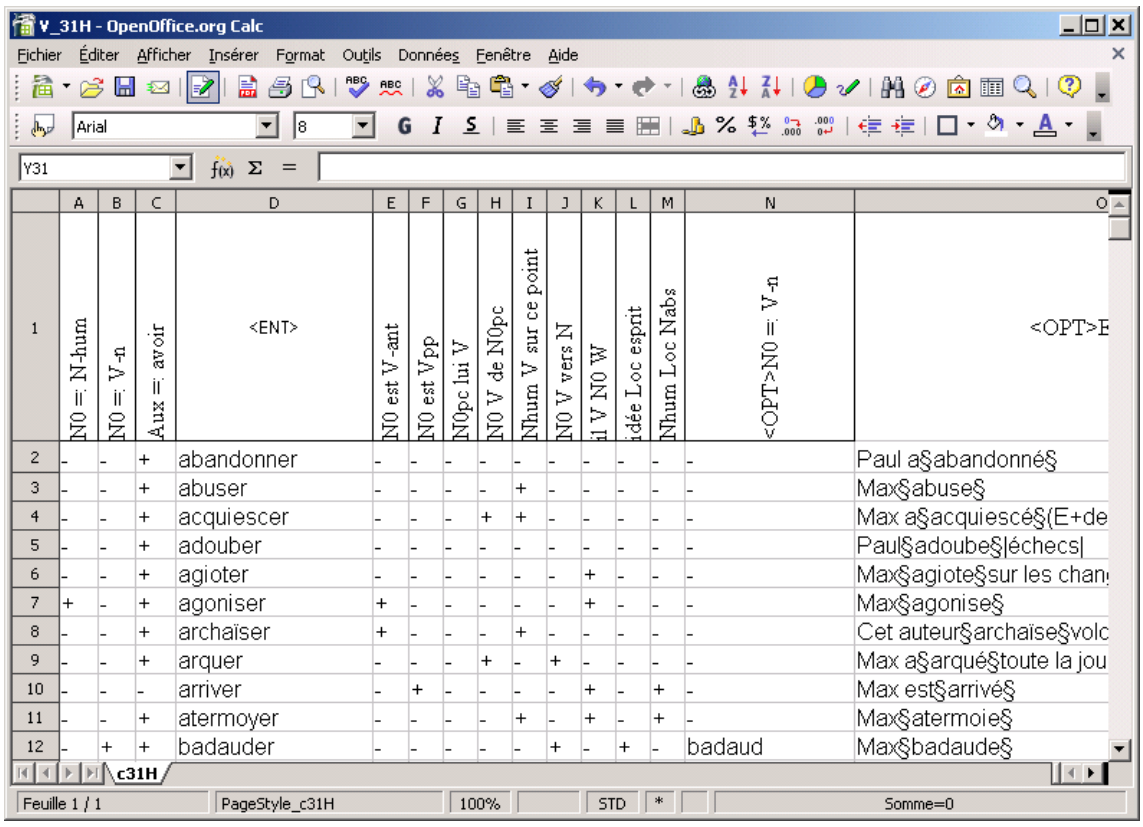


FIGURE 9.3 – Exemple de graphe paramétré



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	N0 = N-hum	N0 = V-n	Aux = avoir	<ENT>	N0 est V-ant	N0 est Vpp	N0pc lui V	N0 V de N0pc	Nhum V sur ce point	N0 V vers N	li V N0 W	idée Loc esprit	Nhum Loc Nabs	<OPT>N0 = V-n	<OPT>E
2	-	-	+	abandonner	-	-	-	-	-	-	-	-	-	-	Paul a§abandonné§
3	-	-	+	abuser	-	-	-	-	+	-	-	-	-	-	Max§abuse§
4	-	-	+	acquiescer	-	-	-	+	+	-	-	-	-	-	Max a§acquiescé§(E+de
5	-	-	+	adouber	-	-	-	-	-	-	-	-	-	-	Paul§adoube§|échecs|
6	-	-	+	agioter	-	-	-	-	-	-	+	-	-	-	Max§agiote§sur les chan
7	+	-	+	agoniser	+	-	-	-	-	-	+	-	-	-	Max§agonise§
8	-	-	+	archaïser	+	-	-	-	+	-	-	-	-	-	Cet auteur§archaïse§volc
9	-	-	+	arquer	-	-	-	+	-	+	-	-	-	-	Max a§arqué§toute la jou
10	-	-	-	arriver	-	+	-	-	-	-	+	-	+	-	Max est§arrivé§
11	-	-	+	atermoyer	-	-	-	-	+	-	+	-	+	-	Max§atermoie§
12	-	+	+	badauder	-	-	-	-	-	+	-	+	-	badaud	Max§badaude§

FIGURE 9.4 – Table de lexique-grammaire 31H

9.2.4 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 9.5). La table doit avoir été préalablement convertie en texte Unicode.

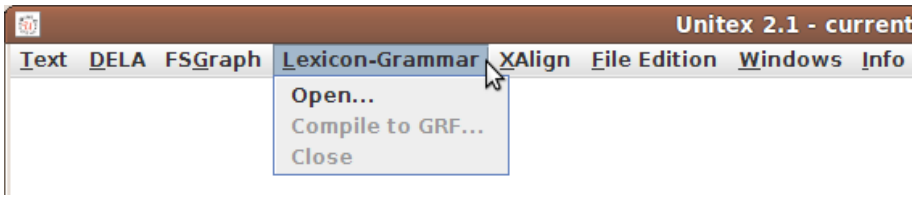
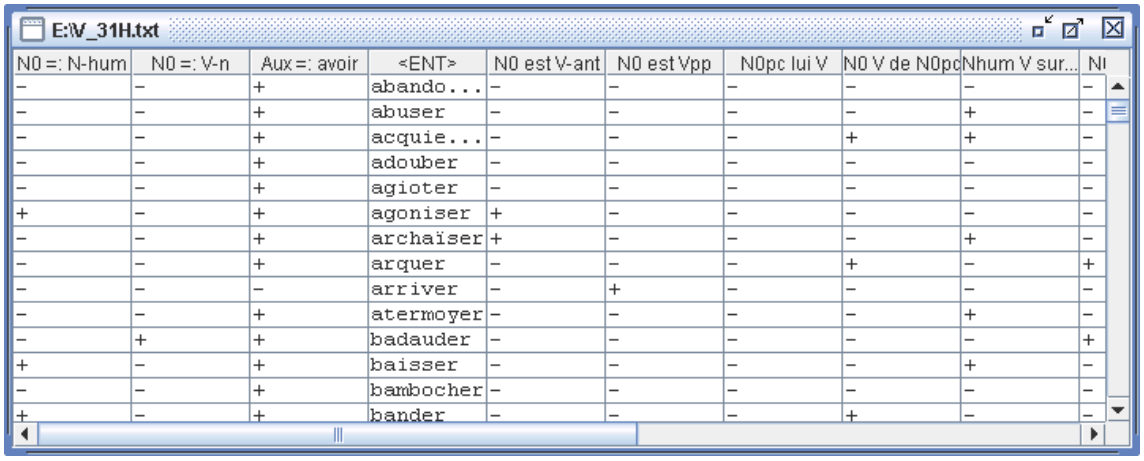


FIGURE 9.5 – Menu "Lexicon-Grammar"

La table sélectionnée est alors affichée dans une fenêtre (voir figure 9.6). Si elle n'apparaît pas sur l'écran, elle peut être occultée par d'autres fenêtres Unitex.


E:\V_31H.txt

N0 =: N-hum	N0 =: V-n	Aux =: avoir	<ENT>	N0 est V-ant	N0 est Vpp	N0pc lui V	N0 V de N0pc	Nhum V sur...	Nl
-	-	+	abando...	-	-	-	-	-	-
-	-	+	abuser	-	-	-	-	+	-
-	-	+	acquie...	-	-	-	+	+	-
-	-	+	adouber	-	-	-	-	-	-
-	-	+	agioter	-	-	-	-	-	-
+	-	+	agoniser	+	-	-	-	-	-
-	-	+	archaïser	+	-	-	-	+	-
-	-	+	arquer	-	-	-	+	-	+
-	-	-	arriver	-	+	-	-	-	-
-	-	+	atermoyer	-	-	-	-	+	-
-	+	+	badauder	-	-	-	-	-	+
+	-	+	baisser	-	-	-	-	+	-
-	-	+	bambocher	-	-	-	-	-	-
+	-	+	bander	-	-	-	+	-	-

FIGURE 9.6 – Displaying a table

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

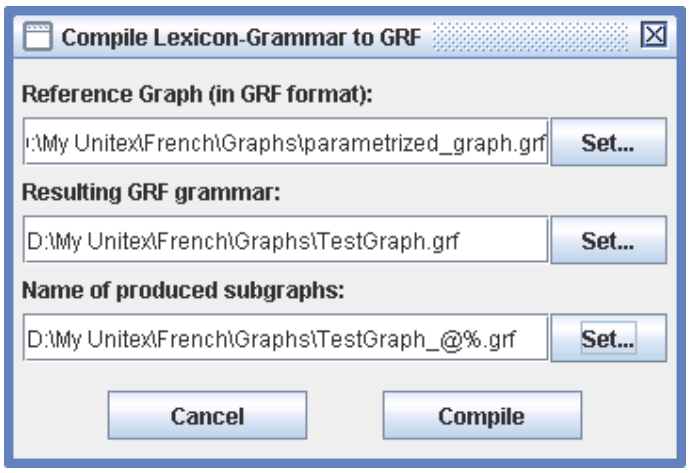


FIGURE 9.7 – Configuration de la génération automatique de graphes

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`" et si les sous-graphes sont nommés "`TestGraph_@%.grf`", le sous-graphe généré à partir de la 16e ligne sera nommé "`TestGraph_0016.grf`".

Les figures 9.8 et 9.9 montrent deux graphes générés en appliquant le graphe paramétré de la figure 9.3 à la table 31H.

La figure 9.10 montre le graphe principal obtenu.

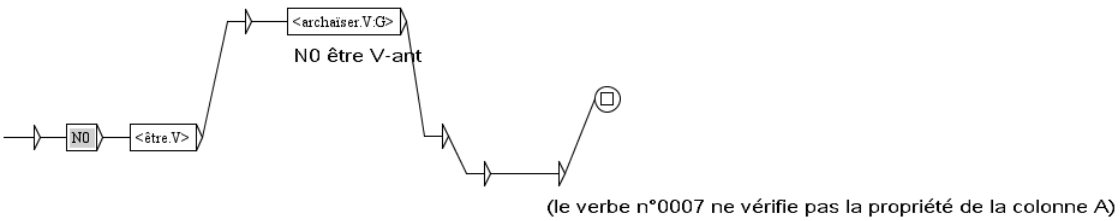


FIGURE 9.8 – Graphe généré pour le verbe `archaïser`

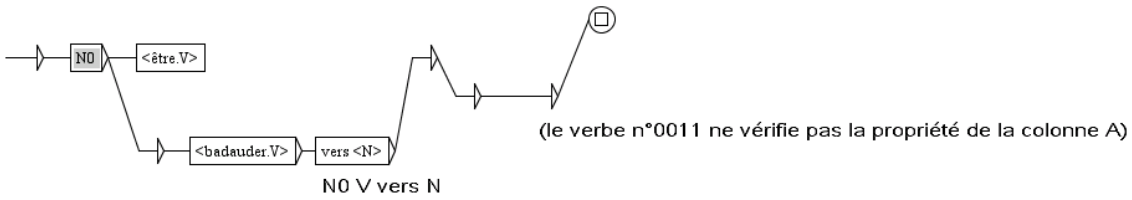


FIGURE 9.9 – Graphe généré pour le verbe `badauder`

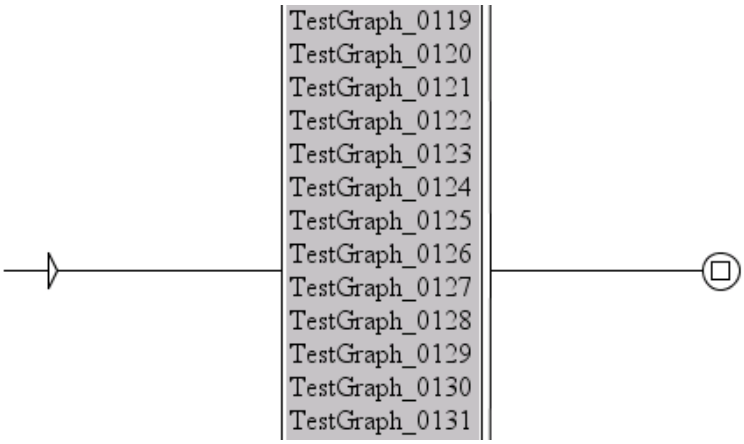


FIGURE 9.10 – Graphe principal appelant tous les graphes générés

Chapitre 10

Alignement de texte

Le principe de l'alignement de texte est simple : quand on aligne deux textes ou plus, le premier est considéré comme le texte source et les autres comme ses traductions. L'alignement s'effectue au niveau de la phrase, parce l'alignement au niveau des mots n'est pas encore possible et certainement pas pertinent. On peut chercher une expression A dans un des textes puis rechercher ses traductions dans les phrases alignées avec celles contenant A.

Pour ajouter cette fonctionnalité à Unitex, Patrick Watrin a intégré l'outil d'alignement de texte Open Source `XAlign`, développé au LORIA ([68]). Dans ce chapitre, nous expliquons comment utiliser le module d'alignement. Le lecteur intéressé par les détails d'intégration de `XAlign` peut consulter [23] ou [78], et [95] pour avoir une idée de ce qui peut être fait avec ce module.

10.1 Chargement de textes

Il faut tout d'abord sélectionner deux textes. Pour cela, allez sur "XAlign>Open files…", et vous verrez le cadre de la figure 10.1. Deux formats de textes peuvent être utilisés : texte brut unicode (comme pour les corpus) ou texte au format TEI (format de type XML ; voir [55]). Dans le dernier champ, choisissez un fichier XML d'alignement, si vous en avez déjà construit un. Si vous choisissez un texte brut, Unitex doit construire une version TEI de votre texte (pour plus de détails, voir section 14.51 au sujet du programme `XMLizer`). Quand vous cliquez sur "OK", le nom d'un fichier XML vous est demandé comme le montre la figure 10.2. Unitex construit alors, si besoin est, les versions XML de vos textes, et affiche le cadre de la figure 10.3. Comme vous pouvez le constater, chaque texte est représenté sous la forme d'une liste, chaque cellule comportant une phrase.

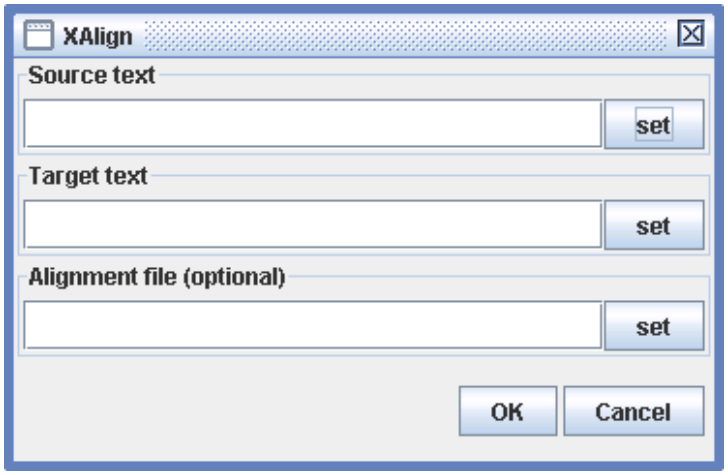


FIGURE 10.1 – Fenêtre de sélection des textes à aligner

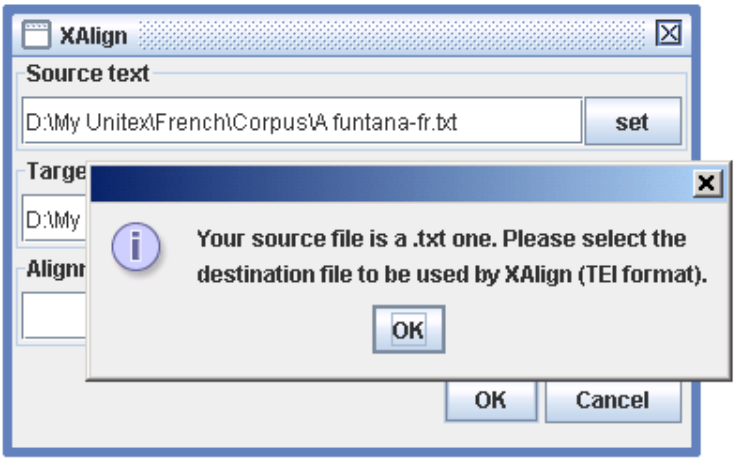


FIGURE 10.2 – Attention aux textes bruts

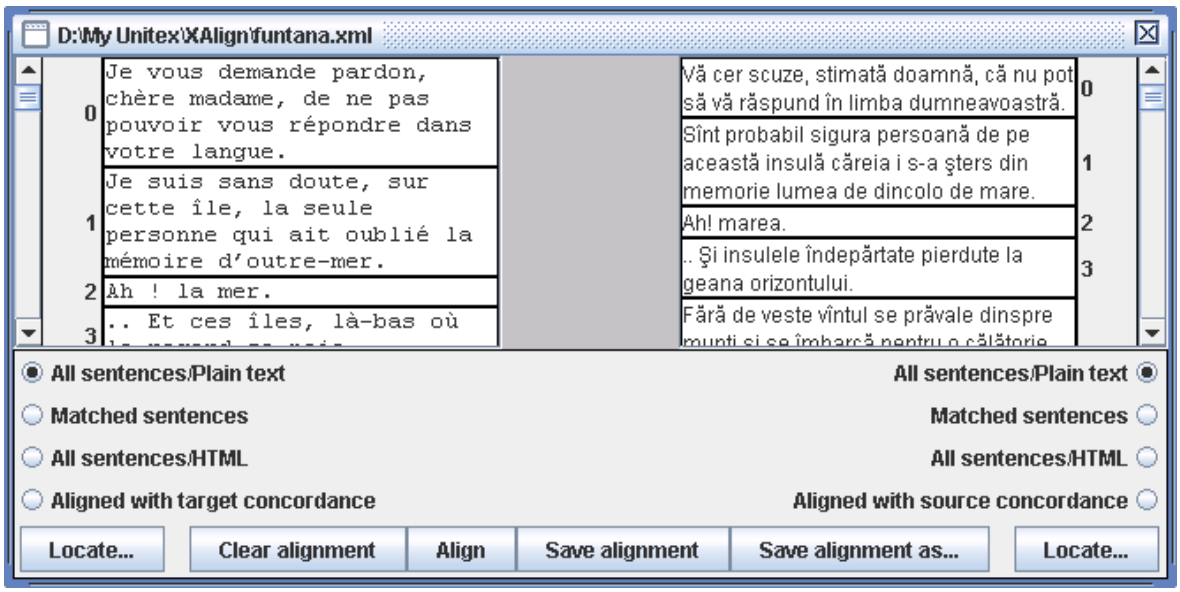


FIGURE 10.3 – Cadre d'alignement de texte

10.2 Aligner des textes

Une fois les textes chargés, vous pouvez les aligner en cliquant sur "Align". Le nom du fichier XML contenant toutes les informations d'alignement vous sera demandé. Ensuite, Unitex lance le programme `XAlign`, vous visualisez alors l'alignement sous la forme de traits rouges entre les phrases alignées comme le montre la figure 10.4.

Il est possible d'éditer les liens d'alignement avec la souris. Le fait de cliquer sur un lien le supprime. Pour ajouter un lien (ou le supprmer, s'il existe déjà), sélectionnez une phrase avec la souris (dans le texte de votre choix, source ou destination) et déplacez la souris jusqu'à la phrase correspondante dans l'autre texte. Le lien en cours de création apparaît en jaune comme le montre la figure 10.5. En le sélectionnant, ce lien est effectivement ajouté et devient rouge. Une fois toutes les corrections effectuées, sauvegardez le nouvel alignement au moyen des boutons "Save alignment" "Save alignment as...".

Une caractéristique intéressante du programme XAlign est qu'il est *réentrant*. Cela siginifie que vous pouvez utiliser un alignement existant comme un ensemble de liens obligatoires en tant qu'entrées du processus d'alignement. Ceci peut-être très utile si vous souhaitez travailler avec des *mots apparentés*. Pour plus de détails au sujet des mots apparentés et de XAlign, voir [78].

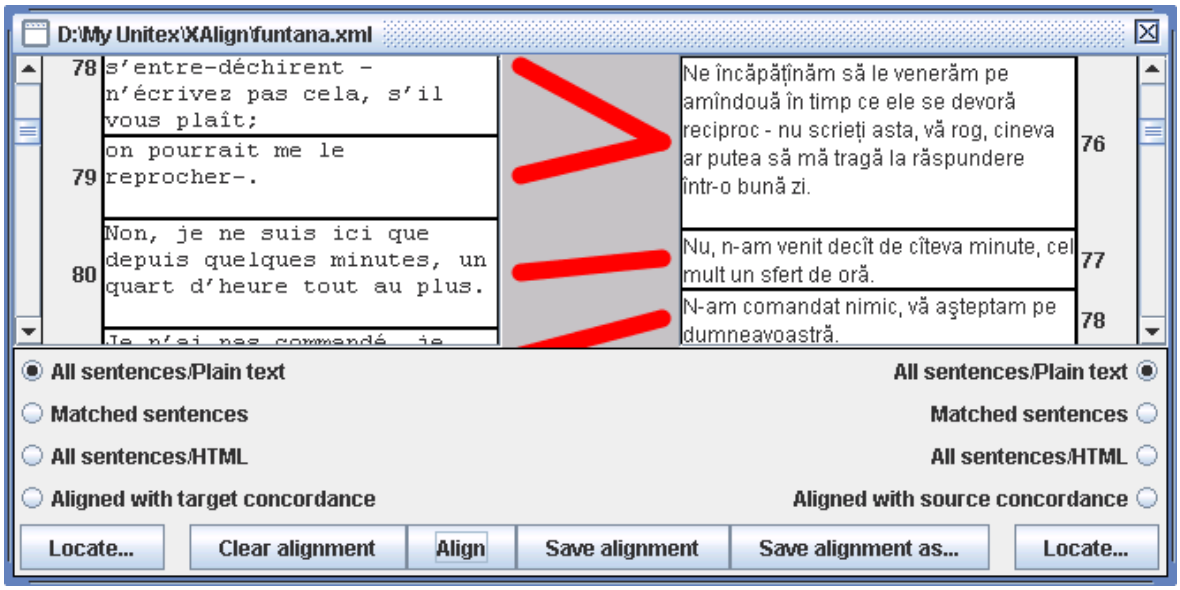


FIGURE 10.4 – Phrases alignées

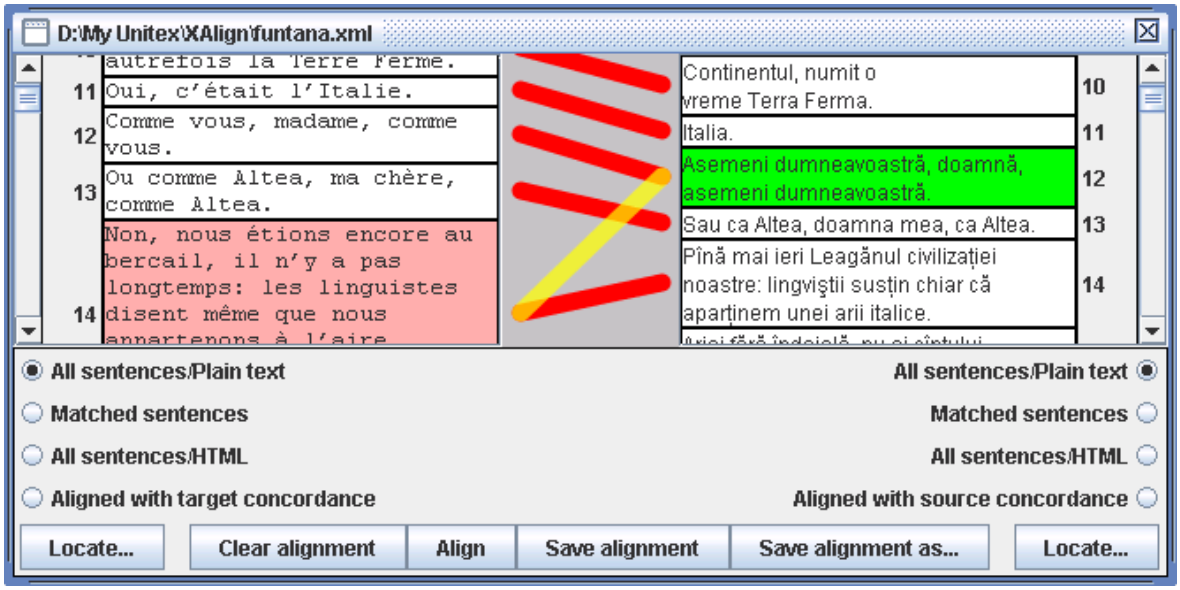


FIGURE 10.5 – Ajout d'un lien

10.3 Recherche de motifs

Vous pouvez effectuer des recherches de motifs sur chacun des textes, en cliquant sur son bouton "Locate". La première fois, Unitex vous demandera de construire une version de travail de votre texte, comme le montre la figure 10.6. Cette version sera prétraitée en tenant compte de la langue du texte (en particulier, les dictionaires sélectionnés par défaut seront appliqués).

ATTENTION : la langue du texte est déterminée à l'aide de son nom complet. Par exemple, si votre fichier se trouve dans le répertoire `.../MyUnitex/Klingon/Corpus`, la langue considérée sera `Klingon`. Donc, si votre texte n'est pas dans un sous-répertoire de votre répertoire de travail, sa langue ne sera pas correctement identifiée.

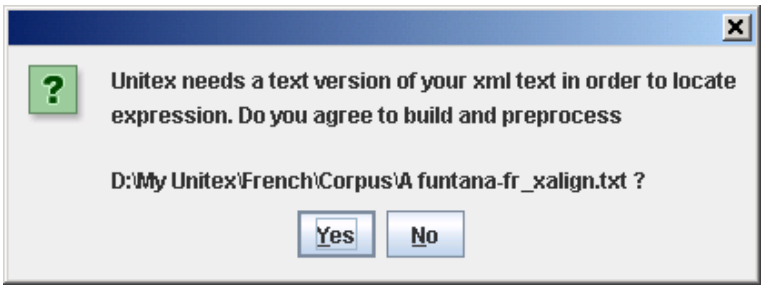


FIGURE 10.6 – Unitex doit construire une version de travail du texte

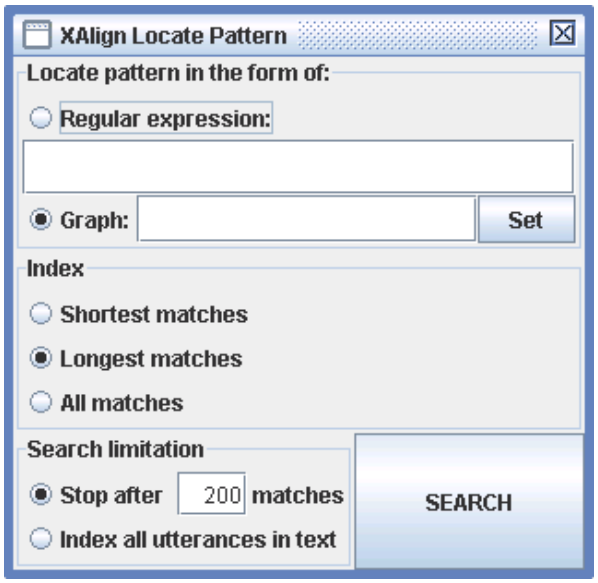


FIGURE 10.7 – Recherche de motifs sur des textes alignés

Une fois qu'Unitex a créé et prétraité la version de travail de votre texte, vous pouvez effec-

tuer une requête comme indiqué figure 10.7. Celle-ci étant faite par le programme `Locate`, elle est tout à fait semblable à celles effectuées sur un corpus normal. La seule restriction est qu'il est impossible d'utiliser les sorties des grammaires si elles en comportent.

Recherchons par exemple le motif `<manger>` dans le texte de notre exemple. Dans un premier temps, nous n'obtenons aucun résultat, car nous n'avons pas encore changé le mode d'affichage du texte, qui par défaut est "All sentences/Plain text". En sélectionnant "Matched sentences", nous voyons seulement les phrases qui contiennent des occurrences, habituellement surlignées en bleu comme le montre la figure 10.8. En cliquant sur "All sentences/HTML" nous obtenons toutes les phrases, avec les occurrences surlignées en bleu.

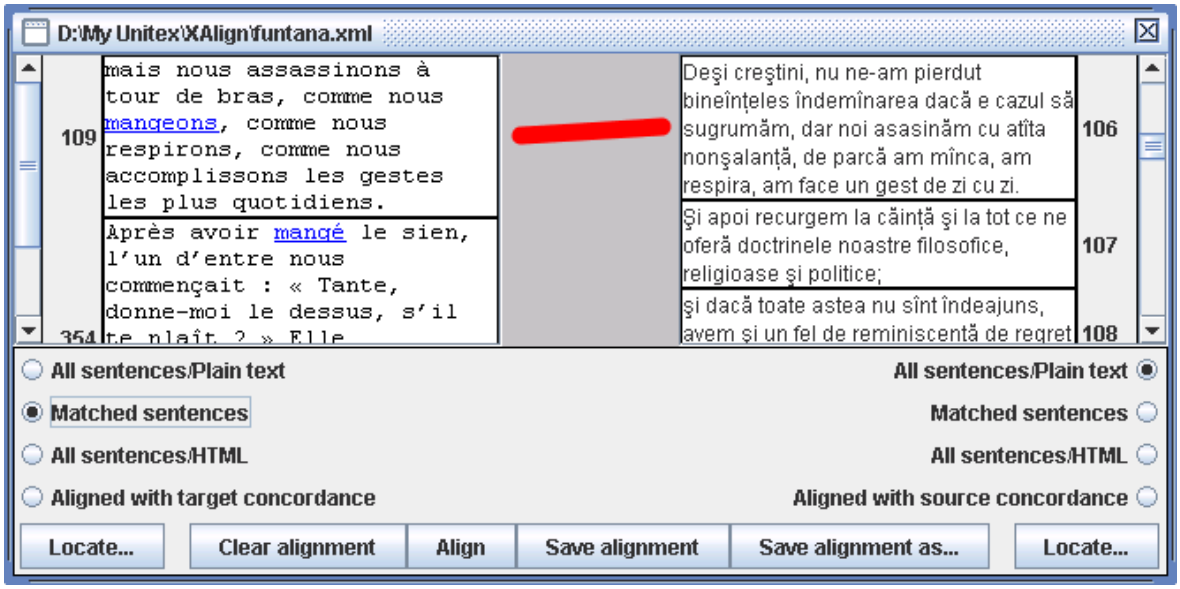


FIGURE 10.8 – Affichages des phrases reconnues

Pour utiliser des textes parallèles, il est intéressant de retrouver les phrases alignées avec les phrases reconnues. Il suffit pour cela de sélectionner *pour l'autre texte*, le mode d'affichage "Aligned with source concordance". Dans ce mode, Unitex filtre les phrases non liées à des phrases reconnues dans le texte source. Il est ainsi facile de rechercher une expression dans un texte et de trouver la phrase correspondante dans l'autre, comme le montre la figure 10.9.

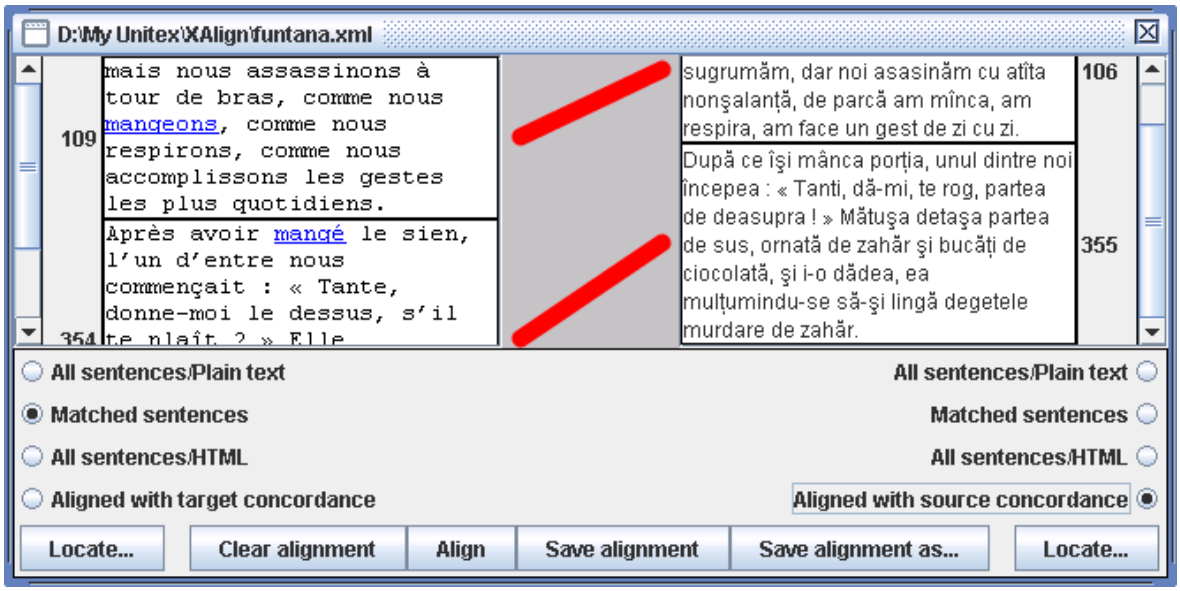


FIGURE 10.9 – Affichages des phrases reconnues et des phrases auxquelles elles sont liées

Chapitre 11

Flexion des mots composés

MULTIFLEX est une plate-forme compatible Unicode de flexion automatique des *mots composés* ou *multi-mots* (en anglais *multi-word units* MWUs). Elle est tout particulièrement conçue pour la création de dictionnaires morphologiques de mots composés. Elle met en œuvre un formalisme fondé sur l'unification ([89]) pour la description du comportement flexionnel des mots composés et suppose l'existence d'un module de flexion des mots simples.

Dans ce chapitre, nous présentons la notion de mots composés et nous décrivons la manière de les fléchir avec MULTIFLEX.

Ce chapitre est fondé sur le manuel de MULTIFLEX, écrit par Agata Savary, l'auteur de MULTIFLEX.

11.1 Mots composés

Les mots composés (ou MWUs) englobent un ensemble d'objets linguistiques difficiles à définir et contreversés (cf. [53], [18]). Leurs nombreuses définitions linguistiques ou pragmatiques ([5], [22], [67], [4], [37], [3], [90], [38], [13]) reposent sur trois principaux points :

- ils se composent de deux ou plusieurs mots
- ils montrent un certain degré de non-compositionnalité sur le plan morphologique, distributionnel ou sémantique
- ils possèdent un référent constant et unique

Cependant, les notions de base (un mot, un référent, la non-compositionnalité) et les mesures (degré de non-compositionnalité) utilisées dans ces définitions sont elles-mêmes controversées.

De façon pragmatique, nous considérons comme mot composé une séquence d'*unités graphiques* contiguës qui pour des raisons applicatives doivent être listées, décrites, (morphologiquement, syntaxiquement, sémantiquement, etc.) et traitées en tant qu'une seule et même unité.

11.1.1 Description formelle du comportement flexionnel des mots composés

L'objectif principal de MULTIFLEX est le mécanisme de flexion des mots composés. Ce phénomène a été analysé en ce qui concerne l'anglais, le polonais et le français dans [88].

Evidemment, un processus fiable de flexion des mots simples est un prérequis pour la flexion des mots composés. Cependant, cette condition est rarement suffisante. Par exemple, en anglais, les formes plurielles de

- *battle cry*
- *battle royal*
- *battle of nerves*

il n'est pas seulement nécessaire de savoir comment générer les pluriels de *battle, royal* et *cry*, mais aussi de savoir quelles formes fléchies de ces constituants se combinent entre elles :

- *battle cries*
- *battle royals*, or *battles royal*,
- *battles of nerves*

mais pas

* *battles cries*
* *battles royals*
* *battles of nerve*

Formellement, une description explicite et complète du paradigme flexionnel des mots-composés doit répondre aux questions suivantes :

- A quelle catégorie grammaticale appartient le mot composé (nom, adjectif, etc.) et donc quelles catégories flexionnelles (nombre, genre, cas, etc.) sont-elles pertinentes pour lui ? [80] se prononce pour une définiton fondée sur la morphosyntaxe des catégories grammaticales : une catégorie grammaticale devrait pleinement déterminer les catégories flexionnelles dans lesquelles le mot se fléchit ainsi que celles qui sont lexicalement fixées pour le mot. Par exemple, en polonais, un nom a un genre et se fléchit en nombre et en cas.
- Quelles sont les exceptions aux catégories flexionnelles déterminées ci-dessus ? Par exemple, en polonais
  - *wybory powszechne*
    (*élections générales*)

est un nom composé qui n'a pas de forme au singulier (cependant son nom tête *wybory* en possède une).

- Quelles sont les caractéristiques flexionnelles (forme canonique, catégorie grammaticale, paradigme flexionnel, etc.) des constituants simples du mot composé ? Par exemple, en français, *porte* est un verbe non fléchi dans

  – *porte-avion*

  alors que c'est un nom fléchi dans

  – *porte-fenêtre*

  qui prend un *s* au pluriel

  – *portes-fenêtres*

- Comment doit-on combiner les formes fléchies des constituants simples pour générer les formes fléchies du composé ? Par exemple, pour fléchir *battle of nerves* et *battle cry* nous devons fléchir respectivement le premier et le dernier constituant.

11.1.2 Approche lexicale ou grammaticale de la description morphologique

Une étude précédente ([88]) a confirmé le statut particulier des mots composés les situant à la frontière de la morphologie et de la syntaxe. Leur structure compositionnelle suggère une productivité qui ne pourrait guère être traitée sans une approche grammaticale.

Toutefois, certaines de leurs propriétés morphologiques, syntaxiques et sémantiques excluent leur traitement seulement en termes des propriétés de leurs constituants. Par exemple, dans les deux exemples ci-dessous :

- *chief justice*
- *lord justice*

il y a peu d'indices automatiquement accessibles indiquant que le dernier est morphologiquement un syntagme nominal anglais standard prenant un *s* à son dernier constituant au pluriel, tandis que le pluriel de celui-ci a trois variantes :

- *chief justices*
- *lord justices*, *lords justice*, *lords justices*

Ainsi, au moins l'un des exemples ci-dessus doit être considéré comme lexicalisé pour que la flexion automatique soit fiable.

MULTIFLEX met en œuvre un formalisme fondé sur l'unification qui permet de décrire la flexion des mots composés [89]. Ses caractéristiques sont décrites dans la section 11.2. Ce

formalisme nécessite que la description soit *pleinement* lexicalisée : chaque mot composé figurant dans un dictionnaire est muni d'un code (ex : *NC_NN, NC_NN2*, etc.) représentant son paradigme flexionnel, par exemple, dans un format de type DELA :

aircraft carrier(carrier.N1 :s),**NC_NN**
chief justice(justice.N1 :s),**NC_NN**
lord(lord.N1 :s) justice(justice.N1 :s),**NC_NN2**
...

Cependant, la grande majorité des mots composés peut être traitée avec un petit nombre de codes. Ainsi, la lexicalisation de la description consiste principalement à définir les mots composés, qui respectent ou ne respectent pas la "grammaire".

11.2 Formalisme de flexion des mots composés

Un formalisme de description de la morphologie des mots composés a été décrit par Agata Savary en 1985 [89]. Il est fondé sur des études sur l'anglais, le polonais et le français, et en outre a été testé pour le serbe [59] et le grec [30]. Il repose sur une représentation indépendante de la langue qui doit être complétée par l'ensemble des éléments caractéristiques d'une langue donnée. Dans cette section, nous donnons une description détaillée de ce formalisme.

11.2.1 Caractéristiques morphologiques de la langue

Lorsque l'on traite les mots composés d'une langue, il faut définir les caractéristiques générales de cette langue. Ces données se trouvent dans deux fichiers textes.

Le fichier `Morphology.txt` indique les catégories grammaticales (nom, adjectif,...), catégories flexionnelles (nombre, genre, cas,...) et leurs valeurs (masculin, féminin, singulier, nominatif,...). Considérons l'exemple suivant :

Polish
<CATEGORIES>
Nb : sing, pl
Case : Nom, Gen, Dat, Acc, Inst, Loc, Voc
Gen : masc_pers, masc_anim, masc_inanim, fem, neu
<CLASSES>)
noun : (Nb,<var>),(Case,<var>),(Gen,<fixed>)
adj :(Nb,<var>),(Case,<var>),(Gen,<var>)
adv :

Le fichier ci-dessus indique que pour le polonais, trois catégories flexionnelles sont considérées : le nombre (*Nb*), le cas (*Case*), et le genre (*Gen*). On donne pour chaque catégorie la liste exhaustive des valeurs qu'elle peut prendre (singulier et pluriel pour le nombre, etc.).

Ensuite, chaque catégorie grammaticale est décrite selon les catégories qui varient avec la flexion, et celles qui sont définies. Par exemple, un nom se fléchit en nombre et en cas et possède un genre défini. Ce type de fichier est nécessaire pour exprimer le fait qu'un certain mot se fléchit en nombre, genre ou cas, sans avoir à énumérer chaque fois les valeurs flexionnelles (singuler, pluriel, masculin, etc.) qu'il accepte.

De façon similaire, pour le français, le fichier `Morphology.txt` ressemble à ceci :

French
<CATEGORIES>
Nb : s, p
Gen : m, f
<CLASSES>
noun : (Nb,<var>),(Gen,<var>)
adj :(Nb,<var>),(Gen,<var>)
adv :

Toutefois, dans les systèmes de flexion existants, de telles descriptions de catégories grammaticales, catégories flexionnelles et valeurs ne sont pas toujours présentes. Par exemple, selon les conventions DELA ([20]) les valeurs morphologiques des mots simples sont des séquences de caractères contigus (e.g. *ms* pour le masculin singulier) sans mention explicite des catégories correspondantes. Afin que le programme soit compatible avec de tels systèmes, on utilise une liste (contenue dans le fichier appelé `Equivalences.txt`) qui décrit quelle caractéristique flexionelle correspond à quelle paire catégorie/valeur dans notre description. Par exemple, les listes suivantes :

$Polish$	$French$
$s : Nb = sing$	$s : Nb = s$
$p : Nb = pl$	$p : Nb = p$
$M : Case = Nom$	$f : Gen = f$
$D : Case = Gen$	$m : Gen = m$
$C : Case = Dat$	
$B : Case = Acc$	
$I : Case = Inst$	
$L : Case = Loc$	
$V : Case = Voc$	
$o : Gen = masc\_pers$	
$z : Gen = masc\_anim$	
$r : Gen = masc\_inanim$	
$f : Gen = fem$	
$n : Gen = neu$	

décrivent les équivalences entre les précédents fichiers `Morphology.txt` du polonais et du français, respectivement, et les caractéristiques représentées par une unique lettre qui peuvent être utilisées dans les dictionnaires DELA pour ces langues dans Unitex.

11.2.2 Décomposition d'un mot composé en constituants

La notion de constituant élémentaire est controversée et varie selon les langues et les systèmes de TAL. Par exemple, dans Unitex, un alphabet, c'est à dire un ensemble de caractères, est d'abord défini pour chaque langue. Tout caractère n'appartenant pas à l'alphabet est appelé séparateur. Un constituant élémentaire est aussi bien un simple séparateur (habituellement un signe de ponctuation, un chiffre, etc.) une séquence de caractères contigus appartenant à l'alphabet (ex *aujourd'hui* comporte selon cette définition, 3 constituants). Dans d'autres systèmes, un constituant peut contenir un signe de ponctuation (e.g. *c'est-à-dire*), ou une limite entre deux constituants peut se produire dans une séquence de caractères alphabétiques (*widział|bym* 'je verrais' , cf. [80]).

Cette variété de définitions possibles d'un constituant a évidemment un impact sur la définition d'un mot composé. Cependant, nous souhaitons que notre formalisme puisse s'adapter à différents systèmes de flexion de "mots simples". Ainsi, la définition d'un constituant est un paramètre de notre système : chaque fois que MULTIFLEX est utilisé avec un module externe pour les mots simples, celui-ci doit décider comment une séquence de caractères est divisée en constituants.

Dans notre formalisme, les constituants sont représentés par des variables numériques *$1, $2, $3*, etc. Par exemple avec Unitex, la séquence

- *Athens '04*

comprend cinq constituants envoyés à MULTIFLEX de cette façon :

$1 = *Athens*
$2 = <space>
$3 = *'*
$4 = *0*
$5 = *4*

Chaque constituant d'un mots composé suceptible d'être fléchi doit être morphologiquement identifié. Cette identification doit permettre de fournir les informations nécessaires afin que n'importe quelle forme fléchie de ce mot puisse être générée à la demande. Par exemple dans :

- *mémoire vive*

nous devons savoir que *vive* est le féminin singulier de *vif*, et ainsi être capable de générer le féminin pluriel, *vives*. Dans MULTIFLEX nous supposons que ce module externe de traitement des mots simples est responsable de leur identification et de la génération de leurs formes fléchies.

Dans Unitex, la génération des formes fléchies est fortement inspirée du système DELA ([20]). Pour générer une ou plusieurs formes fléchies d'un mot, nous devons connaître :

- sa forme canonique
- son paradigme flexionnel (appelé code flexionnel)
- les caractéristiques flexionnelles des formes à produire

Ainsi, dans l'interface Unitex/MULTIFLEX la description d'un mot simple se fait comme suit :

- *vive(vif.A54 :fs)*

où *A54* est le code flexionnel *vif* et *fs* forment la description morphologique de type DELA des caractéristiques présentes dans le fichier `Equivalences.txt` (cf. section 11.2.1). En sachant que *vive* est le féminin singulier de *vif*, on peut demander la génération du pluriel sans avoir à préciser explicitement le genre du pluriel de la forme souhaitée : puisque nous voulons seulement modifier le nombre, le genre reste celui du mot d'origine *vive*, donc féminin.

11.2.3 Paradigme de flexion des mots composés

Dans notre formalisme, la description morphologique des mots-composés repose sur le système DELA dans la mesure où :

- chaque mot composé possède un code flexionnel
- un code flexionnel décrit explicitement chaque forme fléchie en termes de traitement à effectuer sur sa forme canonique, et de caractéristiques à lui associer.

Dans sa version Unitex, MULTIFLEX utilise des codes flexionnels qui renvoient à des graphes Unitex compilés au format `.fst2`. Par exemple, figure 11.1 présente le graphe de flexion pour *battle royal*.

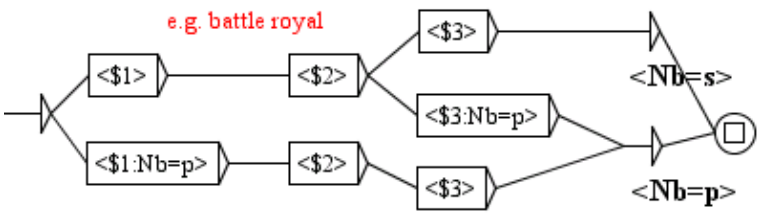


FIGURE 11.1 – Graphe de flexion pour *battle royal*

Selon les conventions d'Unitex, trois constituants sont présents dans *battle royal* : *battle* dénommé *$1*, un espace dénommé *$2*, et *royal* dénommé *$3*. Si des variables apparaissent

seules dans une boîte, le constituant sera le même que dans le lemme du mot composé. Par exemple, <$3> dans le premier chemin du graphe signifie que *royal* doit être recopié tel quel. Si la variable est accompagnée d'assignations de la forme catégories=caractéristiques, le constituant sera fléchi dans la forme demandée. Ainsi <$3 :Nb=p> signifie que la forme plurielle de *royal* est souhaitée.

Pour générer toutes les formes fléchies d'un mot composé, nous devons explorer tous les chemins du graphe. Chaque chemin débute à la flèche droite la plus à gauche et se termine à la boîte encerclée finale. Chaque fois qu'une boîte est atteinte, on réalise l'action qu'elle contient (la recopie ou la flexion d'un constituant) et on accumule les informations présentes sous la boîte. Le total des sorties des boîtes accumulé donne la description morphologique complète de la forme fléchie.

Par exemple, dans le graphe de la figure 11.1 si nous suivons le chemin intermédiaire, extrait à la figure 11.2 :

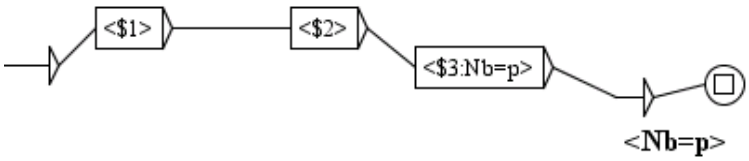


FIGURE 11.2 – Un chemin du graphe de flexion de *battle royal*

nous recopions *battle* (*$1*) et l'espace (*$2*), et nous mettons *royal* au pluriel, ce qui produit la forme du pluriel *battle royals* du mot composé. Le graphe de la figure 11.1 contenant trois chemins différents, l'ensemble des formes fléchies générées pour *battle royal* sera :

battle royal <Nb=s>
battle royals <Nb=p>
battles royal <Nb=p>

Après réécriture de ces formes au format DELACF, on obtient les entrées suivantes :

battle royal,battle royal.N :s
battle royals,battle royal.N :p
battles royal,battle royal.N :p

Remarquons que cette description est indépendante de la manière dont les formes fléchies des mots simples sont générées parce que nous supposons que ce traitement est géré par le module externe de flexion des mots simples. Dans la version Unitex de MULTIFLEX, nous générons le pluriel de *royal* du fait que nous connaissons son code flexionnel *N1* qui correspond au graphe de la figure 11.3.

Dans le paradigme flexionnel d'un mot composé, chaque constituant est accompagné de la catégorie morphologique qui détermine sa flexion. Les catégories inchangées n'ont pas

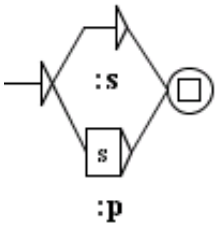


FIGURE 11.3 – Graphe de flexion *N1* pour les mots simples qui se fléchissent comme *royal*

besoin d'être mentionnées. Par exemple, dans *bateau-mouche* les deux noms constituants ont un genre déterminé et ne se fléchissent qu'en nombre : *bateaux-mouches*. C'est pourquoi (figure 11.4) dans le graphe de flexion de ce mot composé, les boîtes correspondantes ne contiennent des assignations de valeurs que pour le nombre. Remarquons que les deux constituants peuvent avoir ou non le même genre, ici *bateau* est masculin tandis que *mouche* est féminin.

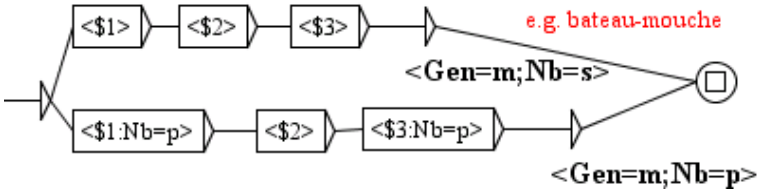


FIGURE 11.4 – Graphe de flexion pour les mots qui se fléchissent comme *bateau-mouche*

Variables d'unification

Une caractéristique importante de notre formalisme est celle des *variables d'unification*. Elles sont représentées par un symbole dollar ($) suivi d'un identifiant pouvant contenir n'importe quel nombre de caractères, comme *$g1*, *$num_10*, *$c*, etc. La figure 11.5 montre un graphe approximativement équivalent[1] à celui de la figure 11.4 dans la mesure où il permet d'engendrer les mêmes formes fléchies pour le même mot composé. Cependant, ici, un chemin unique représente à la fois le singulier et le pluriel. Ceci est rendu possible grâce à la variable *$n* qui est instanciée tour à tour par toutes les valeurs du domaine de sa catégorie (*Nb*), ici *$n=s* puis *$n=p*. Quand une variable d'unification apparait dans une formule du type *Nb=$n*, avec un seul signe égale (=), le système parcourt toutes les valeurs déclarées dans les fichiers de configuration pour cette catégorie (cf. section 11.2.1). Pour chaque valeur, il effectue une nouvelle instanciation de la variable. L'instanciation est la même pour tous les éléments du chemin : si une valeur est attribuée au premier constituant, la même valeur doit être attribuée au troisième, ainsi que pour l'ensemble du mot composé. De même, si nous attribuons à *$n* la valeur *p* dans la première boîte, elle garde cette valeur *p* tout au long du chemin.

1. Même dans le cas où les constituants simples apparaissant dans le lemme d'un mot composé sont déjà au pluriel, comme dans *cross-roads*.

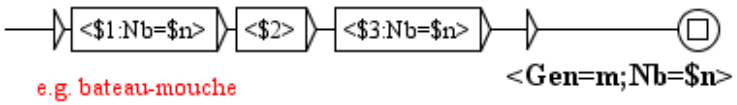


FIGURE 11.5 – Graphe de flexion avec variable pour les mots qui se fléchissent comme *bateau-mouche*

Le graphe de flexion de la figure 11.5 s'applique à la plupart des composés français de types *Nom Nom* et *Nom Adjectif* (*bateau-mouche, ange gardien, circuit séquentiel*, etc.) qui sont de genre masculin : c'est parce que la sortie de la boîte finale contient *Gen=m*. Pour tous les composés des mêmes types, mais de genre féminin, comme *main courante, moissoneuse-batteuse*, etc., un nouveau graphe doit être créé, identique à celui de figure 11.5 jusqu'à la sortie finale contenant *<Gen=f;Nb=$n>*. Ce n'est pas très intuitif puisque *circuit séquentiel* et *main courante* se fléchissent de la même manière, dans la mesure où dans les deux cas nous devons mettre au pluriel le premier et le dernier constituant pour obtenir le pluriel du mot composé.

C'est pourquoi un autre type d'instanciation utilisant l'unification a été introduit. Il s'exprime au moyen de == (par opposition au signe égale simple = comme pour *$n* dans la figure 11.5). Quand une valeur est attribuée à une variable en utilisant ce symbole, la variable est instanciée une seule fois : elle hérite de la catégorie du constituant, telle qu'elle apparaît dans la forme canonique du mot composé. Par exemple, la figure 11.6 contient un graphe décrivant la flexion pour le masculin comme pour le féminin des mots composés de type *Nom Nom* et *Nom Adjectif*. La première boîte contient l'affectation du genre par double signe égale pour la variable *$g*, ce qui signifie que cette variable a pour genre celui du premier constituant. Pour *bateau-mouche* c'est le masculin parce que *bateau* est masculin tandis que pour *main courante* c'est le féminin.

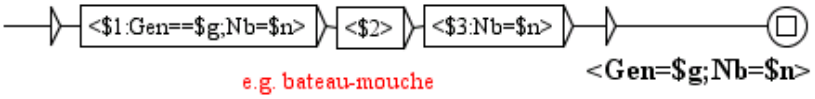


FIGURE 11.6 – Graphe de flexion *bateau-mouche* avec deux types d'instanciation

Quand une affectation par double symbole égale coexiste avec une affectation par simple symbole égale, sur le même chemin et pour la même variable, l'affectation par double symbole égale prévaut sur l'autre : la variable est instanciée une seule fois. Par exemple, sur la figure 11.6 la sortie finale contient *Gen=$g*, mais *$g* prend une seule valeur déterminée par le premier constituant.

Le système d'unification est particulièrement utile pour des langues à la flexion riche. Par exemple, en polonais la plupart des noms se fléchissent en nombre (2 valeurs) et en cas (7 valeurs), ce qui implique au moins 14 formes différentes (si des variantes et des formes syncrétiques diffèrent). Ce score est encore plus élevé pour les adjectifs qui se fléchissent en nombre, en cas et en genre (de 3 à 9 valeurs, selon différentes approches). Si aucun mécanisme d'unification n'était disponible, ces formes devraient être décrites par des chemins

séparés dans le graphe. L'unification permet de réduire considérablement la taille du graphe (jusqu'à un seul chemin dans la plupart des cas).

Par exemple, le graphe de la figure 11.7 permet de fléchir les mots composés polonais qui se fléchissent comme *pranie mózgu* (*lavage du cerveau*) ou *powożenie koniem* (ang. *horse coaching*). Leur troisième constituant a son cas défini (le plus souvent au génitif ou à l'instrumental). Le premier et le troisième constituant se fléchissent en nombre indépendamment l'un de l'autre (*pranie mózgów*, *prania mózgu*, *prania mózgów*, etc.). C'est pourquoi chacun d'eux a une variable différente pour la flexion en nombre (*$n1* et *$n2*). Les trois variables $\$n1$, $\$n2$, et $\$c$ peuvent être instanciées à n'importe quelle valeur de leur domaine respectif (*{sing,pl}*, *{sing,pl}*, et *{Nom,Gen,Dat,Acc,Inst,Loc,Voc}* ; cf. `Morphology.txt` fichier à la section 11.2.1). Le mot composé hérite son genre, son nombre et son cas de son premier constituant. Ce genre est défini par (*Gen==$g*) alors que son nombre et son cas sont instanciés selon 14 combinaisons possibles. Sans unification, le chemin unique de ce graphe aurait du être remplacé par 28 chemins différents.

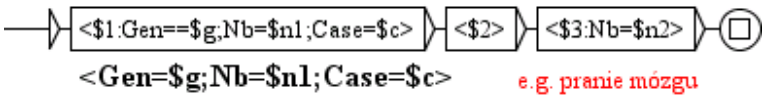


FIGURE 11.7 – Graphe de flexion pour *pranie mózgu*

Variantes orthographiques et autres variantes

Notre formalisme permet à n'importe quel constituant d'être omis ou déplacé au sein de différentes formes fléchies si cela est nécessaire. Il permet également l'insertion de constituants supplémentaires qui n'apparaissent pas dans la forme de base du mot composé. Cela permet d'étendre un paradigme flexionnel à une description de variantes plus générale, orthographique ou, partielle, variante syntaxique (voir [56] pour une étude exhaustive des variantes).

Par exemple en anglais, *student union* apparaît dans un corpus sous les formes *students union*, et *students' union*, au singulier ou au pluriel dans les deux cas. Notre formalisme permet d'ajouter les deux types de variantes à la description (cf. figure 11.8).

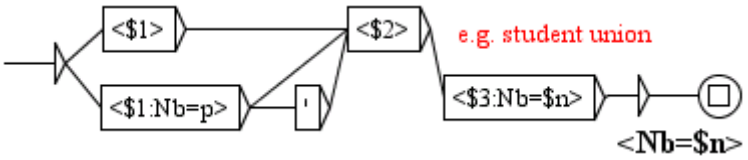


FIGURE 11.8 – Graphe de flexion pour *student union*

figure 11.9 montre un exemple dans lequel, en plus de l'insertion d'un nouveau constituant, l'ordre des constituants peut être inversé. Le chemin du haut permet de générer par exemple

birth date et *birth dates* tandis que celui du bas représente les variantes syntaxiques des formes précédentes : *date of birth* et *dates of birth*.

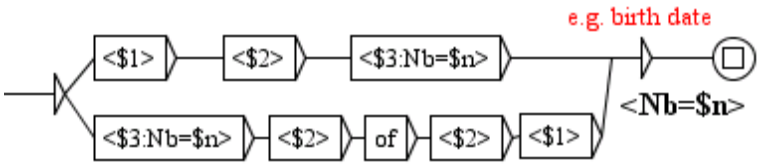


FIGURE 11.9 – Graphe de flexion pour *birth date*

Interface avec le système de flexion des mots simples

MULTIFLEX est une mise en œuvre du formalisme de flexion des mots composés précédemment présenté. Il suppose l'existence d'un sytème de flexion des mots simples qui satisfasse les contraintes d'interface suivantes :

- Pour une séquence de caractères donnée, il renvoie sa décomposition en constituants insécables (tokens) (cf section 11.2.2). Par exemple, dans le cas de la définiton d'un token dans Unitex, la séquence *Athens '04* est divisée en 5 tokens :

  $$\textit{“Athens '04”} \rightarrow \textit{(“Athens”,” “,””’,”0”,”4”)}$$

  Pour une forme fléchie simple donnée, il retourne toutes ses caractéristiques flexionnelles. Ces caractéristiques doivent permettre la génération à la demande de toute autre forme fléchie de même lemme par le même module de flexion. Par exemple, dans le cas d'Unitex, la forme *porte* mène à la reconnaissance de 7 formes (dont 6 sont factorisées selon leur code flexionnel) :

  $$\textit{porte} \rightarrow \textit{((porte,porte.N21 :s),(porte,porter.V3 :P1s :P3s :S1s :S3s :Y2s))}$$

  En cas d'ambiguïté, comme ci-dessus, l'identification correcte doit être faite, pour le moment, par l'utilisateur lors de l'édition du lemme du mot composé à fléchir (par la suite, cette tâche sera partiellement automatisée). Par exemple, dans le cas de *porte-fenêtre*, le premier constituant doit être identifié comme un nom plûtot que comme un verbe.

- Pour une identification morphologique donnée et un ensemble de valeurs flexionnelles, il renvoie toutes les formes fléchies corespondantes. Par exemple, en polonais, si le cas instrumental du mot *ręka* doit être produit, trois formes doivent être renvoyées : *ręką* (instrumental singulier), *rękami* et *rękoma* (deux variantes de l'instrumental pluriel).

  $$\begin{aligned}\textit{(ręka,<Case=Inst>)} \rightarrow\ & \textit{((ręką,<Nb=sing ;Gen=fem ;Case=Inst>),}\\ & \textit{(rękami,<Nb=pl ;Gen=fem ;Case=Inst>),}\\ & \textit{(rękoma,<Nb=pl ;Gen=fem ;Case=Inst>))}\end{aligned}$$

La présence d'une interface entre le système de flexion des mots simples et celui des mots composés permet une meilleure modularité et une indépendance de l'un vis-à-vis de l'autre. Le sytème de flexion des mots composés n'a pas besoin de savoir comment les formes fléchies des mots simples sont décrites, analysées et générées. Il a seulement besoin d'un ensemble de formes correctement fléchies des constituants des mots composés. Réciproquement, le système pour les mots simples ne connait rien de la manière dont celui des mots composés combine les formes fournies.

11.3 Intégration à Unitex

L'un des principes majeurs de conception de MULTIFLEX est d'être aussi indépendant que possible du système de flexion des mots simples. Cependant, l'existence d'un tel système est inévitable parce qu'un mot composé est formé de mots simples que nous devons être en mesure de fléchir dans le but de fléchir un mot composé dans son ensemble.

Dans sa version actuelle, MULTIFLEX repose sur le système de flexion des mots simples d'Unitex

- MULTIFLEX utilise les mêmes codages qu'Unitex, i.e. Unicode 3.0.
- MULTIFLEX utilise l'éditeur de graphe d'Unitex pour représenter la flexion des mots composés.
- MULTIFLEX admet des principes de description morphologique similaires à ceux du système DELA mis en œuvre dans Unitex. Ainsi, un paradigme est un ensemble d'actions à effectuer sur le lemme afin de générer ses formes fléchies, et de leurs associer les informations flexionnelles correspondantes.
- MULTIFLEX permet d'étendre la flexion des mots simples à celle des mots composés en produisant à partir d'un DELAC (DELA électronique des mots composés) un DELACF (DELA électronique des formes fléchies de mots composés). Le format du DELACF généré est compatible avec Unitex tandis que le format du DELAC est nouveau, mais inspiré de celui du DELAS (DELA électronique dictionnaire des mots simples).

Les sections suivantes présentent, pour plusieurs langues, des exemples complets de flexion d'un DELAC en DELACF à travers l'interface MULTIFLEX/Unitex.

11.3.1 Exemple complet en anglais

Supposons que la description des caractéristiques morphologiques de l'anglais est définie par le fichier `Morphology.txt` suivant :

English
<CATEGORIES>
Nb :s,p
<CLASSES>
noun :(Nb,<var>)
adj :

et que les équivalences entre les caractéristiques ci-dessus et leurs codes correspondants dans les dictionnaires DELA sont définis par le fichier `Equivalences.txt` suivant :

English
s : Nb=s
p : Nb=p

Considérons l'extrait du DELAC anglais suivant :

```
angle(angle.N1:s) of reflection,NC_NXXXX
Adam's apple(apple.N1:s),NC_XXXXN
air brake(brake.N1:s),NC_XXN
birth date(date.N1:s),NC_NN_NofN
criminal police,NC_XXXinv
cross-roads,NC_XXNs
head(head.N1:s) of government(government.N1:s),NC_NofNs
notary(notary.N3:s) public(public.N1:s),NC_NsNs
rolling stone(stone.N1:s),NC_XXN
student(student.N1:s) union(union.N1:s),NC_Ns'N
```

Les graphes de flexion correspondants *N1* et *N3* pour les mots simples se trouvent dans les figures 11.10 et figures 11.11 tandis que ceux pour les mots composés s'échelonnent de la figure 11.12 à la figure 11.20.

Le DELACF résultant de la flexion par MULTIFLEX du DELAC précédent est le suivant :

```
angle of reflection,angle of reflection.NC_NXXXX:s
angles of reflection,angle of reflection.NC_NXXXX:p
Adam's apple,Adam's apple.NC_XXXXN:s
Adam's apples,Adam's apple.NC_XXXXN:p
air brake,air brake.NC_XXN:s
air brakes,air brake.NC_XXN:p
date of birth,birth date.NC_NN_NofN:s
dates of birth,birth date.NC_NN_NofN:p
birth date,birth date.NC_NN_NofN:s
birth dates,birth date.NC_NN_NofN:p
criminal police,criminal police.NC_XXXinv:p
cross-roads,cross-roads.NC_XXNs:s
cross-roads,cross-roads.NC_XXNs:p
heads of government,head of government.NC_NofNs:p
heads of governments,head of government.NC_NofNs:p
head of government,head of government.NC_NofNs:s
notaries public,notary public.NC_NsNs:p
notary public,notary public.NC_NsNs:s
```

```
notary publics,notary public.NC_NsNs:p
rolling stone,rolling stone.NC_XXN:s
rolling stones,rolling stone.NC_XXN:p
students' union,student union.NC_Ns'N:s
students' unions,student union.NC_Ns'N:p
students union,student union.NC_Ns'N:s
students unions,student union.NC_Ns'N:p
student union,student union.NC_Ns'N:s
student unions,student union.NC_Ns'N:p
```

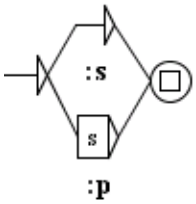


FIGURE 11.10 – Graphe de flexion *N1* de mots simples anglais

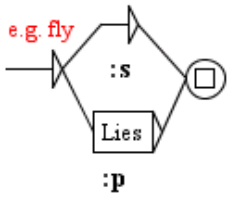


FIGURE 11.11 – Graphe de flexion *N3* de mots simples anglais

FIGURE 11.12 – Graphe de flexion *NC_NXXXX* de mots composés anglais

FIGURE 11.13 – Graphe de flexion *NC_XXXXN* de mots composés anglais

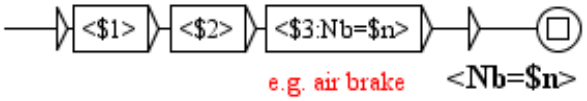


FIGURE 11.14 – Graphe de flexion *NC_XXN* de mots composés anglais

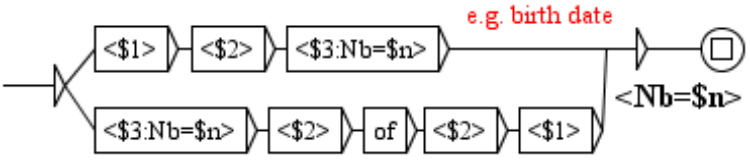


FIGURE 11.15 – Graphe de flexion *NC_NN_NofN* de mots composés anglais

FIGURE 11.16 – Graphe de flexion *NC_XXXinv* de mots composés anglais

FIGURE 11.17 – Graphe de flexion *NC_XXNs* de mots composés anglais

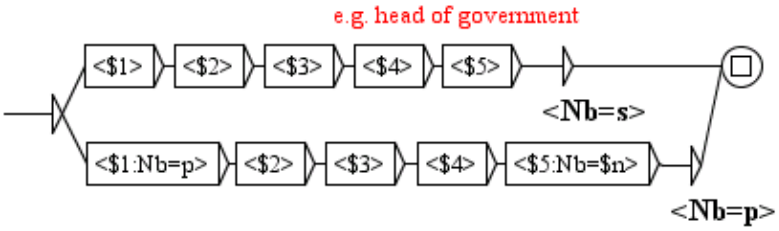


FIGURE 11.18 – Graphe de flexion *NC_NofNs* de mots composés anglais

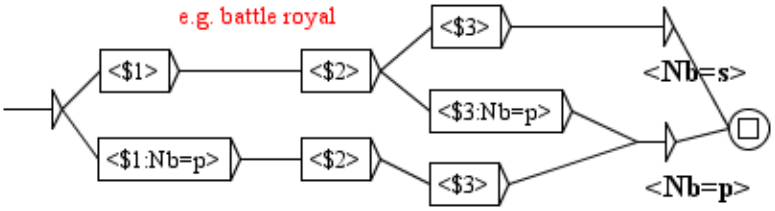


FIGURE 11.19 – Graphe de flexion *NC_NsNs* de mots composés anglais

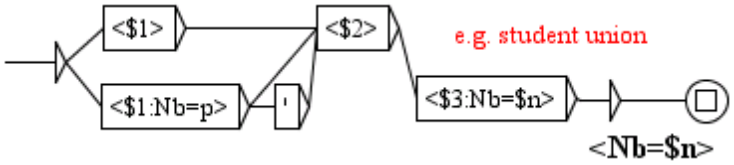


FIGURE 11.20 – Graphe de flexion *NC_Ns'N* de mots composés anglais

11.3.2 Exemple complet en français

Supposons que la description des caractéristiques morphologiques du français est définie par le fichier `Morphology.txt` suivant :

French
<CATEGORIES>
Nb : s, p
Gen : m, f
<CLASSES>
noun : (Nb,<var>),(Gen,<var>)
adj :(Nb,<var>),(Gen,<var>)
adv :

et que les équivalences entre les caractéristiques ci-dessus et leurs codes correspondants dans les dictionnaires DELA sont définis par le fichier `Equivalences.txt` suivant :

French
s : Nb=s
p : Nb=p
m : Gen=m
f : Gen=f

Considérons l'extrait du DELAC français suivant (les codes flexionnels des mots simples peuvent être différents de ceux présents dans Unitex) :

```
avant-garde(garde.N21:fs),NC_XXN
bateau(bateau.N3:ms)-mouche(mouche.N21:fs),NC_NN
café(café.N1:ms) au lait,NC_NXXXX
carte(carte.N21:fs) postale(postal.A8:fs),NC_NN$
cousin(cousin.N8:ms) germain(germain.A8:ms),NC_NNmf
franc(franc.A47:ms) maçon(maçon.N41:ms),NC_AN1
mémoire(mémoire.N21:fs) vive(vif.A48:fs),NC_NN
microscope(microscope.N1:ms) à effet tunnel,NC_NXXXXXX
porte-serviette(serviette.N21:fs),NC_VNm
```

Les graphes de flexion correspondants se trouvent de la figure 11.21 à la figure 11.27.

Le DELACF résultant de la flexion par MULTIFLEX du DELAC précédent est le suivant :

```
avant-garde,avant-garde.NC_XXN:fs
avant-gardes,avant-garde.NC_XXN:fp
bateau-mouche,bateau-mouche.NC_NN:ms
bateaux-mouches,bateau-mouche.NC_NN:mp
café au lait,café au lait.NC_NXXXX:ms
cafés au lait,café au lait.NC_NXXXX:mp
carte postale,carte postale.NC_NN:fs
cartes postales,carte postale.NC_NN:fp
cousin germain,cousin germain.NC_NNmf:ms
cousins germains,cousin germain.NC_NNmf:mp
cousine germaine,cousin germain.NC_NNmf:fs
cousines germaines,cousin germain.NC_NNmf:fp
franc-maçon,franc maçon.NC_AN1:ms
franc-maçonne,franc maçon.NC_AN1:fs
franc maçon,franc maçon.NC_AN1:ms
franc maçonne,franc maçon.NC_AN1:fs
francs-maçons,franc maçon.NC_AN1:mp
francs-maçonnes,franc maçon.NC_AN1:fp
francs maçons,franc maçon.NC_AN1:mp
francs maçonnes,franc maçon.NC_AN1:fp
mémoire vive,mémoire vive.NC_NN:fs
mémoires vives,mémoire vive.NC_NN:fp
microscope à effet tunnel,microscope à effet tunnel.NC_NXXXXXX:ms
microscopes à effet tunnel,microscope à effet tunnel.NC_NXXXXXX:mp
porte-serviette,porte-serviette.NC_VNm:ms
porte-serviettes,porte-serviette.NC_VNm:ms
porte-serviettes,porte-serviette.NC_VNm:mp
```

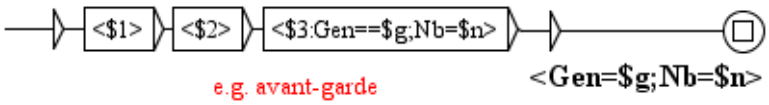


FIGURE 11.21 – Graphe de flexion *NC_XXN* de mots composés français

FIGURE 11.22 – Graphe de flexion *NC_NN* de mots composés français

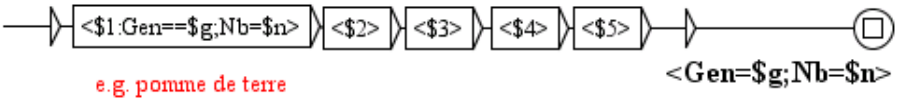


FIGURE 11.23 – Graphe de flexion *NC_NXXXX* de mots composés français

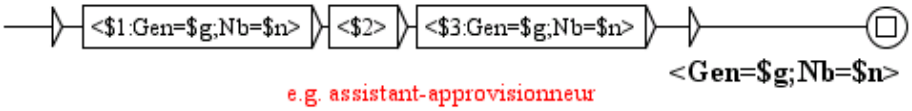


FIGURE 11.24 – Graphe de flexion *NC_NNmf* de mots composés français

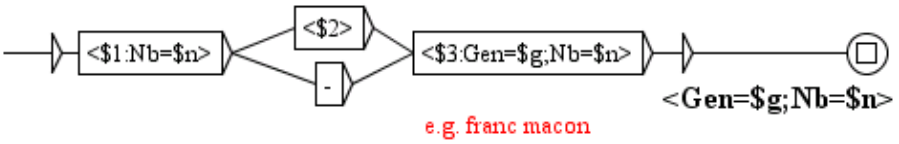


FIGURE 11.25 – Graphe de flexion *NC_AN1* de mots composés français

FIGURE 11.26 – Graphe de flexion *NC_NXXXXXX* de mots composés français

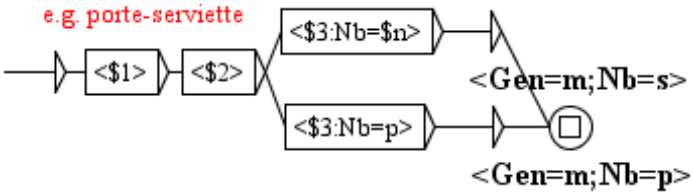


FIGURE 11.27 – Graphe de flexion *NC_VNm* de mots composés français

11.3.3 Exemple en serbe

Supposons que la description des caractéristiques morphologiques du serbe est définie par le fichier `Morphology.txt` suivant :

```
Serbian
<CATEGORIES>
Nb :s,p,w
Case :1,2,3,4,5,6,7
Gen :m,f,n
Anim :v,q,g
Comp :a,b,c
Det :d,k,e
<CLASSES>
noun :(Nb,<var>),(Case,<var>),(Gen,<var>),(Anim,<fixed>)
adj :(Nb,<var>),(Case,<var>),(Gen,<var>),(Anim,<var>),(Comp,<var>),(Det,<var>)
adv :
```

La particuliarité de ce modèle morphologique n'est pas seulement sa richesse mais aussi l'existence de *no-care* features comme *Anim=g* ou *Det=e*. Ces caractéristiques s'accordent avec les autres caractéristiques de la même catégorie. Elles sont utilisées uniquement pour certaines sous-classes particulières de noms ou d'adjectifs et sont nécessaires pour une meilleure compacité des paradigmes flexionnels des mots simples qui sont déjà très imposants, et le seraient encore plus sans elles.

Supposons que les équivalences entre les caractéristiques ci-dessus et leurs codes correspondants dans les dictionnaires DELA soient définis par le fichier `Equivalences.txt` suivant :

Serbian
s :Nb=s
p :Nb=p
w :Nb=w
1 :Case=1
2 :Case=2
3 :Case=3
4 :Case=4
5 :Case=5
6 :Case=6
7 :Case=7
m :Gen=m
f :Gen=f
n :Gen=n
v :Anim=v
q :Anim=q
g :Anim=g
a :Comp=a
b :Comp=b
c :Comp=c
d :Det=d
k :Det=k
e :Det=e

Considérons l'extrait du DELAC serbe suivant (les codes flexionnels des mots simples peuvent être différents de ceux présents dans Unitex) :

```
zxiro racyun(racyun.N1:ms1q),NC_2XN1+N+Comp
avio-prevoznik(prevoznik.N10:ms1v),NC_2XN2+N+Comp
predsednik(predsednik.N10:ms1v) drzxave(drzxava.N600:fs2q),NC_N2X1+N+Comp
Ujedinxene(Ujedinxen.A1:aefp1g) nacije(nacija.N600:fp1q),NC_AXN3+N+Comp+NProp+Org
Kosovo(Kosovo.N308:ns1q) i Metohija(Metohija.N623:fs1q),NC_N3XN+N+Comp+NProp+Top+Reg
istrazxni(istrazxni.A2:adms1g) sudija(sudija.N679:ms1v),NC_AXNF+N+Comp
Mirosinka(Mirosinka.N1637:fs1v) Dinkicx(Dinkicx.N1028:ms1v),NC_ImePrezime+N+Comp+Hum+PersName
gladan(gladan.A18:akms1g) kao vuk(vuk.N128:ms1v),AC_A3XN2/hungry as a wolf
```

Les graphes de flexion correspondants se trouvent de la figure 11.28 à la figure 11.35.

Le DELACF résultant de la flexion par MULTIFLEX du DELAC précédent est le suivant :

```
zxiro-racyun,zxiro racyun.NC_2XN1+N+Comp:s1qm
zxiro-racyuna,zxiro racyun.NC_2XN1+N+Comp:s2qm
zxiro-racyunu,zxiro racyun.NC_2XN1+N+Comp:s3qm
zxiro-racyun,zxiro racyun.NC_2XN1+N+Comp:s4qm
zxiro-racyune,zxiro racyun.NC_2XN1+N+Comp:s5qm
zxiro-racyunom,zxiro racyun.NC_2XN1+N+Comp:s6qm
zxiro-racyunu,zxiro racyun.NC_2XN1+N+Comp:s7qm
```

```
zxiro-racyuni,zxiro racyun.NC_2XN1+N+Comp:p1qm
zxiro-racyuna,zxiro racyun.NC_2XN1+N+Comp:p2qm
zxiro-racyunima,zxiro racyun.NC_2XN1+N+Comp:p3qm
zxiro-racyune,zxiro racyun.NC_2XN1+N+Comp:p4qm
zxiro-racyuni,zxiro racyun.NC_2XN1+N+Comp:p5qm
zxiro-racyunima,zxiro racyun.NC_2XN1+N+Comp:p6qm
zxiro-racyunima,zxiro racyun.NC_2XN1+N+Comp:p7qm
zxiro-racyuna,zxiro racyun.NC_2XN1+N+Comp:w2qm
zxiro-racyuna,zxiro racyun.NC_2XN1+N+Comp:w4qm
zxiro racyun,zxiro racyun.NC_2XN1+N+Comp:s1qm
zxiro racyuna,zxiro racyun.NC_2XN1+N+Comp:s2qm
zxiro racyunu,zxiro racyun.NC_2XN1+N+Comp:s3qm
zxiro racyun,zxiro racyun.NC_2XN1+N+Comp:s4qm
zxiro racyune,zxiro racyun.NC_2XN1+N+Comp:s5qm
zxiro racyunom,zxiro racyun.NC_2XN1+N+Comp:s6qm
zxiro racyunu,zxiro racyun.NC_2XN1+N+Comp:s7qm
zxiro racyuni,zxiro racyun.NC_2XN1+N+Comp:p1qm
zxiro racyuna,zxiro racyun.NC_2XN1+N+Comp:p2qm
zxiro racyunima,zxiro racyun.NC_2XN1+N+Comp:p3qm
zxiro racyune,zxiro racyun.NC_2XN1+N+Comp:p4qm
zxiro racyuni,zxiro racyun.NC_2XN1+N+Comp:p5qm
zxiro racyunima,zxiro racyun.NC_2XN1+N+Comp:p6qm
zxiro racyunima,zxiro racyun.NC_2XN1+N+Comp:p7qm
zxiro racyuna,zxiro racyun.NC_2XN1+N+Comp:w2qm
zxiro racyuna,zxiro racyun.NC_2XN1+N+Comp:w4qm
avio-prevoznik,avio-prevoznik.NC_2XN2+N+Comp:s1vm
avio-prevoznika,avio-prevoznik.NC_2XN2+N+Comp:s2vm
avio-prevozniku,avio-prevoznik.NC_2XN2+N+Comp:s3vm
avio-prevoznika,avio-prevoznik.NC_2XN2+N+Comp:s4vm
avio-prevoznicye,avio-prevoznik.NC_2XN2+N+Comp:s5vm
avio-prevoznikom,avio-prevoznik.NC_2XN2+N+Comp:s6vm
avio-prevozniku,avio-prevoznik.NC_2XN2+N+Comp:s7vm
avio-prevoznici,avio-prevoznik.NC_2XN2+N+Comp:p1vm
avio-prevoznika,avio-prevoznik.NC_2XN2+N+Comp:p2vm
avio-prevoznicima,avio-prevoznik.NC_2XN2+N+Comp:p3vm
avio-prevoznike,avio-prevoznik.NC_2XN2+N+Comp:p4vm
avio-prevoznici,avio-prevoznik.NC_2XN2+N+Comp:p5vm
avio-prevoznicima,avio-prevoznik.NC_2XN2+N+Comp:p6vm
avio-prevoznicima,avio-prevoznik.NC_2XN2+N+Comp:p7vm
avio-prevoznika,avio-prevoznik.NC_2XN2+N+Comp:w2vm
avio-prevoznika,avio-prevoznik.NC_2XN2+N+Comp:w4vm
avioprevoznik,avio-prevoznik.NC_2XN2+N+Comp:s1vm
avioprevoznika,avio-prevoznik.NC_2XN2+N+Comp:s2vm
avioprevozniku,avio-prevoznik.NC_2XN2+N+Comp:s3vm
avioprevoznika,avio-prevoznik.NC_2XN2+N+Comp:s4vm
avioprevoznicye,avio-prevoznik.NC_2XN2+N+Comp:s5vm
avioprevoznikom,avio-prevoznik.NC_2XN2+N+Comp:s6vm
avioprevozniku,avio-prevoznik.NC_2XN2+N+Comp:s7vm
avioprevoznici,avio-prevoznik.NC_2XN2+N+Comp:p1vm
avioprevoznika,avio-prevoznik.NC_2XN2+N+Comp:p2vm
avioprevoznicima,avio-prevoznik.NC_2XN2+N+Comp:p3vm
avioprevoznike,avio-prevoznik.NC_2XN2+N+Comp:p4vm
avioprevoznici,avio-prevoznik.NC_2XN2+N+Comp:p5vm
avioprevoznicima,avio-prevoznik.NC_2XN2+N+Comp:p6vm
avioprevoznicima,avio-prevoznik.NC_2XN2+N+Comp:p7vm
```

```
avioprevoznika,avio-prevoznik.NC_2XN2+N+Comp:w2vm
avioprevoznika,avio-prevoznik.NC_2XN2+N+Comp:w4vm
predsednik drzxave,predsednik drzxave.NC_N2X1+N+Comp:s1vm
predsednika drzxave,predsednik drzxave.NC_N2X1+N+Comp:s2vm
predsedniku drzxave,predsednik drzxave.NC_N2X1+N+Comp:s3vm
predsednika drzxave,predsednik drzxave.NC_N2X1+N+Comp:s4vm
predsednicye drzxave,predsednik drzxave.NC_N2X1+N+Comp:s5vm
predsednikom drzxave,predsednik drzxave.NC_N2X1+N+Comp:s6vm
predsedniku drzxave,predsednik drzxave.NC_N2X1+N+Comp:s7vm
predsednici drzxave,predsednik drzxave.NC_N2X1+N+Comp:p1vm
predsednici drzxava,predsednik drzxave.NC_N2X1+N+Comp:p1vm
predsednika drzxave,predsednik drzxave.NC_N2X1+N+Comp:p2vm
predsednika drzxava,predsednik drzxave.NC_N2X1+N+Comp:p2vm
predsednicima drzxave,predsednik drzxave.NC_N2X1+N+Comp:p3vm
predsednicima drzxava,predsednik drzxave.NC_N2X1+N+Comp:p3vm
predsednike drzxave,predsednik drzxave.NC_N2X1+N+Comp:p4vm
predsednike drzxava,predsednik drzxave.NC_N2X1+N+Comp:p4vm
predsednici drzxave,predsednik drzxave.NC_N2X1+N+Comp:p5vm
predsednici drzxava,predsednik drzxave.NC_N2X1+N+Comp:p5vm
predsednicima drzxave,predsednik drzxave.NC_N2X1+N+Comp:p6vm
predsednicima drzxava,predsednik drzxave.NC_N2X1+N+Comp:p6vm
predsednicima drzxave,predsednik drzxave.NC_N2X1+N+Comp:p7vm
predsednicima drzxava,predsednik drzxave.NC_N2X1+N+Comp:p7vm
predsednika drzxave,predsednik drzxave.NC_N2X1+N+Comp:w2vm
predsednika drzxava,predsednik drzxave.NC_N2X1+N+Comp:w2vm
predsednika drzxave,predsednik drzxave.NC_N2X1+N+Comp:w4vm
predsednika drzxava,predsednik drzxave.NC_N2X1+N+Comp:w4vm
Ujedinxene nacije,Ujedinxene nacije.NC_AXN3+N+Comp+NProp+Org:fp1q
Ujedinxenih nacija,Ujedinxene nacije.NC_AXN3+N+Comp+NProp+Org:fp2q
Ujedinxenima nacijama,Ujedinxene nacije.NC_AXN3+N+Comp+NProp+Org:fp3q
Ujedinxenim nacijama,Ujedinxene nacije.NC_AXN3+N+Comp+NProp+Org:fp3q
Ujedinxene nacije,Ujedinxene nacije.NC_AXN3+N+Comp+NProp+Org:fp4q
Ujedinxene nacije,Ujedinxene nacije.NC_AXN3+N+Comp+NProp+Org:fp5q
Ujedinxenima nacijama,Ujedinxene nacije.NC_AXN3+N+Comp+NProp+Org:fp6q
Ujedinxenim nacijama,Ujedinxene nacije.NC_AXN3+N+Comp+NProp+Org:fp6q
Ujedinxenima nacijama,Ujedinxene nacije.NC_AXN3+N+Comp+NProp+Org:fp7q
Ujedinxenim nacijama,Ujedinxene nacije.NC_AXN3+N+Comp+NProp+Org:fp7q
Kosovo i Metohija,Kosovo i Metohija.NC_N3XN+N+Comp+NProp+Top+Reg:ns1q
Kosova i Metohije,Kosovo i Metohija.NC_N3XN+N+Comp+NProp+Top+Reg:ns2q
Kosovu i Metohiji,Kosovo i Metohija.NC_N3XN+N+Comp+NProp+Top+Reg:ns3q
Kosovo i Metohiju,Kosovo i Metohija.NC_N3XN+N+Comp+NProp+Top+Reg:ns4q
Kosovo i Metohijo,Kosovo i Metohija.NC_N3XN+N+Comp+NProp+Top+Reg:ns5q
Kosovom i Metohijom,Kosovo i Metohija.NC_N3XN+N+Comp+NProp+Top+Reg:ns6q
Kosovu i Metohiji,Kosovo i Metohija.NC_N3XN+N+Comp+NProp+Top+Reg:ns7q
istrazxne sudije,istrazxni sudija.NC_AXNF+N+Comp:1vfp
istrazxnih sudija,istrazxni sudija.NC_AXNF+N+Comp:2vfp
istrazxnima sudijama,istrazxni sudija.NC_AXNF+N+Comp:3vfp
istrazxnim sudijama,istrazxni sudija.NC_AXNF+N+Comp:3vfp
istrazxne sudije,istrazxni sudija.NC_AXNF+N+Comp:4vfp
istrazxne sudije,istrazxni sudija.NC_AXNF+N+Comp:5vfp
istrazxnima sudijama,istrazxni sudija.NC_AXNF+N+Comp:6vfp
istrazxnim sudijama,istrazxni sudija.NC_AXNF+N+Comp:6vfp
istrazxnima sudijama,istrazxni sudija.NC_AXNF+N+Comp:7vfp
istrazxnim sudijama,istrazxni sudija.NC_AXNF+N+Comp:7vfp
istrazxne sudije,istrazxni sudija.NC_AXNF+N+Comp:2vfw
```

```
istrazxne sudije,istrazxni sudija.NC_AXNF+N+Comp:4vfw
istrazxnoga sudiju,istrazxni sudija.NC_AXNF+N+Comp:ms4v
istrazxnog sudiju,istrazxni sudija.NC_AXNF+N+Comp:ms4v
istrazxni sudija,istrazxni sudija.NC_AXNF+N+Comp:1vms
istrazxnoga sudije,istrazxni sudija.NC_AXNF+N+Comp:2vms
istrazxnog sudije,istrazxni sudija.NC_AXNF+N+Comp:2vms
istrazxnomu sudiji,istrazxni sudija.NC_AXNF+N+Comp:3vms
istrazxnome sudiji,istrazxni sudija.NC_AXNF+N+Comp:3vms
istrazxnom sudiji,istrazxni sudija.NC_AXNF+N+Comp:3vms
istrazxnomu sudiji,istrazxni sudija.NC_AXNF+N+Comp:7vms
istrazxnome sudiji,istrazxni sudija.NC_AXNF+N+Comp:7vms
istrazxnom sudiji,istrazxni sudija.NC_AXNF+N+Comp:7vms
istrazxni sudijo,istrazxni sudija.NC_AXNF+N+Comp:5vms
istrazxni sudija,istrazxni sudija.NC_AXNF+N+Comp:5vms
istrazxnim sudijom,istrazxni sudija.NC_AXNF+N+Comp:6vms
Dinkicx Mirosinka,Mirosinka Dinkicx.NC_ImePrezime+N+Comp+Hum+PersName:s1vf
Dinkicx Mirosinke,Mirosinka Dinkicx.NC_ImePrezime+N+Comp+Hum+PersName:s2vf
Dinkicx Mirosinki,Mirosinka Dinkicx.NC_ImePrezime+N+Comp+Hum+PersName:s3vf
Dinkicx Mirosinku,Mirosinka Dinkicx.NC_ImePrezime+N+Comp+Hum+PersName:s4vf
Dinkicx Mirosinka,Mirosinka Dinkicx.NC_ImePrezime+N+Comp+Hum+PersName:s5vf
Dinkicx Mirosinkom,Mirosinka Dinkicx.NC_ImePrezime+N+Comp+Hum+PersName:s6vf
Dinkicx Mirosinki,Mirosinka Dinkicx.NC_ImePrezime+N+Comp+Hum+PersName:s7vf
Mirosinka Dinkicx,Mirosinka Dinkicx.NC_ImePrezime+N+Comp+Hum+PersName:s1vf
Mirosinke Dinkicx,Mirosinka Dinkicx.NC_ImePrezime+N+Comp+Hum+PersName:s2vf
Mirosinki Dinkicx,Mirosinka Dinkicx.NC_ImePrezime+N+Comp+Hum+PersName:s3vf
Mirosinku Dinkicx,Mirosinka Dinkicx.NC_ImePrezime+N+Comp+Hum+PersName:s4vf
Mirosinka Dinkicx,Mirosinka Dinkicx.NC_ImePrezime+N+Comp+Hum+PersName:s5vf
Mirosinkom Dinkicx,Mirosinka Dinkicx.NC_ImePrezime+N+Comp+Hum+PersName:s6vf
Mirosinki Dinkicx,Mirosinka Dinkicx.NC_ImePrezime+N+Comp+Hum+PersName:s7vf
gladni kao vuk,gladan kao vuk.AC_A3XN2:s1mgda//hungry as a wolf
gladan kao vuk,gladan kao vuk.AC_A3XN2:s1mgka//hungry as a wolf
gladna kao vuk,gladan kao vuk.AC_A3XN2:s1fgea//hungry as a wolf
gladno kao vuk,gladan kao vuk.AC_A3XN2:s1ngea//hungry as a wolf
gladnoga kao vuk,gladan kao vuk.AC_A3XN2:s2mgda//hungry as a wolf
gladnog kao vuk,gladan kao vuk.AC_A3XN2:s2mgda//hungry as a wolf
gladna kao vuk,gladan kao vuk.AC_A3XN2:s2mgka//hungry as a wolf
gladne kao vuk,gladan kao vuk.AC_A3XN2:s2fgea//hungry as a wolf
gladnoga kao vuk,gladan kao vuk.AC_A3XN2:s2ngda//hungry as a wolf
gladnog kao vuk,gladan kao vuk.AC_A3XN2:s2ngda//hungry as a wolf
gladna kao vuk,gladan kao vuk.AC_A3XN2:s2ngka//hungry as a wolf
gladnome kao vuk,gladan kao vuk.AC_A3XN2:s3mgda//hungry as a wolf
gladnom kao vuk,gladan kao vuk.AC_A3XN2:s3mgda//hungry as a wolf
gladnu kao vuk,gladan kao vuk.AC_A3XN2:s3mgka//hungry as a wolf
gladnoj kao vuk,gladan kao vuk.AC_A3XN2:s3fgea//hungry as a wolf
gladnome kao vuk,gladan kao vuk.AC_A3XN2:s3ngda//hungry as a wolf
gladnom kao vuk,gladan kao vuk.AC_A3XN2:s3ngda//hungry as a wolf
gladnu kao vuk,gladan kao vuk.AC_A3XN2:s3ngka//hungry as a wolf
gladnu kao vuk,gladan kao vuk.AC_A3XN2:s4fgea//hungry as a wolf
gladno kao vuk,gladan kao vuk.AC_A3XN2:s4ngea//hungry as a wolf
gladni kao vuk,gladan kao vuk.AC_A3XN2:s5mgea//hungry as a wolf
gladna kao vuk,gladan kao vuk.AC_A3XN2:s5fgea//hungry as a wolf
gladno kao vuk,gladan kao vuk.AC_A3XN2:s5ngea//hungry as a wolf
gladnim kao vuk,gladan kao vuk.AC_A3XN2:s6mgea//hungry as a wolf
gladnom kao vuk,gladan kao vuk.AC_A3XN2:s6fgea//hungry as a wolf
gladnim kao vuk,gladan kao vuk.AC_A3XN2:s6ngea//hungry as a wolf
```

```
gladnome kao vuk,gladan kao vuk.AC_A3XN2:s7mgda//hungry as a wolf
gladnom kao vuk,gladan kao vuk.AC_A3XN2:s7mgda//hungry as a wolf
gladnu kao vuk,gladan kao vuk.AC_A3XN2:s7mgka//hungry as a wolf
gladnoj kao vuk,gladan kao vuk.AC_A3XN2:s7fgea//hungry as a wolf
gladnome kao vuk,gladan kao vuk.AC_A3XN2:s7ngda//hungry as a wolf
gladnom kao vuk,gladan kao vuk.AC_A3XN2:s7ngda//hungry as a wolf
gladnu kao vuk,gladan kao vuk.AC_A3XN2:s7ngka//hungry as a wolf
gladni kao vuk,gladan kao vuk.AC_A3XN2:p1mgea//hungry as a wolf
gladni kao vuci,gladan kao vuk.AC_A3XN2:p1mgea//hungry as a wolf
gladni kao vukovi,gladan kao vuk.AC_A3XN2:p1mgea//hungry as a wolf
gladne kao vuk,gladan kao vuk.AC_A3XN2:p1fgea//hungry as a wolf
gladne kao vuci,gladan kao vuk.AC_A3XN2:p1fgea//hungry as a wolf
gladne kao vukovi,gladan kao vuk.AC_A3XN2:p1fgea//hungry as a wolf
gladna kao vuk,gladan kao vuk.AC_A3XN2:p1ngea//hungry as a wolf
gladna kao vuci,gladan kao vuk.AC_A3XN2:p1ngea//hungry as a wolf
gladna kao vukovi,gladan kao vuk.AC_A3XN2:p1ngea//hungry as a wolf
gladnih kao vuk,gladan kao vuk.AC_A3XN2:p2mgea//hungry as a wolf
gladnih kao vuci,gladan kao vuk.AC_A3XN2:p2mgea//hungry as a wolf
gladnih kao vukovi,gladan kao vuk.AC_A3XN2:p2mgea//hungry as a wolf
gladnih kao vuk,gladan kao vuk.AC_A3XN2:p2fgea//hungry as a wolf
gladnih kao vuci,gladan kao vuk.AC_A3XN2:p2fgea//hungry as a wolf
gladnih kao vukovi,gladan kao vuk.AC_A3XN2:p2fgea//hungry as a wolf
gladnih kao vuk,gladan kao vuk.AC_A3XN2:p2ngea//hungry as a wolf
gladnih kao vuci,gladan kao vuk.AC_A3XN2:p2ngea//hungry as a wolf
gladnih kao vukovi,gladan kao vuk.AC_A3XN2:p2ngea//hungry as a wolf
gladnima kao vuk,gladan kao vuk.AC_A3XN2:p3mgea//hungry as a wolf
gladnima kao vuci,gladan kao vuk.AC_A3XN2:p3mgea//hungry as a wolf
gladnima kao vukovi,gladan kao vuk.AC_A3XN2:p3mgea//hungry as a wolf
gladnim kao vuk,gladan kao vuk.AC_A3XN2:p3mgea//hungry as a wolf
gladnim kao vuci,gladan kao vuk.AC_A3XN2:p3mgea//hungry as a wolf
gladnim kao vukovi,gladan kao vuk.AC_A3XN2:p3mgea//hungry as a wolf
gladnima kao vuk,gladan kao vuk.AC_A3XN2:p3fgea//hungry as a wolf
gladnima kao vuci,gladan kao vuk.AC_A3XN2:p3fgea//hungry as a wolf
gladnima kao vukovi,gladan kao vuk.AC_A3XN2:p3fgea//hungry as a wolf
gladnim kao vuk,gladan kao vuk.AC_A3XN2:p3fgea//hungry as a wolf
gladnim kao vuci,gladan kao vuk.AC_A3XN2:p3fgea//hungry as a wolf
gladnim kao vukovi,gladan kao vuk.AC_A3XN2:p3fgea//hungry as a wolf
gladnima kao vuk,gladan kao vuk.AC_A3XN2:p3ngea//hungry as a wolf
gladnima kao vuci,gladan kao vuk.AC_A3XN2:p3ngea//hungry as a wolf
gladnima kao vukovi,gladan kao vuk.AC_A3XN2:p3ngea//hungry as a wolf
gladnim kao vuk,gladan kao vuk.AC_A3XN2:p3ngea//hungry as a wolf
gladnim kao vuci,gladan kao vuk.AC_A3XN2:p3ngea//hungry as a wolf
gladnim kao vukovi,gladan kao vuk.AC_A3XN2:p3ngea//hungry as a wolf
gladne kao vuk,gladan kao vuk.AC_A3XN2:p4mgea//hungry as a wolf
gladne kao vuci,gladan kao vuk.AC_A3XN2:p4mgea//hungry as a wolf
gladne kao vukovi,gladan kao vuk.AC_A3XN2:p4mgea//hungry as a wolf
gladne kao vuk,gladan kao vuk.AC_A3XN2:p4fgea//hungry as a wolf
gladne kao vuci,gladan kao vuk.AC_A3XN2:p4fgea//hungry as a wolf
gladne kao vukovi,gladan kao vuk.AC_A3XN2:p4fgea//hungry as a wolf
gladna kao vuk,gladan kao vuk.AC_A3XN2:p4ngea//hungry as a wolf
gladna kao vuci,gladan kao vuk.AC_A3XN2:p4ngea//hungry as a wolf
gladna kao vukovi,gladan kao vuk.AC_A3XN2:p4ngea//hungry as a wolf
gladni kao vuk,gladan kao vuk.AC_A3XN2:p5mgea//hungry as a wolf
gladni kao vuci,gladan kao vuk.AC_A3XN2:p5mgea//hungry as a wolf
gladni kao vukovi,gladan kao vuk.AC_A3XN2:p5mgea//hungry as a wolf
```

```
gladne kao vuk,gladan kao vuk.AC_A3XN2:p5fgea//hungry as a wolf
gladne kao vuci,gladan kao vuk.AC_A3XN2:p5fgea//hungry as a wolf
gladne kao vukovi,gladan kao vuk.AC_A3XN2:p5fgea//hungry as a wolf
gladna kao vuk,gladan kao vuk.AC_A3XN2:p5ngea//hungry as a wolf
gladna kao vuci,gladan kao vuk.AC_A3XN2:p5ngea//hungry as a wolf
gladna kao vukovi,gladan kao vuk.AC_A3XN2:p5ngea//hungry as a wolf
gladnima kao vuk,gladan kao vuk.AC_A3XN2:p6mgea//hungry as a wolf
gladnima kao vuci,gladan kao vuk.AC_A3XN2:p6mgea//hungry as a wolf
gladnima kao vukovi,gladan kao vuk.AC_A3XN2:p6mgea//hungry as a wolf
gladnim kao vuk,gladan kao vuk.AC_A3XN2:p6mgea//hungry as a wolf
gladnim kao vuci,gladan kao vuk.AC_A3XN2:p6mgea//hungry as a wolf
gladnim kao vukovi,gladan kao vuk.AC_A3XN2:p6mgea//hungry as a wolf
gladnima kao vuk,gladan kao vuk.AC_A3XN2:p6fgea//hungry as a wolf
gladnima kao vuci,gladan kao vuk.AC_A3XN2:p6fgea//hungry as a wolf
gladnima kao vukovi,gladan kao vuk.AC_A3XN2:p6fgea//hungry as a wolf
gladnim kao vuk,gladan kao vuk.AC_A3XN2:p6fgea//hungry as a wolf
gladnim kao vuci,gladan kao vuk.AC_A3XN2:p6fgea//hungry as a wolf
gladnim kao vukovi,gladan kao vuk.AC_A3XN2:p6fgea//hungry as a wolf
gladnima kao vuk,gladan kao vuk.AC_A3XN2:p6ngea//hungry as a wolf
gladnima kao vuci,gladan kao vuk.AC_A3XN2:p6ngea//hungry as a wolf
gladnima kao vukovi,gladan kao vuk.AC_A3XN2:p6ngea//hungry as a wolf
gladnim kao vuk,gladan kao vuk.AC_A3XN2:p6ngea//hungry as a wolf
gladnim kao vuci,gladan kao vuk.AC_A3XN2:p6ngea//hungry as a wolf
gladnim kao vukovi,gladan kao vuk.AC_A3XN2:p6ngea//hungry as a wolf
gladnima kao vuk,gladan kao vuk.AC_A3XN2:p7mgea//hungry as a wolf
gladnima kao vuci,gladan kao vuk.AC_A3XN2:p7mgea//hungry as a wolf
gladnima kao vukovi,gladan kao vuk.AC_A3XN2:p7mgea//hungry as a wolf
gladnim kao vuk,gladan kao vuk.AC_A3XN2:p7mgea//hungry as a wolf
gladnim kao vuci,gladan kao vuk.AC_A3XN2:p7mgea//hungry as a wolf
gladnim kao vukovi,gladan kao vuk.AC_A3XN2:p7mgea//hungry as a wolf
gladnima kao vuk,gladan kao vuk.AC_A3XN2:p7fgea//hungry as a wolf
gladnima kao vuci,gladan kao vuk.AC_A3XN2:p7fgea//hungry as a wolf
gladnima kao vukovi,gladan kao vuk.AC_A3XN2:p7fgea//hungry as a wolf
gladnim kao vuk,gladan kao vuk.AC_A3XN2:p7fgea//hungry as a wolf
gladnim kao vuci,gladan kao vuk.AC_A3XN2:p7fgea//hungry as a wolf
gladnim kao vukovi,gladan kao vuk.AC_A3XN2:p7fgea//hungry as a wolf
gladnima kao vuk,gladan kao vuk.AC_A3XN2:p7ngea//hungry as a wolf
gladnima kao vuci,gladan kao vuk.AC_A3XN2:p7ngea//hungry as a wolf
gladnima kao vukovi,gladan kao vuk.AC_A3XN2:p7ngea//hungry as a wolf
gladnim kao vuk,gladan kao vuk.AC_A3XN2:p7ngea//hungry as a wolf
gladnim kao vuci,gladan kao vuk.AC_A3XN2:p7ngea//hungry as a wolf
gladnim kao vukovi,gladan kao vuk.AC_A3XN2:p7ngea//hungry as a wolf
gladna kao vuk,gladan kao vuk.AC_A3XN2:w2mgea//hungry as a wolf
gladna kao vuci,gladan kao vuk.AC_A3XN2:w2mgea//hungry as a wolf
gladna kao vukovi,gladan kao vuk.AC_A3XN2:w2mgea//hungry as a wolf
gladne kao vuk,gladan kao vuk.AC_A3XN2:w2fgea//hungry as a wolf
gladne kao vuci,gladan kao vuk.AC_A3XN2:w2fgea//hungry as a wolf
gladne kao vukovi,gladan kao vuk.AC_A3XN2:w2fgea//hungry as a wolf
gladna kao vuk,gladan kao vuk.AC_A3XN2:w2ngea//hungry as a wolf
gladna kao vuci,gladan kao vuk.AC_A3XN2:w2ngea//hungry as a wolf
gladna kao vukovi,gladan kao vuk.AC_A3XN2:w2ngea//hungry as a wolf
gladna kao vuk,gladan kao vuk.AC_A3XN2:w4mgea//hungry as a wolf
gladna kao vuci,gladan kao vuk.AC_A3XN2:w4mgea//hungry as a wolf
gladna kao vukovi,gladan kao vuk.AC_A3XN2:w4mgea//hungry as a wolf
gladne kao vuk,gladan kao vuk.AC_A3XN2:w4fgea//hungry as a wolf
```

```
gladne kao vuci,gladan kao vuk.AC_A3XN2:w4fgea//hungry as a wolf
gladne kao vukovi,gladan kao vuk.AC_A3XN2:w4fgea//hungry as a wolf
gladna kao vuk,gladan kao vuk.AC_A3XN2:w4ngea//hungry as a wolf
gladna kao vuci,gladan kao vuk.AC_A3XN2:w4ngea//hungry as a wolf
gladna kao vukovi,gladan kao vuk.AC_A3XN2:w4ngea//hungry as a wolf
```

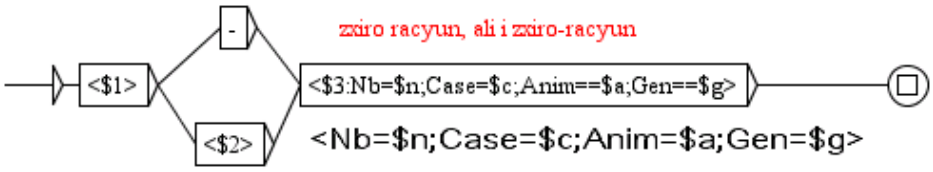


FIGURE 11.28 – Graphe de flexion *NC_2XN1* de mots composés serbes

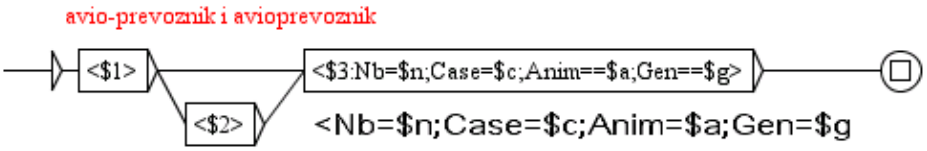


FIGURE 11.29 – Graphe de flexion *NC_2XN2* de mots composés serbes

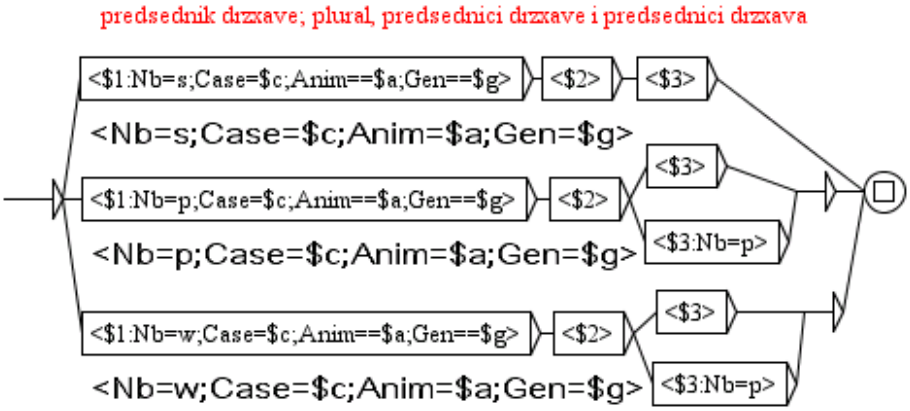


FIGURE 11.30 – Graphe de flexion *NC_N2X1* de mots composés serbes

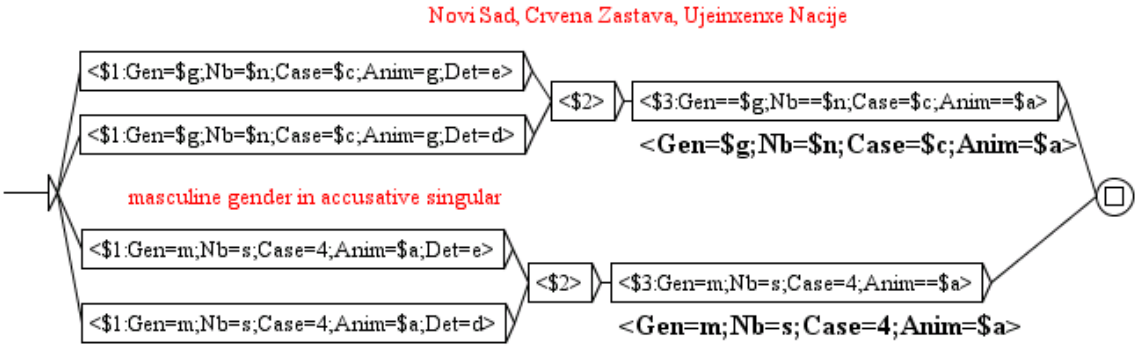


FIGURE 11.31 – Graphe de flexion *NC_AXN3* de mots composés serbes

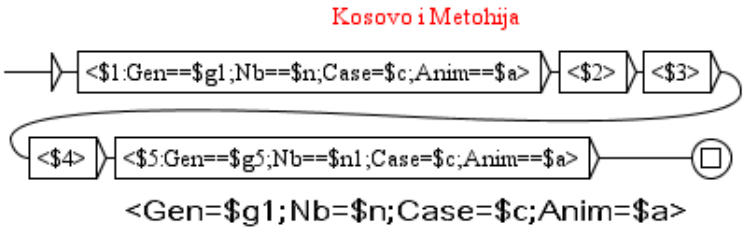


FIGURE 11.32 – Graphe de flexion *NC_N3XN* de mots composés serbes

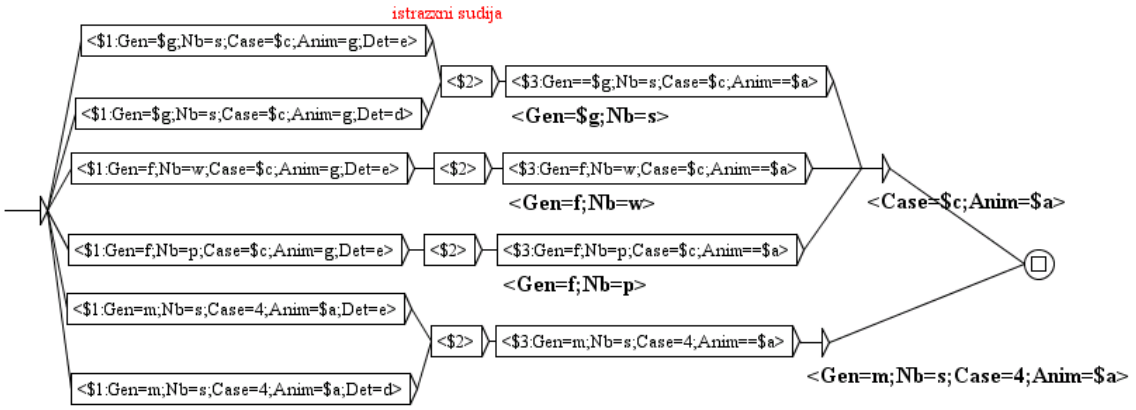


FIGURE 11.33 – Graphe de flexion *NC_AXNF* de mots composés serbes

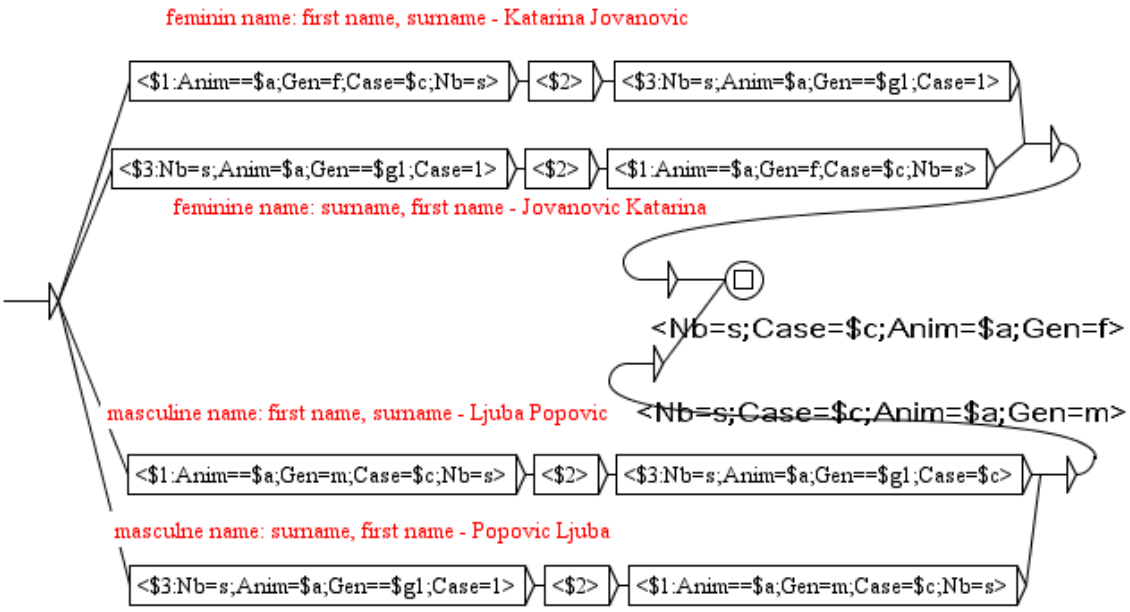


FIGURE 11.34 – Graphe de flexion *NC_ImePrezime* de mots composés serbes

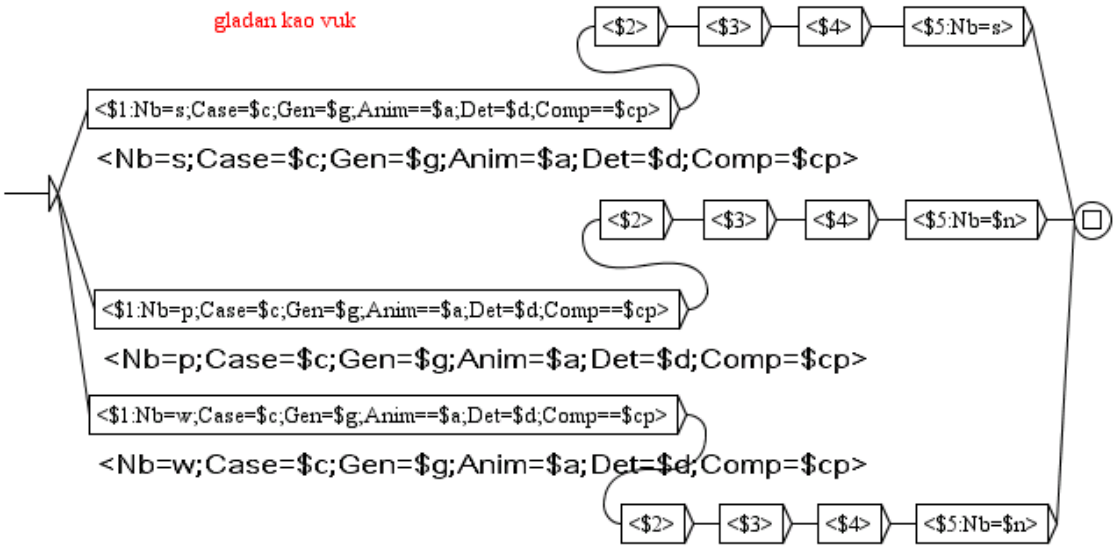


FIGURE 11.35 – Graphe de flexion *AC_A3XN2* de mots composés serbes

Chapitre 12

Cascades de Transducteurs

Ce chapitre présente l'outil *CasSys* qui donne la possibilité de créer une cascade de transducteurs et de nouvelles manières de travailler sur la langue naturelle avec des graphes à états finis. Une *cascade de transducteurs* applique plusieurs graphes (automates ou transducteurs), l'un après l'autre, sur le texte : chaque graphe modifie le texte, et les changements peuvent être utilisés pour des traitements supplémentaires par les graphes suivants. Ce type de système est notamment utilisé pour l'analyse syntaxique, le parenthésage (chunking), l'extraction d'information, la reconnaissance d'entités nommées, etc. Pour faire cela, CasSys utilise une succession de "locate pattern" avec les options adéquates.

Le premier prototype du système *CasSys* a été créé en 2002 au laboratoire LI (Laboratoire d'informatique de l'université de Tours) ([31]). Ce prototype était entièrement spécialisé pour l'extraction d'entités nommées. CasSys a été ensuite généralisé pour effectuer n'importe quelle sorte de traitement nécessitant une cascade. Il a été constamment amélioré au cours des années, puis intégré à Unitex, dans le cadre d'un projet[1].

Les grammaires Unitex sont de type Context free et intègrent la notion de transduction issue du domaine des automates à états finis. Une grammaire avec transduction (un transducteur) est capable de produire une sortie. CasSys est spécialisé dans l'application de transducteurs sous la forme d'une cascade.

Les transducteurs sont intéressants car ils permettent d'associer à la séquence reconnue l'information qui se trouve dans les sorties des graphes. Ces sorties peuvent :

- Être ajoutées à la séquence reconnue et apparaître dans la concordance résultante ou le texte modifié.
- Remplacer la séquence reconnue pour modifier le texte.

Ces deux opérations transforment le texte ou lui ajoute des informations.

Dans ce chapitre, nous expliquons comment créer des cascades de transducteurs et comment les appliquer. Ensuite, nous détaillons les options et possibilités offertes par CasSys.

1. "Feder-Région Centre entités nommées et nommables" dirigé par Denis Maurel, LI, Tours, France, intégration réalisée par Nathalie Friburger et David Nott

12.1 Appliquer une cascade de Transducteurs avec CasSys

Appliquer une cascade de transducteurs avec CasSys consiste à représenter un phénomène linguistique par une liste de transducteurs à appliquer au texte dans un ordre précis : CasSys et son interface dans Unitex permettent d'y parvenir. Cette section explique comment utiliser l'interface pour créer et gérer les graphes (ordre, ajout, suppression) et appliquer la cascade.

12.1.1 Création de la liste des transducteurs

Afin de pouvoir gérer la liste de transducteurs, le menu FSGraph comporte deux sous-menus : "*New cascade*" et "*Edit cascade...*" (Figure 12.1). Pour créer la liste des transducteurs, sélectionnez "*New cascade*". Si vous souhaitez modifier une cascade existante, sélectionnez "*Edit cascade...*", puis choisissez le nom de la cascade à ouvrir.

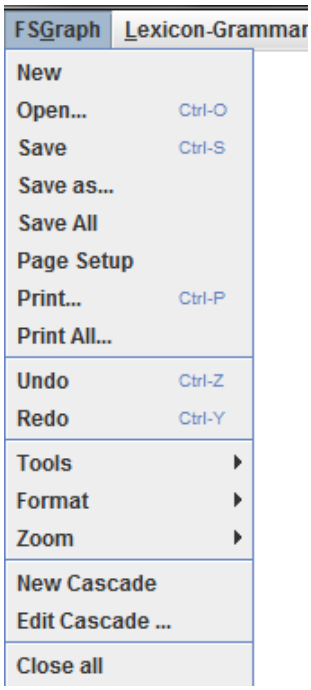


FIGURE 12.1 – Menu "FSGraph" d'Unitex et sous-menu "New Cascade" et "*Edit cascade...*"

Le répertoire de la langue courante contient un sous-répertoire nommé CasSys dans lequel se trouvent les fichiers de configuration d'une cascade. Ce sont des fichiers textes avec l'extension *.csc* (ex : maCascade.csc)

12.1.2 Édition de la liste des transducteurs

La fenêtre de configuration de CasSys (Figure 12.2) comporte trois parties :

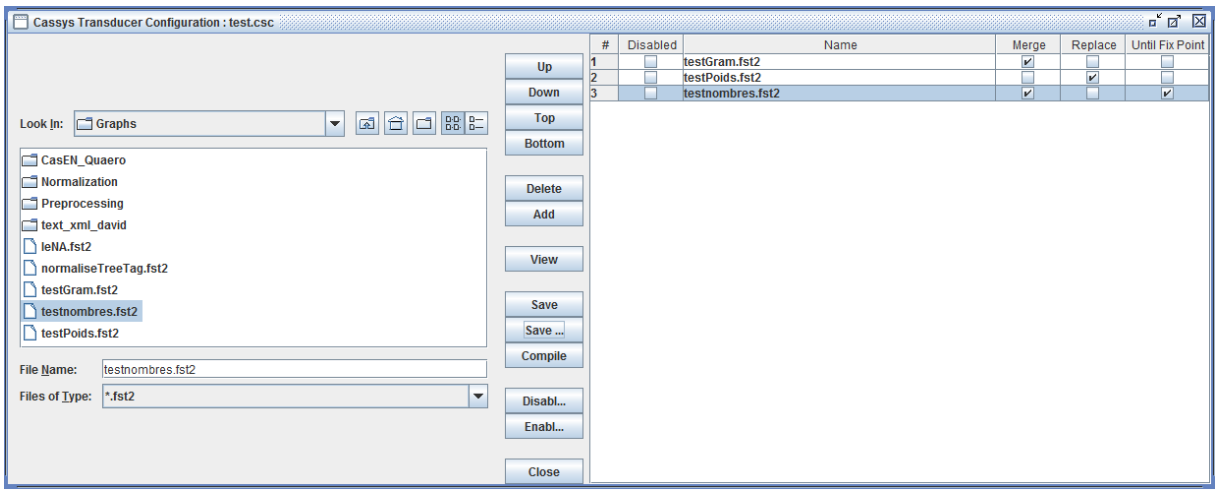


FIGURE 12.2 – Fenêtre de configuration de CasSys avec à droite la liste des transducteurs

1. Un *gestionnaire de fichier* à gauche du cadre permet de choisir les transducteurs à mettre dans la cascade. Le gestionnaire n'affiche que les fichiers *fst2* (tous les graphes que vous souhaitez mettre dans la liste doivent être compilés au format *fst2*).

   Pour éditer la cascade, choisissez les graphes à gauche et mettez-les à droite à l'aide d'un glisser-déposer.

2. Le *tableau* de droite affiche la cascade : la liste ordonnée des transducteurs et les options sélectionnées pour chaque graphe. Le tableau est évidemment vide pour une nouvelle cascade.

   Les colonnes du tableau (Figure 12.3) donne le numéro de chaque graphe et permettent de choisir leur comportement.

   - **#** : Numéro du graphe/transducteur dans la cascade pour chaque graphe ; le fichier *fst2* est numéroté.
   - **Disabled** : Pour désélectionner le graphe courant. *Disabled* siginifie : "*non appliqué dans la cascade*". Les graphes non sélectionnés apparaissent sans numéro, en grisé et barré.
   - **Name** : Le nom du graphe (avec l'extension *fst2*). Si vous laissez la souris sur le nom du graphe, une info-bulle apparaît avec le chemin complet du graphe (depuis le répertoire courant *Graphs*). Les graphes dont le fichier source n'est pas trouvé apparaissent en italique et en rouge.
   - **Merge** : Si le transducteur doit être appliqué en mode *merge*.
   - **Replace** : Si le transducteur doit être appliqué en mode *replace*.
   - **Until fix point** : Si le transducteur doit être appliqué une ou plusieurs fois jusqu'à ce que le texte reste inchangé, c'est-à-dire jusqu'à ce qu'un point fixe soit atteint (voir 12.2.2).

3. Au centre se trouvent les boutons décrits ci-dessous :

   - *"Up"/"Down"/"Top"/"Bottom"* sont utilisés pour modifier l'ordre des transducteurs dans la liste (ils déplacent le transducteur sélectionné) ; *"Up"* et *"Down"* déplacent le transducteur sélectionné d'une ligne vers le haut ou vers le bas, *"Top"* et *"Bottom"* le positonnent au début ou à la fin de la liste.
   - *"Delete"* permet de supprimer le transducteur sélectionné de la liste des transducteurs.
   - *"Add"* ajoute un transducteur (précédemment sélectionné) dans la liste. Il remplace le gisser-déposer préalablement décrit.
   - *"View"* ouvre le graphe sélectionné aussi bien dans l'explorateur de fichiers que dans la liste de transducteurs. Il est très utile d'avoir un accès rapide à n'importe quel transducteur aussi bien pour y jeter un coup d'œil que pour le modifier.
   - *"Save"* et *"Save as"* permettent d'enregistrer la liste des transducteurs. Par défaut, les listes des transducteurs sont placées dans le répertoire CasSys de la langue courante (par exemple French/CasSys).
   - *"Compile"* recompile tous les graphes de la cascade.
   - *"Disable all"* pour désélectionner tous les graphes de la cascade.
   - *"Enable all"* pour sélectionner tous les graphes de la cascade.
   - *"Close"* ferme la fenêtre courante.

#	Disabled	Name	Merge	Replace	Iter
1	☐	toolFigement.fst2	☐	☑	☐
2	☐	persNoel.fst2	☑	☐	☐
3	☐	amount.fst2	☑	☐	☐
4	☐	amountAmount.fst2	☑	☐	☐
5	☐	timeDateCalendaireAvecFin.fst2	☑	☐	☐
6	☐	timeDateCalendaire.fst2	☑	☐	☐
7	☐	timeAnneeSiecle.fst2	☑	☐	☐
8	☐	timeDateRelative.fst2	☑	☐	☐
9	☐	timeDateAbsolue.fst2	☑	☐	☐
	☑	~~timePeriode.fst2~~	☑	☐	☐
10	☐	timePrep.fst2	☑	☐	☐
11	☐	amountPrepDuree.fst2	☑	☐	☐
12	☐	amountDureeLesHour.fst2	☑	☐	☐
	☑	~~timeHoraire.fst2~~	☑	☐	☐
13	☐	timeLocution.fst2	☑	☐	☐
14	☐	time.fst2	☑	☐	☐
15	☐	timeDet0.fst2	☑	☐	☐
16	☐	adhocEtapeTimeMois.fst2	☑	☐	☐
17	☐	adhocEtapeTime.fst2	☑	☐	☐
18	☐	persCollectif.fst2	☑	☐	☐
19	☐	foncCollectiveExtractor.fst2	☑	☐	☐
20	☐	foncCollective.fst2	☑	☐	☐
21	☐	foncCollectiveCtxtD.fst2	☑	☐	☐
22	☐	orgInstitution.fst2	☑	☐	☐
23	☐	orgCtxtDico.fst2	☑	☐	☐
24	☐	orgCtxt.fst2	☑	☐	☐
25	☐	orgCommerceDroite.fst2	☑	☐	☐

FIGURE 12.3 – La table/liste de transducteurs

12.1.3 Application d'une cascade

Dans le menu "Text", sélectionner le sous-menu "*Apply CasSys cascade...*" (Figure 12.4) pour ouvrir la fenêtre CasSys. Ce sous-menu "*Apply CasSys cascade...*" n'est actif que si un texte a été préalablement ouvert.

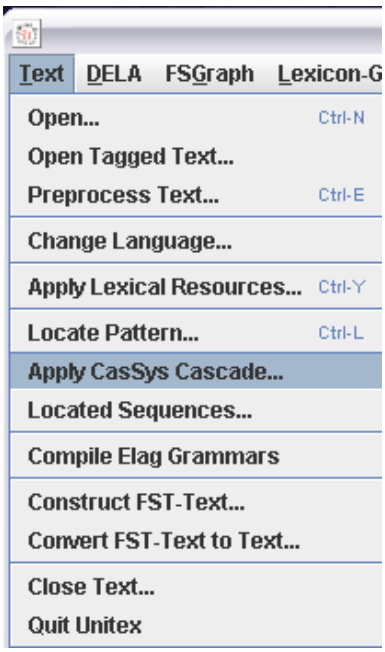


FIGURE 12.4 – Menu "Text" d'Unitex et sous-menu "Apply CasSys Cascade"

La fenêtre CasSys (Figure 12.5) affiche le contenu du répertoire CasSys de la langue courante. Elle permet de choisir le fichier contenant la liste de transducteurs à appliquer au texte. Une fois que cette liste est choisie, vous pouvez cliquer sur le bouton "Launch" pour appliquer la cascade. Attention, pour que l'affichage se fasse ici, il faut obligatoirement que le nom du fichier contenant la liste de transducteurs ait pour extension `.csc`.

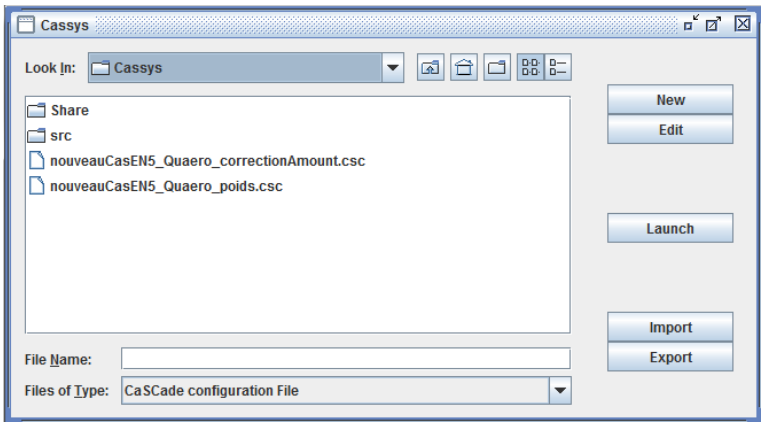


FIGURE 12.5 – Fenêtre de lancement de la cascade de transducteurs

N'importe quel dictionnaire du mode morphologique déclaré dans vos préférences est utlisable dans vos graphes. Les préférences peuvent être modifiées à partir du menu "Info" (Info –> Preferences –> morphological-mode dictionaries).

12.2 CasSys en détail

Dans cette section nous présentons une description détaillée du fonctionnement de CasSys.

12.2.1 Type de graphe utilisé

CasSys utilise la version compilée des graphes (format `.fst2`). CasSys gère les grammaires locales (section 6.1.4) présentées dans le chapitre 6. Les grammaires utilisées dans une cascade suivent les mêmes règles que les grammaires habituellement utilisées dans Unitex. Elles peuvent comporter des sous-graphes, utiliser le mode morphologique et les filtres morphologiques, et faire référence aux informations présentes dans les dictionnaires.

CasSys n'est pas compatible avec les fichiers `fst2` en mode debug (section 6.10.7). Quand on applique un graphe en mode debug avec le menu `Text>Locate Pattern`, le système compile le graphe dans un format spécial de mode debug. Pour obtenir un fichier au format `fst2` normal, recompilez le graphe, soit avec le menu `FSGraph`, soit en ligne de commande, soit en décochant le mode debug avant d'appliquer le graphe avec `Locate Pattern`.

12.2.2 Application itérative

CasSys peut appliquer un graphe sur un texte de manière itérative tant que de nouvelles concordances sont obtenues. Ce comportement est sélectionné ou non pour chaque graphe selon que la case `Until fix point` est cochée ou non.

Considérons par exemple le graphe 12.6 qui reconnait *AB* et le remplace par *A*.

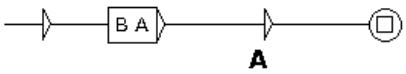


FIGURE 12.6 – Transducteur qui modifie BA en A

Considérons le texte *B B B A A A*. L'application du graphe 12.6 sur ce texte avec *Until fix point* donne :

initial text	B	B	B	A	A	A	
itération 1		B	B	A	A	A	1 match
itération 2			B	A	A	A	1 match
itération 3				A	A	A	1 match
itération 4				A	A	A	0 match

Durant les trois premières itérations, une concordance est obtenue, le graphe est alors appliqué à nouveau au texte résultant. A la quatrième itération, aucune concordance n'est trouvée, le graphe n'est donc plus réappliqué.

Attention : Prendre garde à la possibilité de blocage en utilisant cette option. Par exemple, un transducteur qui reconnaît *A* et le remplace par *A* causerait un blocage s'il était appliqué sur le texte de l'exemple.

12.2.3 Règles utilisées dans une cascade

Dans une cascade, chaque graphe observe les règles utilisées dans Unitex :

- Insertion à gauche des motifs reconnus : en mode "merge", la sortie est insérée à gauche de la séquence reconnue.
- Priorité au motif le plus à gauche : lors de l'application d'une grammaire locale, les occurrences qui se chevauchent sont toutes indexées. Durant la construction de la concordance, toutes ces occurrences sont présentes, mais comme CasSys modifie le texte après application de chaque graphe de la cascade, il est nécessaire de choisir parmi ces occurrences celle à prendre en compte. La priorité est donnée à la séquence la plus à gauche.
- Priorité au plus long motif : dans CasSys, lors de l'application d'un graphe, c'est la séquence la plus longue qui est conservée.
- Limitation du nombre d'occurrences recherchées : dans CasSys, ce nombre n'est pas limité : une telle limitation n'a aucun sens dans CasSys. Toutes les occurrences sont toujours indexées dans le texte.

12.2.4 Marquage de motifs dans CasSys

La sortie des transducteurs peut être utilisée pour insérer des informations dans les textes, en particulier pour marquer les motifs reconnus : il est possible d'utiliser toute sorte de marques, ( ), [], "", etc. ou des balises xml comme <xxx> </xxx>, mais CasSys propose une manière particulière d'annoter les motifs reconnus, offrant certaines possibilités que nous présentons maintenant.

Unitex découpe les textes en tokens de différentes sortes comme le marqueur de fin de phrase {S} ; le marqueur {STOP} ; des séquences de lettres contiguës ; des étiquettes lexicales {aujourd'hui,.ADV}, etc. Les étiquettes lexicales sont utilisées dans CasSys de manière particulière. Une étiquette lexicale (entre accolades) est habituellement utilisée pour éviter les ambiguïtés (voir les explications dans les sections 2.5.4 et 7.5.1). Par exemple, dans un texte, si vous avez le token *{curly brackets,.N}*, ni "curly" ni "brackets" ne seront reconnus, mais seulement la séquence toute entière "curly brackets".

Une étiquette lexicale peut contenir une suite complexe d'informations lexicales, comme *N+Pers+Hum :fs*. Dans un graphe, il est possible de chercher un token en utilisant l'information contenue dans un masque lexical : par exemple, on peut écrire <.*N*> pour chercher un nom, <.*Pers+Hum*> pour un être humain ou <.*Pers*>. Ces masques lexicaux sont décrits dans le chapitre "Recherche d'expressions rationnelles" (section 4.3.1).

Une cascade de transducteurs est intéressante pour localiser un îlot de certitude. Il est nécessaire pour ce type de système d'éviter que des motifs précédemment reconnus soient ambigus avec ceux reconnus par les graphes suivants. Pour éviter cela, on étiquette les motifs reconnus par les graphes sous la forme *{* et *,.tag1+tag2+tagn}* (où *tag1, tag2, etc.* sont vos propres étiquettes).

Pour expliciter ce comportement, voici un exemple très simple. Le texte sur lequel nous travaillons est :

bac a b c cc a b b ba ab a b bca a b c abaabc.

Le graphe grfAB (Figure 12.7) reconnaît la séquence *ab* dans le texte et lui ajoute l'étiquette lexicale {a b,.AB}. Ce graphe appliqué en mode MERGE ajoute *{* et *,.AB}* dans le texte.

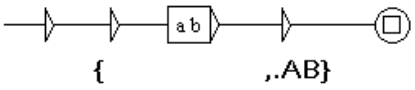


FIGURE 12.7 – Le graphe grfAB

Le texte résultant est : *bac {a b,.AB} c cc {a b,.AB} b ba ab {a b,.AB} bca {a b,.AB} c abaabc*.

Maintenant le motif *a b* est étiqueté *AB*. La partie (a ou b seul) de ce motif ne peut pas l'être à cause de l'étiquetage de *a b*.

Après ce graphe, la cascade applique un autre graphe nommé "tagAB" (Figure 12.8) contenant le masque lexical <AB>. Il reconnait toutes les séquences lexicalement étiquetées par le graphe précédent.

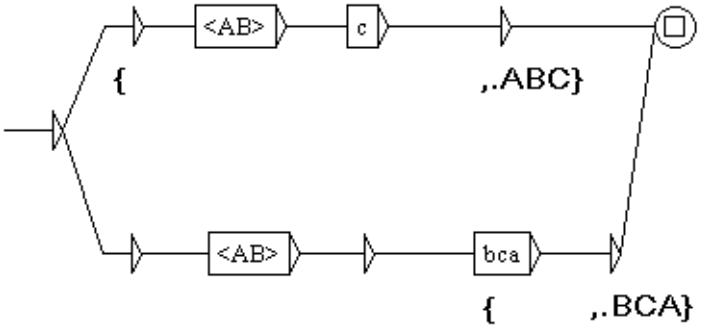


FIGURE 12.8 – Le graphe tagAB

Le texte résultant est : *bac {{a b,.AB} c,.ABC} cc {a b,.AB} b ba ab {a b,.AB} {bca,.BCA} {{a b,.AB} c,.ABC} abaabc*.

La concordance affichée par Unitex devrait ressembler à celle de la figure 12.9. Pour des raisons techniques (les ambiguïtés entre les caractères entre accolades des étiquettes lexicales), nous n'avons d'autre option que de placer des \ avant chaque caractère ambigu ; c'est pourquoi ces symboles sont précédés de \ dans la concordance.

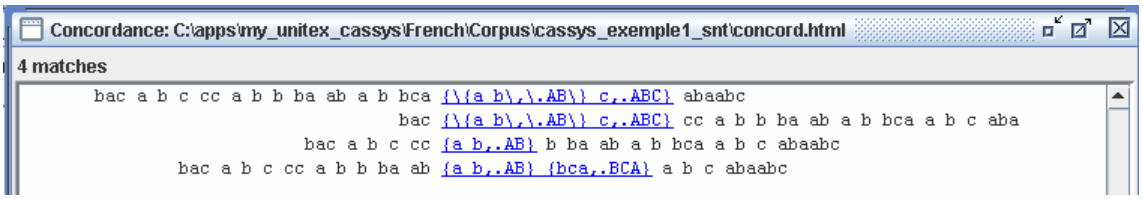


FIGURE 12.9 – La concordance issue l'application de la cascade

12.3 Graphes de généralisation d'étiquetage

Parfois, on a détecté des éléments grâce à des contextes déclencheurs, mais on ne les reconnaît pas s'ils apparaissent dans un autre contexte. Afin de trouver de telles occurrences, CasSys propose d'utiliser des graphes de généralisation d'étiquetage. Ces graphes contiennent des boîtes vides que le programme remplit automatiquement en extrayant de la liste de tokens du texte (tokens.txt) les formes ayant déjà été étiquetées d'une certaine façon. En appliquant au texte les graphes obtenus, on généralise cet étiquetage aux autres occurrences de ces formes.

12.3.1 Déclaration

Pour que CasSys reconnaisse un graphe de généralisation d'étiquetage, il faut cocher la colonne *Generalization* (figure 12.10).

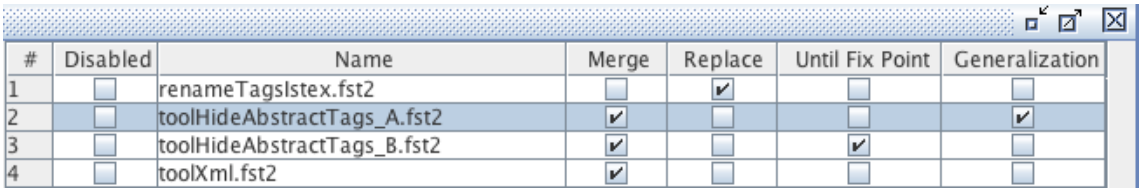


FIGURE 12.10 – Graphe de généralisation d'étiquetage

12.3.2 Graphes simples

Pour créer un graphe de généralisation d'étiquetage, on marque le début de la partie généralisante par une boîte $G avec en sortie une accolade ouvrante, et la fin par une boîte vide avec en sortie une accolade fermante éventuellement précédée d'un ou plusieurs traits. Entre les deux boites, une (et une seule) boite vide doit avoir en sortie l'étiquette ciblée,

FIGURE 12.11 – Bouton $G

précédée de ",.". On peut utiliser le bouton $G (figure 12.11) pour insérer automatiquement les boîtes de début et de fin de la partie généralisante après avoir sélectionné une boîte.

Par exemple, le graphe de la figure 12.12 est conçu pour généraliser l'étiquetage *x*.

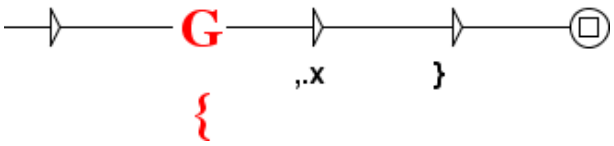


FIGURE 12.12 – Graphe de généralisation d'étiquetage simple

Si le fichier de tokens contient l'étiquette lexicale *{A,.x}*, le traitement va générer le graphe en figure 12.13.

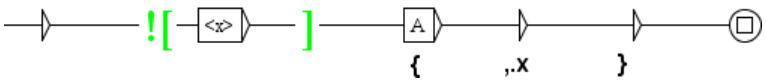


FIGURE 12.13 – Graphe de généralisation d'étiquetage simple, après traitement

Ce graphe va attribuer l'étiquette *x* aux occurrences de *A* qui ne l'ont pas encore reçue. Le contexte droit négatif ajouté automatiquement empêche un deuxième étiquetage des occurrences déjà étiquetées. Attention cependant : les éléments que le programme insère dans la boite après le contexte droit négatif sont des masques lexicaux (voir la justification en section 2.5.4). En cas d'ambiguïté entre un mot et un code grammatical, ils peuvent être interprétés comme des codes grammaticaux. C'est ce qui se produit avec le graphe de 12.12, qui est catastrophique de ce point de vue, car celui de 12.13 reconnait tous les adjectifs du texte !

Les graphes de généralisation d'étiquetage ne fonctionnent que par insertion d'étiquettes lexicales délimitées par des accolades, car le programme trouve les formes à étiqueter dans la liste de tokens du texte.

Lors de la création d'un graphe de généralisation d'étiquetage, plusieurs parties généralisantes peuvent être insérées dans le graphe, et des boîtes classiques peuvent être ajoutées avant et après chaque partie généralisante, comme le montre la figure 12.14.

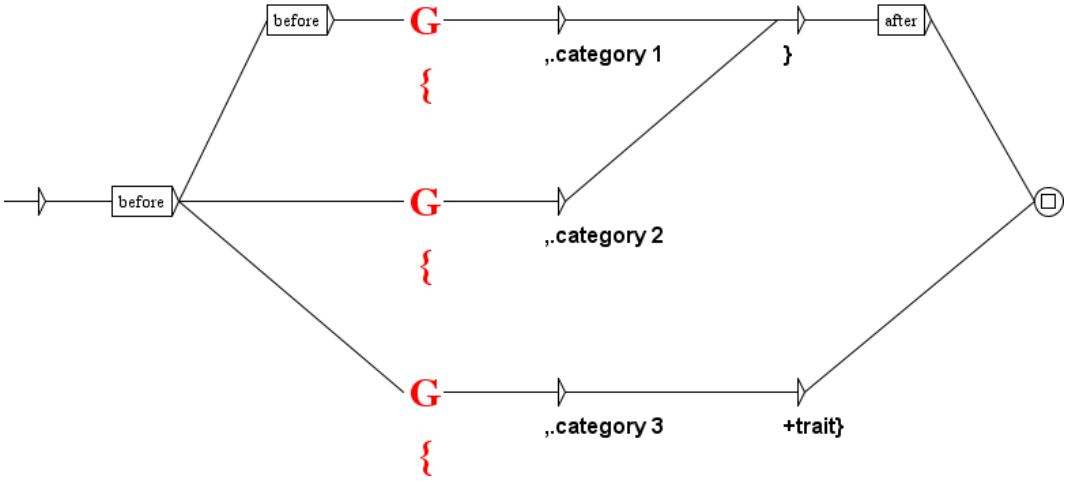


FIGURE 12.14 – Graphe de généralisation d'étiquetage avec plusieurs chemins

12.3.3 Graphes avec restrictions

Il peut arriver que l'on veuille récupérer dans le fichier tokens.txt des formes ayant reçu une sous-étiquette lexicale. Dans le cas d'un token comme :

{{mars,.mois} {2018,.année},.date}

nous appelons "sous-étiquettes lexicales" les éléments *{mars,.mois}* et *{2018,.année}*, et "sous-catégories" les catégories *mois* et *année*. Pour cibler les formes d'une sous-catégorie incluses dans des formes d'une catégorie donnée, on peut utiliser des restrictions dans les graphes de généralisation d'étiquetage[2]. On place dans la boîte le nom de la sous-catégorie[3], avec en sortie la catégorie précédée de ",.". Par exemple, supposons que l'on souhaite étendre les étiquetages *main* et *A* et que la ligne suivante soit présente dans le fichier tokens.txt :

{{first,.A} {{second,.A} {third,.B},.B},.main}

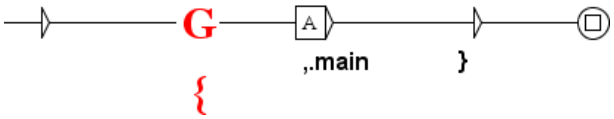


FIGURE 12.15 – Graphe de généralisation d'étiquetage avec une restriction sur la sous-catégorie A

Le graphe en figure 12.15 une fois traité donnera celui de la figure 12.16.

2. Dans certaines versions 3.2 alpha, des restrictions négatives étaient présentes. Elles ne le sont plus, mais le comportement d'une partie généralisante utilisant des restrictions négatives peut être reproduit en utilisant des restrictions positives. Pour plus d'information ou en cas de besoin, contacter la liste Google Unitex.

3. La sous-catégorie doit être seule dans la boîte. Si on veut extraire plusieurs sous-catégories, il faut créer plusieurs chemins.

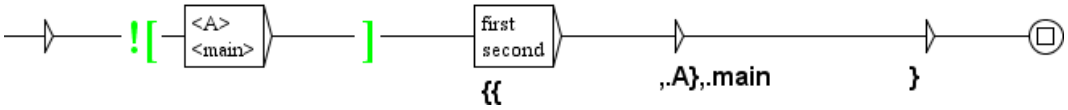


FIGURE 12.16 – Graphe de généralisation d'étiquetage avec une restriction sur la sous-catégorie A, après traitement

Ce graphe généralise la sous-catégorie *A* à deux formes, *first* et *second*. Le contexte droit négatif contient la catégorie du token principal et la sous-catégorie recherchée. Les éléments trouvés sont doublement étiquetés, avec les deux catégories imbriquées.

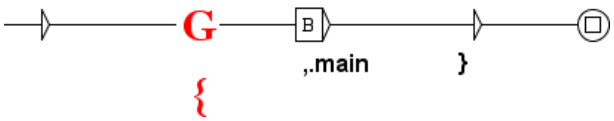


FIGURE 12.17 – Graphe de généralisation d'étiquetage avec une restriction sur la sous-catégorie B

Toujours avec le même fichier tokens.txt, le graphe en figure 12.17 donnera celui de la figure 12.18.

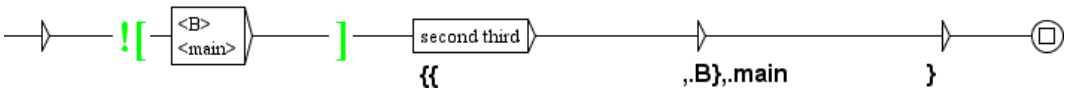


FIGURE 12.18 – Graphe de généralisation d'étiquetage avec une restriction sur la sous-catégorie B, après traitement

Une seule sous-étiquette lexicale de sous-catégorie *B* a été trouvée dans tokens.txt ; cette sous-étiquette lexicale contient elle-même deux autres sous-étiquettes lexicales ; le contenu extrait est donc *second third*. Comme la sous-catégorie correspondait, la recherche n'a pas été propagée aux sous-étiquettes lexicales de niveau inférieur et la sous-étiquette lexicale *third* toute seule n'a pas été trouvée.

Un graphe de généralisation d'étiquetage ne doit pas contenir une partie généralisante sans restriction et une autre avec une restriction pour la même catégorie, à cause de l'ambiguïté qui en résulterait, sauf en utilisant des poids sur les chemins (voir section 5.2.4), comme le montre la figure 12.19.

12.3.4 Remplacement de la catégorie

Il peut arriver que l'on veuille modifier la catégorie qui sera indiquée dans les sous-étiquettes lexicales que l'on extrait de la liste de tokens. Par exemple, dans le cas suivant :

{from {{January,.month} {2017,.year},.date} to {{November,.month} {2018,.year},.date},.period}

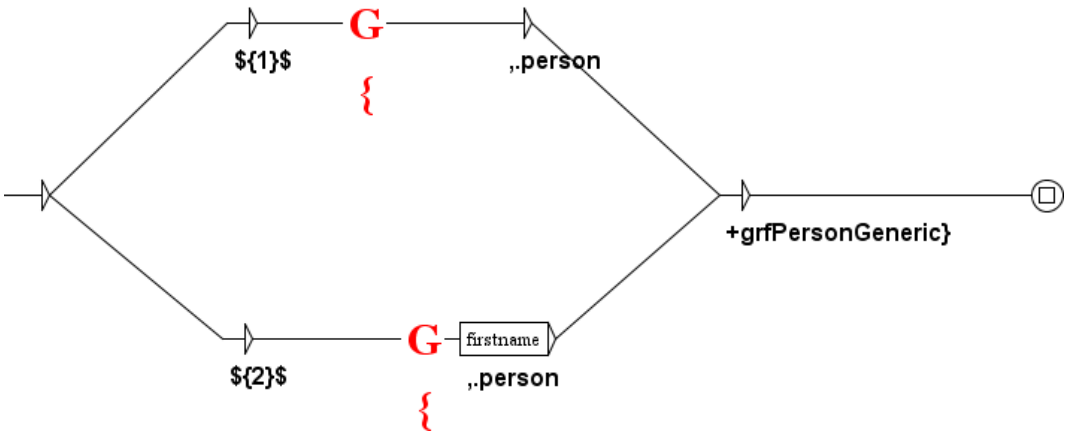


FIGURE 12.19 – Graphe avec des poids pour éviter les ambiguïtés

en utilisant un graphe de généralisation d'étiquetage avec restriction pour extraire les années, les nouvelles étiquettes lexicales insérées seraient :

{{2017,.year},.period} et *{{2018,.year},.period}*

Ici, on voudrait remplacer "period", qui ne représente pas très bien la catégorie de ces occurrences, par "date". Pour cela il est possible de préciser la catégorie finale, en l'ajoutant (après un point) à la sous-catégorie à rechercher (voir les figures 12.20 et 12.21).

FIGURE 12.20 – Graphe avec remplacement de la catégorie

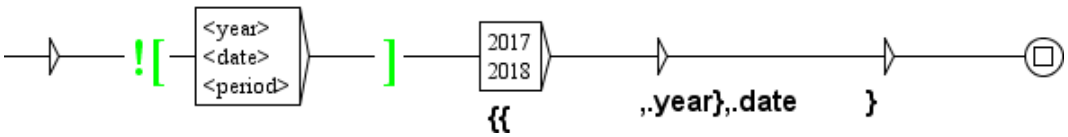


FIGURE 12.21 – Graphe avec remplacement de la catégorie, après traitement

12.4 Les résultats d'une cascade

12.4.1 Affichage des résultats de la cascade

Le résultat de l'application d'une cascade est un fichier d'index (*concord.ind*), comme c'est le cas lors d'une recherche de motif avec *"Locate pattern"*. Ce fichier d'index contient toutes les séquences reconnues conformément aux règles fixées dans Unitex.

Pour afficher une concordance, il suffit de cliquer sur le bouton "Build concordance" (comme décrit au chapitre 6) dans la menu "Text / Located sequences". La figure 12.22 présente un échantillon de concordance d'une cascade qui reconnaît les entités nommées.

FIGURE 12.22 – Concordance de CasSys dans Unitex

12.4.2 Les différents fichiers résultats d'une cascade

CasSys conserve tous les textes créés par chaque graphe de la cascade. Ceci peut être utile pour des tests, le débogage ou la vérification de différents résultats de la cascade. Il est alors possible de corriger les erreurs selon l'ordre d'application des graphes ou de trouver des erreurs dans leur écriture. Il est pratique d'ajouter dans la sortie d'un transducteur le nom de ce dernier, afin de voir dans le résultat final quel motif a été reconnu par quel graphe.

Si l'on applique une cascade au texte exemple.txt, deux répertoires sont créés : `exemple_snt` et `exemple_csc`. Les fichiers créés dans `exemple_csc` sont les résultats obtenus par chaque graphe. Ces fichiers sont intitulés selon le numéro du graphe qui les a produit. Par exemple, si le troisième graphe reconnaît un motif, les résultats de l'application de ce graphe seront stockés dans le répertoire `exemple_3_0_snt` le fichier `exemple_3_0.snt` contiendra le texte modifié.

12.4.3 Un texte au format de type XML pour les étiquettes lexicales

En sortie, le résultat est fourni sous deux formes : le texte résultant directement de l'application des transducteurs, et un format de type XML dans lequel les étiquettes lexicales ont été transformées en XML. Ce changement est fait dans le but de proposer un texte plus manipulable à l'utilisateur final. A partir de ce format, il est possible d'utiliser l'un des nombreux outils de traitement du XML. Il est également facile d'appliquer des transducteurs supplémentaires afin d'obtenir la sortie souhaitée. Le texte résultant directement des transducteurs est sauvegardé dans le fichier `exemple_csc.raw`, et la version XML-isée est dans le fichier `exemple_csc.txt`.

Plus précisement, les étiquettes lexicales sont dans le format suivant :

`{forme.lemme,code1+code2:flex1:flex2}`

La sortie de type XML correspondante a le format suivant :

```
<csc>
        <form>forme</form>
        <lemma>lemme</lemma>
        <code>code1</code>
        <code>code2</code>
        <inflect>flex1</inflect>
        <inflect>flex2</inflect>
</csc>
```

La DTD de notre format est la suivante :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT text (#PCDATA|csc)*>
<!ELEMENT csc (form,lemma?,code*,inflect*) >
<!ELEMENT form (#PCDATA|csc)*>
<!ELEMENT lemma (#PCDATA)>
<!ELEMENT code (#PCDATA)>
<!ELEMENT inflect (#PCDATA)>
```

12.5 La création d'un inventaire d'occurrences balisées

L'option `standoff` de CasSys (section 14.6) permet de traiter un corpus balisé dans le style XML, et de produire un fichier résultat répertoriant les expressions du corpus qui sont délimitées par des balises, avec le nombre d'occurrences de chaque expression distincte. Cette option n'est pas disponible via l'interface d'Unitex et doit être lancée par un script (chapitre 13) ou un programme externe.

Il faut fournir un graphe qui reconnait les éléments à rechercher (éléments au sens XML, c'est-à-dire expressions délimitées par une balise ouvrante et la balise fermante correspondante). Ce graphe aura généralement un contexte droit négatif (section 6.3). Par exemple, le graphe de la figure 12.23 permet d'inventorier les expressions balisées entre `<placeName>` et `</placeName>`.

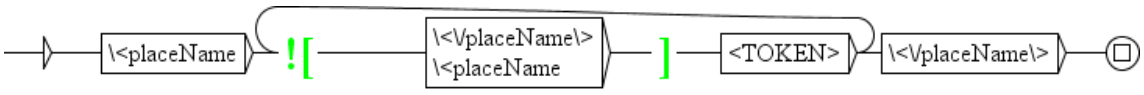


FIGURE 12.23 – Graphe pour chercher les balises *placeName*

Le programme peut traiter le cas où un élément contient un autre élément d'un nom différent[4], mais pas celui où il contient un élément du même nom.

Pour utiliser l'option `standoff`, on lance une cascade contenant le graphe. Le résultat est un fichier dont le nom est suffixé par `_standoff`, et qui liste les expressions trouvées. Il les présente classées suivant les noms d'éléments, et secondairement suivant les valeurs de

4. Le nom d'un élément XML est le nom qui figure à l'intérieur de la balise.

l'attribut `type`, si les balises ouvrantes en possèdent un. Prenons comme exemple le texte balisé suivant :

<placeName type="City">Tours</placeName> est une très belle ville.
<placeName type="City">Tours</placeName> est la capitale
de la <placeName type="Region">Touraine</placeName>.

En y appliquant une cascade avec le graphe de la figure 12.23, on obtient :

```
Voici les expressions trouvées :
     Liste pour l'élément "placeName" et l'attribut "City"
          terme="Tours"
          nombre=2
     Fin de la liste pour cette paire.
     Liste pour l'élément "placeName" et l'attribut "Region"
          terme="Touraine"
          nombre=1
     Fin de la liste pour cette paire.
Fin de la liste pour ce corpus
```

Dans le graphe qui reconnait les éléments, on peut aussi restreindre les valeurs de l'attribut `type`. Par exemple, le graphe de la figure 12.24 permet de répertorier les balises `placeName`, sauf celles de type `Region`.

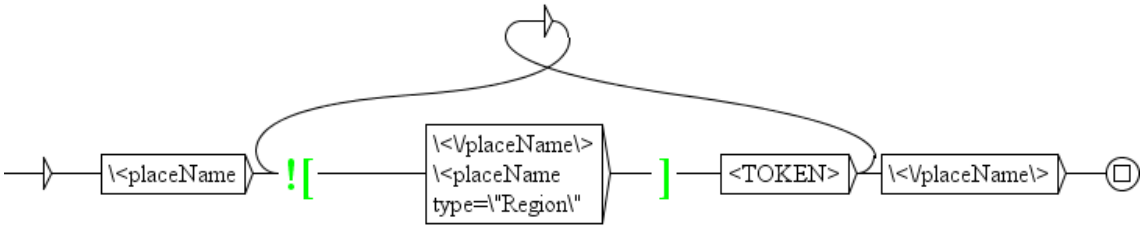


FIGURE 12.24 – Graphe pour les balises *placeName*, sauf celles de type *Region*

En passant une cascade avec le graphe de la figure 12.24, on obtient un résultat avec une seule combinaison élément-type :

```
Voici les expressions trouvées :
     Liste pour l'élément "placeName" et l'attribut "City"
          terme="Tours"
          nombre=2
     Fin de la liste pour cette paire.
Fin de la liste pour ce corpus
```

Le fichier résultat est construit à partir d'un fichier modèle, dont on donne le nom en ligne de commande avec l'option `--standoff=<template file name>`. Le fichier modèle est un fichier texte qui comprend au plus dix sections :

1. un texte introductif libre

2. une ligne contenant juste `#LINE` : la partie entre `#LINE` et `#REST` servira de modèle pour former dans le fichier résultat la partie sur un élément (ou sur une combinaison élément-type)
3. un texte qui introduit un nom d'élément, noté `{TYPE}`, et éventuellement une valeur de l'attribut `type`, notée `{SUBTYPE}`. La partie optionnelle doit être délimitée entre doubles chevrons : `<<...>>`
4. une ligne `#BLOCK` : la partie entre `#BLOCK` et `#END` servira de modèle pour former dans le fichier résultat la partie sur une expression balisée
5. un texte introduisant une expression balisée, notée `{TERM}`
6. un texte introduisant le nombre d'occurrences de cette expression, noté `{COUNT}`
7. une ligne `#END`
8. un texte signalant la fin de la partie sur un élément (ou sur une combinaison élément-type)
9. une ligne `#REST`
10. un texte conclusif libre

Voici le fichier modèle qui correspond à l'exemple ci-dessus :

```
Voici les expressions trouvées :
#LINE
     Liste pour l'élément "{TYPE}"<< et l'attribut "{SUBTYPE}">>
#BLOCK
          terme="{TERM}"
          nombre={COUNT}
#END
     Fin de la liste pour cette paire.
#REST
Fin de la liste pour ce corpus
```

Chapitre 13

Utilisation d'Unitex/GramLab à l'aide de scripts

On peut utiliser Unitex/GramLab par l'intermédiaire de scripts au lieu de l'interface graphique. Le script lance des programmes externes d'Unitex, documentés dans le chapitre 14. L'avantage de cette possibilité est qu'on a accès à des options supplémentaires dans les programmes, par exemple l'option `standoff` de CasSys (section 12.5), et même à des programmes supplémentaires, comme `DumpOffsets` (section 14.13). On peut aussi, pendant le prétraitement, enchainer en cascade plusieurs graphes en mode MERGE, alors que l'interface graphique ne permet d'en lancer qu'un. L'utilisation de scripts nécessite plus d'habileté en informatique, et en particulier plus de familiarité avec le système d'exploitation.

13.1 Traduire en script un traitement lancé via l'interface graphique

Une façon simple d'écrire un script qui utilise Unitex/GramLab est de lancer l'interface graphique Unitex, d'y réaliser une première version du traitement, puis de traduire les clics en une suite de commandes. Pour cela, on peut produire un fichier journal des opérations lancées (section 15.13.8). On peut aussi tirer parti du fait que l'interface graphique d'Unitex garde en mémoire ces opérations et peut les afficher dans la console ("Info > Console", section 14.2, fig. 14.3). On sélectionne les étapes à retenir, on les copie dans un fichier texte et éventuellement on les adapte en un script opérationnel. Pour sélectionner des étapes dans la console, cliquer sur les cellules correspondantes dans la colonne "Log #" (un majuscule-clic et un ctrl-clic ont l'effet habituel pour sélectionner plusieurs éléments). On peut ajouter dans le script des programmes ou options documentés dans le chapitre 14, même s'ils ne sont pas disponibles via l'interface.

On peut formaliser un script sous la forme shell ou batch (section 13.2), ou encore sous la forme d'un script interprété par Unitex (section 13.3).

13.2 Scripts shell ou batch

On peut mettre un script sous la forme shell ou batch et le faire exécuter par l'interpréteur de commandes du système d'exploitation. Par exemple, les commandes suivantes ouvrent le corpus `80jours`, sans le prétraitement en mode REPLACE ni en mode MERGE, mais avec le dictionnaire par défaut, `Dela_fr`, puis lancent un "Locate pattern" avec le graphe `pattern.fst2` et créent une concordance :

```
mkdir "U:\Unitex\French\Corpus\80jours_snt"
"C:\Unitex-GramLab\App\UnitexToolLogger.exe" Normalize
   "U:\Unitex\French/corpus.txt" "-rU:\Unitex\French\Norm.txt"
   "--output_offsets=U:\Unitex\French\Corpus\80jours_snt\normalize.out.offsets" -qutf8-no-bom
"C:\Unitex-GramLab\App\UnitexToolLogger.exe" Tokenize "U:\Unitex\French\Corpus\80jours.snt"
   "-aU:\Unitex\French\Alphabet.txt" -qutf8-no-bom
"C:\Unitex-GramLab\App\UnitexToolLogger.exe" Dico "-tU:\Unitex\French\Corpus\80jours.snt"
   "-aU:\Unitex\French\Alphabet.txt" "C:\Unitex-GramLab\French\Dela\Dela_fr.bin" -qutf8-no-bom
"C:\Unitex-GramLab\App\UnitexToolLogger.exe" SortTxt "U:\Unitex\French\Corpus\80jours_snt\dlf"
   "-lU:\Unitex\French\Corpus\80jours_snt\dlf.n" "-oU:\Unitex\French\Alphabet_sort.txt"
   -qutf8-no-bom
"C:\Unitex-GramLab\App\UnitexToolLogger.exe" SortTxt "U:\Unitex\French\Corpus\80jours_snt\dlc"
   "-lU:\Unitex\French\Corpus\80jours_snt\dlc.n" "-oU:\Unitex\French\Alphabet_sort.txt"
   -qutf8-no-bom
"C:\Unitex-GramLab\App\UnitexToolLogger.exe" SortTxt "U:\Unitex\French\Corpus\80jours_snt\err"
   "-lU:\Unitex\French\Corpus\80jours_snt\err.n" "-oU:\Unitex\French\Alphabet_sort.txt"
   -qutf8-no-bom
"C:\Unitex-GramLab\App\UnitexToolLogger.exe" SortTxt
   "U:\Unitex\French\Corpus\80jours_snt\tags_err"
   "-lU:\Unitex\French\Corpus\80jours_snt\tags_err.n" "-oU:\Unitex\French\Alphabet_sort.txt"
   -qutf8-no-bom
"C:\Unitex-GramLab\App\UnitexToolLogger.exe" Locate "-tU:\Unitex\French\Corpus\80jours.snt"
   "U:\Unitex\French\Graphs\pattern.fst2" "-aU:\Unitex\French\Alphabet.txt" -L -M --all -b -Y
   --stack_max=1000 --max_matches_per_subgraph=200 --max_matches_at_token_pos=400
   --max_errors=50 -qutf8-no-bom
"C:\Unitex-GramLab\App\UnitexToolLogger.exe" Concord
   "U:\Unitex\French\Corpus\80jours_snt\concord.ind" "-fCourier new" -s10 -l40 -r55 --html
   "-aU:\Unitex\French\Alphabet_sort.txt" --CL -qutf8-no-bom
```

Pour les détails des scripts et les commentaires éventuels, il faut respecter les conventions propres au système d'exploitation.

13.3 Scripts interprétés par Unitex/GramLab

Unitex/GramLab a aussi un interpréteur de scripts, qui a plusieurs avantages par rapport à des scripts shell ou batch.

- L'exécution est plus rapide que celle d'un script shell ou batch équivalent, car l'interpréteur fait appel à un système de fichiers virtuel.

- Le script est encapsulé, avec toutes les ressources qu'il utilise, dans un "package linguistique" qu'on peut ensuite déployer dans un autre environnement. De cette façon, la mise au point du script et son utilisation dans une chaine de traitement sont entièrement indépendantes :
  - on peut déployer le package sans aucune connaissance sur Unitex ni sur les traitements linguistiques réalisés par le script,
  - l'environnement de mise au point et l'environnement d'utilisation ne sont pas nécessairement sous le même système d'exploitation,
  - le traitement ne modifie aucune donnée dans l'environnement d'utilisation, car il est réalisé sur une copie du corpus et avec une copie des ressources contenues dans le package. Seuls les fichiers explicitement spécifiés comme fichiers résultats du traitement sont copiés dans l'environnement d'utilisation à la fin du traitement.

Le langage de script reconnu est constitué des programmes externes d'Unitex, y compris quelques programmes qui ont été implémentés spécialement pour l'interpréteur de scripts (section 13.3.4). Il peut y avoir des lignes de commentaires, commençant par le caractère #. Les commandes du système d'exploitation ne sont pas reconnues.

L'interpréteur de scripts d'Unitex/GramLab existe en deux variantes :

- `RunScript` permet de lancer un script sur un corpus contenu dans un ou plusieurs fichiers ;
- `BatchRunScript` permet de traiter un corpus constitué par tous les fichiers d'un répertoire, en lançant le script une fois sur chaque fichier ; il produit à chaque fois un fichier résultat séparé.

13.3.1 Réalisation d'un package linguistique

Un package linguistique pour Unitex/GramLab regroupe un script et toutes les ressources linguistiques qu'il utilise, sauf le corpus d'entrée. Le package est organisé sous la forme d'une arborescence de répertoires, puis compressé au format Zip.

La racine de l'arborescence est un répertoire `<ling_pkg_name>` avec deux sous-répertoires, obligatoirement nommés `script` et `resource`. Le script doit être placé dans le répertoire `<ling_pkg_name>/script` et toutes les ressources linguistiques nécessaires dans `<ling_pkg_name>/resource`. On peut reproduire dans `resource` l'organisation d'un répertoire personnel Unitex, avec un sous-répertoire pour chaque langue traitée par le script (par exemple `French`), puis, à l'intérieur de chaque répertoire de langue, les fichiers d'alphabet `Alphabet.txt` et `Alphabet_sort.txt`, le fichier de normalisation `Norm.txt`, etc. et des sous-répertoires `Dela`, `Graphs`, etc. contenant à leur tour les ressources correspondantes. Avec cette organisation, on peut plus facilement écrire le script en adaptant un script issu de la console.

Si on souhaite que le script ait accès à la date du package, il faut créer dans le répertoire `<ling_pkg_name>/resource` un fichier texte vide dont le nom est `VERSION_AAAAMMJJ` (sans extension).

Une fois cette arborescence réalisée, on la compresse au format Zip.

Par exemple, si on veut encapsuler dans un package `pkg.zip` le traitement de la section 13.2, il faut prévoir dans un répertoire `pkg` :

- dans le sous-répertoire `resource` :
  - les fichiers d'alphabet, `Alphabet.txt` et `Alphabet_sort.txt`, et le fichier de normalisation `Norm.txt` dans `French`;
  - les fichiers dictionnaire `Dela_fr.bin` et `Dela_fr.inf` dans `French/Dela`;
  - le graphe `pattern.fst2` dans `French/Graphs` (on peut aussi y placer optionnellement le fichier `pattern.grf`);
- dans le sous-répertoire `script`, le fichier contenant le script.

13.3.2 Lancement avec RunScript

`RunScript` permet de lancer un script sur un corpus contenu dans un ou plusieurs fichiers. Pour cela, on invoque `UnitexToolLogger` (section 14.49) en respectant la syntaxe suivante :

UnitexToolLogger [{ SelectOutput <args1> }] { InstallLingResourcePackage <args2> } { RunScript <args3> } { InstallLingResourcePackage <args4> }

Chaque paire d'accolades délimite l'invocation d'une commande Unitex avec ses arguments.

1. L'étape `SelectOutput` est facultative mais recommandée. En l'incluant sous la forme suivante :
   ```
   { SelectOutput --output=off }
   ```
   on masque les sorties de mise au point de chaque commande ultérieure, qui sont encombrantes dans le cas d'une exécution massive.

2. L'étape suivante, `InstallLingResourcePackage`, installe le contenu du package linguistique dans le système de fichiers virtuel.

3. L'étape `Runscript` lance le script. Elle doit définir dans `INPUT_FILE_1` et éventuellement `INPUT_FILE_2`, etc., le nom du ou des fichiers contenant le corpus d'entrée ; dans `OUTPUT_FILE_1`, `OUTPUT_FILE_2`, etc., le nom du ou des fichiers résultats ; dans `PACKAGE_DIR` le nom du répertoire racine du package linguistique dans le système de fichiers virtuel ; dans `CORPUS_WORK_DIR` le nom du répertoire qui contiendra les fichiers créés pendant l'exécution du script.

4. La dernière étape écrit le ou les fichiers résultats, puis met fin au système de fichiers virtuel.

Par exemple, la commande suivante utilise le package linguistique `pkg.zip` de la section 13.3.1 et lance le script `uniscript` contenu dans ce package, avec comme corpus d'entrée `80jours.txt` et comme fichier de sortie `80jours_result.html`.

```
"C:\Unitex-GramLab\App\UnitexToolLogger.exe" { SelectOutput --output=off } {
   InstallLingResourcePackage -p U:\Scripts\pkg.zip -x "$:UnitexPkgResource" -v } { RunScript -v
   -a INPUT_FILE_1=U:\Unitex\French\Corpus\80jours.txt
   -a "CORPUS_WORK_DIR=$:UnitexPkgWork" -a "PACKAGE_DIR=$:UnitexPkgResource"
   -a OUTPUT_FILE_1=U:\Unitex\French\Corpus\80jours_result.html
   "$:UnitexPkgResource\script\uniscript" } { InstallLingResourcePackage -p U:\Scripts\pkg.zip
   -x "$:UnitexPkgResource" -u -v }
```

13.3.3 Lancement avec BatchRunScript

`BatchRunScript` permet de traiter un corpus constitué par tous les fichiers d'un répertoire. Il lance le script une fois sur chaque fichier, produisant à chaque fois un fichier résultat séparé. On peut ainsi réaliser des traitements massifs efficacement, même sur des documents nombreux contenus dans des fichiers séparés. Le résultat est stocké dans un répertoire de sortie, dans des fichiers portant les mêmes noms que les fichiers d'entrée, suffixés par défaut [1] par `.result.txt`. `BatchRunScript` fonctionne en multithread et lance plusieurs exécutions en parallèle : ainsi tous les cœurs de la machine de traitement peuvent fonctionner en même temps et traiter l'ensemble des fichiers plus vite.

Pour déployer un script avec `BatchRunScript`, on respecte la syntaxe suivante :

```
UnitexToolLogger { BatchRunScript <args> }
```

Les arguments doivent définir le nom du répertoire contenant le corpus d'entrée dans le système de fichiers virtuel, le nom du répertoire de sortie, le nombre de threads, le nom du package linguistique et le nom du script. Par exemple, la commande suivante traite tous les fichiers du répertoire `input_folder`, stocke les résultats dans `output_folder` et fonctionne sur quatre threads :

```
"C:\Unitex-GramLab\App\UnitexToolLogger.exe" { BatchRunScript -i .\input_folder -o .\output_folder
   -t 4 .\pkg -v -p -m -s script/uniscript }
```

Les options `-v`, `-m` et `-p` permettent de contrôler quelles informations sont émises sur le terminal pendant les traitements. Ainsi, en utilisant `-v -p`, on affiche le maximum de sorties, ce qui est utile pendant la mise au point. Avec `-m` et plus encore `-f`, on affiche le minimum de messages.

Avec `BatchRunScript`, une exécution du script sur un fichier ne peut pas renvoyer plusieurs fichiers, contrairement à `RunScript` [2].

1. On peut spécifier un autre suffixe avec l'option `-x`. Avec l'option `-e`, l'extension du fichier d'entrée est retirée avant d'ajouter le suffixe.

2. Il est possible de contourner cette limitation avec les outils `PackFile` et `UnpackFile`, en regroupant plusieurs fichiers résultats en un seul fichier de type Zip.

13.3.4 Mise au point d'un script pour RunScript

Pour écrire un script dans un package linguistique, il est recommandé d'adapter un script issu de la console, comme celui de la section 13.2.

- Dans chaque ligne du script issu de la console, supprimer la mention du programme `UnitexToolLogger` en début de ligne, c'est-à-dire dans notre exemple : `"C:\Unitex-GramLab\App\UnitexToolLogger.exe"`.
- Dans les noms des ressources, remplacer le chemin du répertoire de travail par : `{PACKAGE_DIR}/resource`
  Dans notre exemple, le fichier de normalisation pour le français devient : `{PACKAGE_DIR}/resource/French/Norm.txt`
- Dans les noms des fichiers créés par le script, remplacer le chemin complet par : `{CURRENT_WORK_DIR}`
  Dans notre exemple, le nom du fichier `80jours.snt` devient : `{CURRENT_WORK_DIR}/80jours.snt`

La valeur de `PACKAGEDIR` est déjà définie quand le script commence à s'exécuter, mais pour `CURRENT_WORK_DIR` il faut commencer le script par quelques lignes qui donnent un nom à ce répertoire, le créent, y copient le corpus d'entrée et font quelques autres préparatifs :

```
CURRENT_WORK_DIR = {CORPUS_WORK_DIR}/{UNIQUE_VALUE}
```

Cette ligne donne un nom à `CURRENT_WORK_DIR`. La valeur de `CORPUS_WORK_DIR` est déjà définie quand le script commence à s'exécuter. La variable `UNIQUE_VALUE` contient une chaine de caractères définie par `RunScript` ou par `BatchRunScript`, avec la garantie que la chaine est différente pour chaque exécution de `RunScript` ou pour chaque thread de `BatchRunScript` s'exécutant simultanément. Cela permet d'éviter toute collision entre fichiers temporaires pendant des travaux simultanés. Ensuite, comme la commande `mkdir` n'est pas utilisable dans un script interprété par Unitex/GramLab, on utilise `DuplicateFile` pour créer le répertoire :

```
DuplicateFile -p {CURRENT_WORK_DIR}
```

On y copie le corpus d'entrée :

```
DuplicateFile -i {INPUT_FILE_1} {CURRENT_WORK_DIR}/corpus.txt
```

Dans notre exemple, la première commande du script issu de la console utilisait la commande `mkdir` pour créer le sous-répertoire `corpus_snt`. On doit donc la remplacer par `DuplicateFile` :

```
DuplicateFile --make-dir {CURRENT_WORK_DIR}/corpus_snt
```

Les deux lignes suivantes écrivent dans des fichiers des informations sur la version d'Unitex utilisée :

```
VersionInfo -n -o {CURRENT_WORK_DIR}/newrevision.txt
VersionInfo -s -o {CURRENT_WORK_DIR}/semver.txt
```

La ligne suivante sauvegarde la date de version du package linguistique, à condition d'avoir inclus dans le package un fichier `VERSION_AAAAMMJJ` (voir section 13.3.1) :

```
VersionInfo -B -o {CURRENT_WORK_DIR}/builddate.txt
```

Le corps du script peut apparaitre ensuite :

```
Normalize "{CURRENT_WORK_DIR}/corpus.txt" "-r{PACKAGE_DIR}/resource/French/Norm.txt"
   "--output_offsets={CURRENT_WORK_DIR}/corpus_snt/normalize.out.offsets" -qutf8-no-bom
Tokenize "{CURRENT_WORK_DIR}/corpus.snt" "-a{PACKAGE_DIR}/resource/French/Alphabet.txt"
   -qutf8-no-bom
Dico "-t{CURRENT_WORK_DIR}/corpus.snt" "-a{PACKAGE_DIR}/resource/French/Alphabet.txt"
   "{PACKAGE_DIR}/resource/French/Dela/Dela_fr.bin" -qutf8-no-bom
SortTxt "{CURRENT_WORK_DIR}/corpus_snt/dlf" "-l{CURRENT_WORK_DIR}/corpus_snt/dlf.n"
   "-o{PACKAGE_DIR}/resource/French/Alphabet_sort.txt" -qutf8-no-bom
SortTxt "{CURRENT_WORK_DIR}/corpus_snt/dlc" "-l{CURRENT_WORK_DIR}/corpus_snt/dlc.n"
   "-o{PACKAGE_DIR}/resource/French/Alphabet_sort.txt" -qutf8-no-bom
SortTxt "{CURRENT_WORK_DIR}/corpus_snt/err" "-l{CURRENT_WORK_DIR}/corpus_snt/err.n"
   "-o{PACKAGE_DIR}/resource/French/Alphabet_sort.txt" -qutf8-no-bom
SortTxt "{CURRENT_WORK_DIR}/corpus_snt/tags_err"
   "-l{CURRENT_WORK_DIR}/corpus_snt/tags_err.n"
   "-o{PACKAGE_DIR}/resource/French/Alphabet_sort.txt" -qutf8-no-bom
Locate "-t{CURRENT_WORK_DIR}/corpus.snt"
   "{PACKAGE_DIR}/resource/French/Graphs/test.fst2"
   "-a{PACKAGE_DIR}/resource/French/Alphabet.txt" -L -M --all -b -Y --stack_max=1000
   --max_matches_per_subgraph=200 --max_matches_at_token_pos=400 --max_errors=50
   -qutf8-no-bom
Concord "{CURRENT_WORK_DIR}/corpus_snt/concord.ind" "-fCourier new" -s10 -l40 -r55 --html
   "-a{PACKAGE_DIR}/resource/French/Alphabet_sort.txt" --CL -qutf8-no-bom
```

La fin du script doit indiquer le nom du ou des fichiers résultats et libérer la mémoire :

```
DuplicateFile -i {CURRENT_WORK_DIR}/corpus_snt/concord.html {OUTPUT_FILE_1}
DuplicateFile --recursive-delete {CURRENT_WORK_DIR}
```

On peut lancer le script avec la commande de la section 13.3.3. Le résultat obtenu est juste un fichier de concordance `80jours_result.html`, placé à côté de `80jours.txt`.

Si on ajoute au script, en avant-dernière position, la ligne :

```
DuplicateFile -i {CURRENT_WORK_DIR}/corpus_snt/dlf {OUTPUT_FILE_2}
```

on doit aussi ajouter à la commande de lancement l'option :

```
-a OUTPUT_FILE_2=U:\Unitex\French\Corpus\80jours_result.dic
```

On obtient alors comme résultat supplémentaire un fichier dictionnaire, placé à côté des deux autres et nommé `80jours_result.dic`.

Chapitre 14

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 (en fait, `UnitexToolLogger` est appelé, afin de réduire de manière importante la taille du fichier zip). Il est possible de voir les commandes qui ont été exécutées en cliquant sur "Info>Console". Il est aussi possible de voir les options des differents programmes dans "Info>Help on commands" (voir Figure 14.1). Remarquons que tous les programmes Unitex possèdent l'option `-h/--help`.

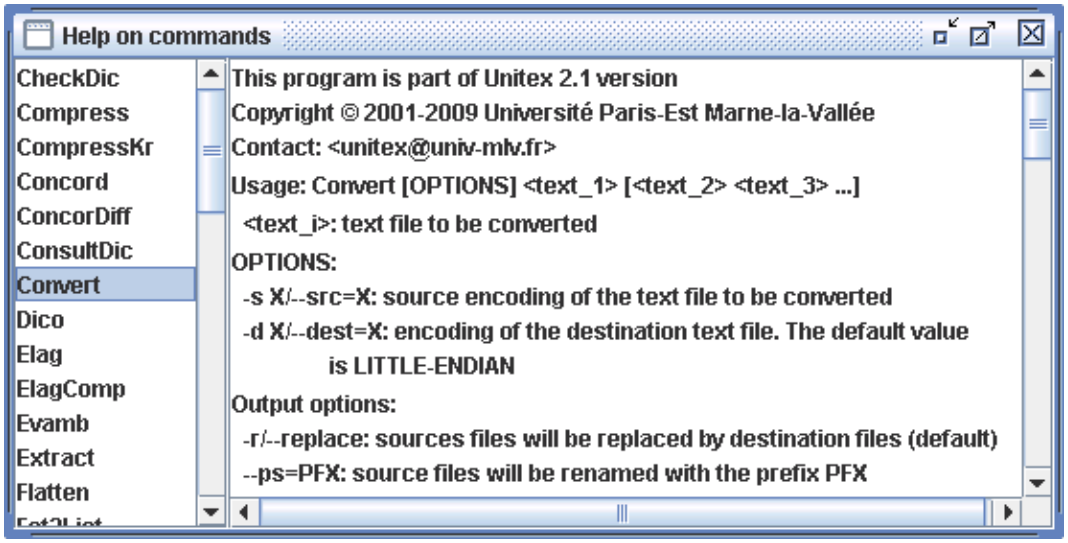


FIGURE 14.1 – Help on commands

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 devrez 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.

ATTENTION (3) : beaucoup de programmes utilisent un fichier `Alphabet.txt`. Cette information peut être omise pour l'ensemble de ces programmes. Dans ce cas, une définition (par défaut) de lettres est utilisée (voir `u_is_letter` dans le fichier source`Unicode.cpp`).

14.1 Création de fichiers log

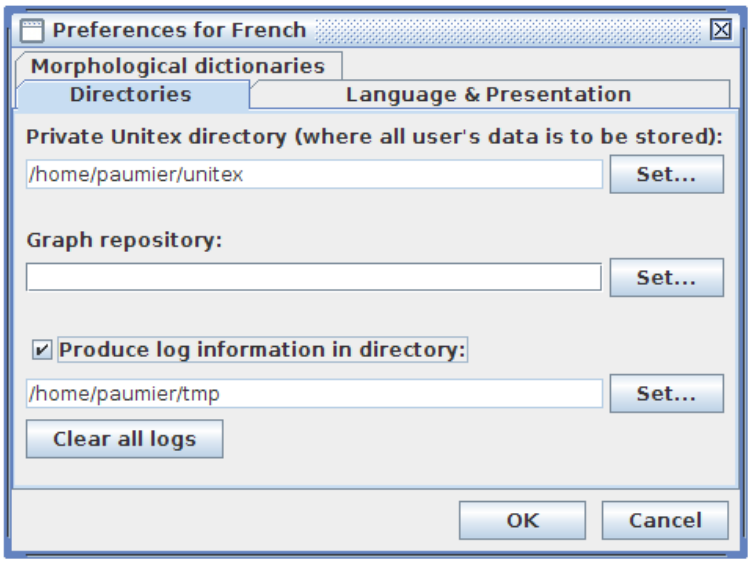


FIGURE 14.2 – Configuration de fichiers log

Vous pouvez créer des fichiers `log` des programmes externes exécutés. Ces fichiers log peuvent être utiles pour le débogage ou des tests de régression. Vous avez juste besoin d'activer cette fonctionnalité dans le cadre Préférences. Vous devez simplement choisir un répertoire de fichiers log dans lequel tous les fichiers sont stockés, et cocher la case "Produce log" En cliquant sur le bouton "Clear all logs" vous supprimez tous les fichiers log éventuellement contenus dans ce répertoire. Désormais, toute nouvelle exécution du promme produit un fichier `unitex_log_XXX.ulp` dans le répertoire de fichiers log. `XXX` représente le numéro de log qui se trouve dans la console (voir section suivante).

14.2 La console

Lorsque Unitex lance un programme externe, la ligne de commande appelée est mémorisée dans la console. Pour la voir, cliquez sur Info "> Console". Quand une commande n'émet aucun message d'erreur, elle est affichée avec une icône verte. Sinon, l'icône est un

triangle rouge sur lequel vous pouvez cliquer pour voir les messages d'erreur, comme indiqué sur la figure 14.3 . Ceci est utile lorsque un message d'erreur se produit si vite que vous ne pouvez pas le lire. Si une commande a été enregistrée, son numéro de log apparaît dans la deuxième colonne. Notez que vous pouvez exporter toutes les commandes affichées dans la console vers le presse-papiers avec Ctrl + C.

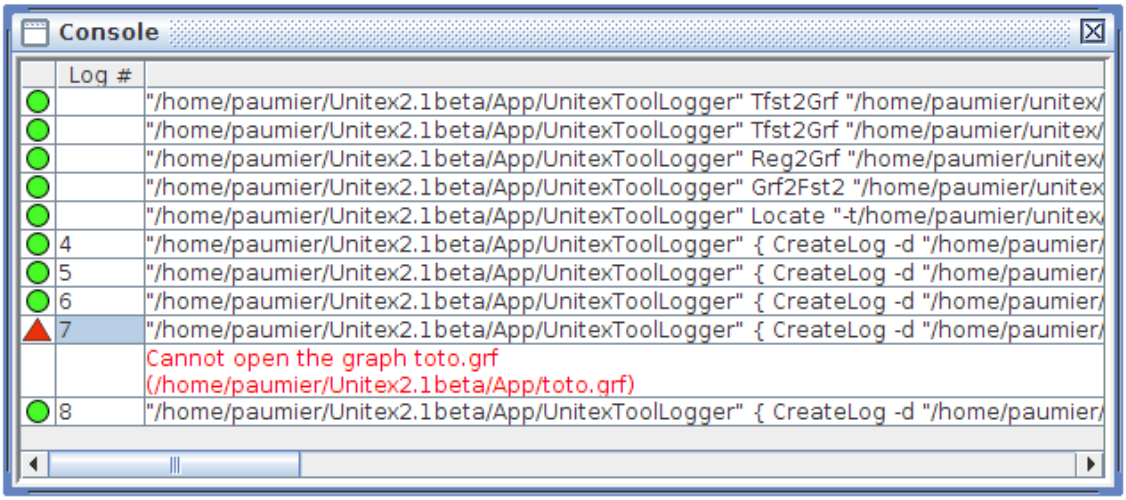


FIGURE 14.3 – Console

14.3 Unitex JNI

Vous pouvez utilisez Unitex avec JNI by en incluant les imports suivants :

```
import fr.umlv.unitex.jni.UnitexJni;
import java.io.*;
import fr.umlv.unitex.*;
```

Ceci vous permet de charger en mémoire les dictionnaires (.bin), les grammaires ou graphes dictionnaires (.fst2) et les fichiers alphabet et de les garder en mémoire de manière persistante. Vous utilisez alors le nom de fichier renvoyé par la foncton loadPersistent*.

```
String persistentAlphabet = UnitexJni.loadPersistentAlphabet("/.../unitex/Fren
String persistentFst2 = UnitexJni.loadPersistentFst2("/.../unitex/French/Dela/
String persistentDictionary = UnitexJni.loadPersistentDictionary(
"/.../unitex/French/Dela/communesFR+.bin");
```

14.4 Paramètres de codage des fichiers textes

Unitex utilise Unicode pour les fichiers textes 15.1. Tous les programmes qui lisent ou écrivent des fichiers textes partagent les mêmes paramètres d'encodage. Les formats pos-

sibles sont utf16le-bom, utf16le-no-bom, utf16be-bom, utf16be-no-bom, utf8-bom, utf8-no-bom, qui correspondent à Unicode Big-Endian, Little-Endian et UTF-8, avec ou sans "Unicode byte order mark" (bom) au début du fichier. Pour le format d'entrée, vous pouvez spécifier plusieurs encodages *-bom (avec bom) codage séparées par des virgules, mais seulement un encodage *-no-bom (sans bom).

OPTIONS :

- `-k=ENCODING/--input_encoding=ENCODING` : format du texte source. Peut contenir plusieurs valeurs séparées par des virgules ;
- `-q=ENCODING/--output_encoding=ENCODING` : format du texte de sortie.

Par défaut, les valeurs sont : `--input_encoding=utf16le-bom,utf16be-bom,utf8-bom --output_encoding=utf16le-bom`.

14.5 BuildKrMwuDic

```
BuildKrMwuDic [OPTIONS] dic
```

Ce programme génère des graphes de flexion pour les mots composés à partir d'un tableau `dic` qui décrit chaque constituant de chaque mot composé **OPTIONS :**

- `-o GRF/--output=GRF` : fichier `.grf` à produire ;
- `-d DIR/--directory=DIR` : répertoire de flexion qui contient les graphes de flexion nécéssaires pour produire les variantes morphologiques des racines ;
- `-a ALPH/--alphabet=ALPH` : fichier alphabet à utiliser ;
- `-b BIN/--binary=BIN` : dictionnaire des mots simples de type `.bin` à utiliser ;

14.6 CasSys

```
Cassys [OPTIONS] <snt>
```

Ce programme applique une liste ordonnée de grammaires à un texte et construit un index des occurrences trouvées. **OPTIONS :**

- `-a ALPH/--alphabet=ALPH` : fichier alphabet de la langue ;
- `-r X/--transducer_dir=X` : prend un transducteur dans le répertoire `X` (cela évite de donner le chemin complet pour chaque transducteur) ; `X` doit se terminer par un (anti)slash ;

- `-w DIC/--morpho=DIC` : indique que `DIC` est un `.bin` dictionnaire à utiliser en mode morphologique. Utiliser autant de `-m XXX` qu'il y a de `.bin`. Il est également possible de séparer plusieurs `.bin` par le caractère deux-points.
- `-l TRANSDUCERS_LIST/--transducers_list=TRANSDUCERS_LIST` : fichier contenant la liste des transducteurs avec leur mode d'application ;
- `-s transducer.fst2/--transducer_file=transducer.fst2` : un transducteur à appliquer ;
- `-m output_policy/--transducer_policy=output_policy` : le mode d'application du transducteur spécifié ;
- `-t TXT/--text=TXT` : le fichier texte avec l'extension `.snt` à modifier ;
- `-i/--in_place` : signifie qu'il faut utiliser les mêmes répertoires `csc/snt` pour chaque transducteur ;
- `-d/--no_create_directory` : signifie que tous les répertoires `snt/csc` existent déjà et n'ont pas besoin d'être crées ;
- `-g minus/--negation_operator=minus` : utilise `moins` comme opérateur de négation pour les graphes version Unitex 2.0 ;
- `-g tilde/--negation_operator=tilde` : utilise `tilde` comme opérateur de négation (par défaut) ;
- `--standoff=` : indique le chemin et le nom du fichier indiquant le modèle de standoff et lance la création du standOff du texte ;
- `-h/--help` : affiche cette aide

CasSys applique une liste de grammaires à un texte et sauve les séquences reconnues dans un fichier index nommé `concord.ind` stocké dans le répertoire texte. Le fichier cible doit être un fichier snt avec son répertoire `_snt`. Le fichier contenant la liste des transducteurs est un fichier dans lequel chaque ligne contient le nom complet du transducteur suivi de son mode d'application.

A la place d'une liste, vous pouvez spécifier chaque fichier et mode d'application par un ensemble de couple d'arguments pour représenter la liste `-s/--transducer\_file` et `-m/--transducer\_policy`

Le mode d'application peut être MERGE ou REPLACE.

L'option de fichier, l'option alphabet et l'option fichier liste de transducteurs sont obligatoires

Comme le programme Locate, ce programme enregistre les références des occurrences dans un fichier `concord.ind` stocké dans le répertoire `_snt` du texte. Le fichier `concord.ind`

produit est dans le même format que celui décrit chapitre 15 , mais la cascade peut être formée de graphes appliqués en mode merge ou replace, de ce fait #M ou #R à la première ligne du fichier `concord.ind` n'a pas de sens dans ce contexte.

14.7 CheckDic

```
CheckDic [OPTIONS] dic
```

Ce programme effectue la vérification du format d'un dictionnaire de type DELAS ou DELAF.`dic` qui correspond au nom du dictionnaire à vérifier

OPTIONS :

- `-f/--delaf` : vérifie un dictionnaire de formes fléchies ;
- `-s/--delas` : vérifie un dictionnaire de formes canoniques ;
- `-r/--strict` : vérification stricte de la syntaxe, la déspécialisation des points et virgules ;
- `-t/--tolerate` : tolère des points et des virgules non déspécialisés (par défaut) ;
- `-n/--no_space_warning` : tolère des espaces dans les codes grammaticaux/sémantiques/flexionn
- `-p/--skip_path` : n'affiche pas le chemin complet du dictionnaire (utiles pour la compatibilité de fichiers de log sur plusieurs systèmes) ;
- `-a ALPH/--alphabet=ALPH` : indique le fichier alphabet à utiliser.

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`.

Le choix de `--strict` permet de détecter l'utilisation de points non déspécialisés dans la forme fléchie ou de virgules non déspécialisées dans la forme canonique. L'option `--tolerate` se comporte comme dans les versions Unitex 2.0 et antérieures et ne les détecte pas.

14.8 Compress

```
Compress [OPTIONS] dictionary
```

OPTIONS :

- `-o BIN/--output=BIN` : définit le fichier de sortie. Par défaut, un fichier `xxx.dic` produit un fichier `xxx.bin` ;

- `-f/--flip` : indique que les formes fléchies et canoniques doivent être inversées dans le dictionnaire comprimé. Cette option est utilisée pour construire un dictionnaire inversé nécessaire au programme `Reconstrucao` ;
- `-s/--semitic` : indique que l'algorithme de compression pour langue sémitique doit être utilisé. Cette option utilisée avec des langues sémitiques comme l'arabe réduit sensiblement la taille du dictionnaire produit ;
- `--v1` : produit un fichier `.bin` ancienne manière ;
- `--v2` : produit un fichier `.bin` nouvelle manière, mieux comprimé et sans limitation de taille de fichier à 16 Mb (par défaut)

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 15.

14.9 Concord

```
Concord [OPTIONS] <index>
```

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 :

OPTIONS :

- `-f FONT/--font=FONT` : nom de la police de caractères à utiliser si la sortie est fichier HTML ;
- `-s N/--fontsize=N` : taille de la police si la sortie est fichier HTML. Les paramètres concernant la police sont ignorés si la sortie n'est pas au format HTML ;
- `--only_ambiguous` : Affiche seulement les occurrences identiques avec une sortie ambiguë, dans l'odre du texte.
- `--only_matches` : cette option définit un mode sans contexte. En outre si elle est utilisée avec `-t/--text`, Concord n'entoure pas les séquences reconnues de tabulations

- `-l X/--left=X` : nombre de caractères à gauche des occurrences (par défaut=0). Dans le mode Thai, ceci correspond au nombre de caractères non diacritiques.
- `-r X/--right=X` : nombre de caractères (non diacritiques dans le mode Thai) à droite des occurrences (par défaut=0). Si l'occurrence est plus petite que cette valeur, la ligne de concordance est complétée jusqu'à `right`. Si l'occurrence est plus longue que la valeur définie par `right`, elle est néanmoins entièrement conservée.

  NOTE : Pour `--left` et `--right`, vous pouvez ajouter le caractère `s` pour arrêter au premier symbole de fin de phrase `{S}`. Par exemple, si vous mettez `40s` comme valeur de gauche, le contexte gauche sera au plus à 40 caractères, moins si le `{S}` est trouvé avant.

Options de tri :

- `--TO` : ordre dans lequel les occurrences apparaissent dans le texte (par défaut) ;
- `--LC` : contexte gauche comme premier tri, occurrence comme second tri ;
- `--LR` : contexte gauche, contexte droit ;
- `--CL` : occurrence, contexte gauche ;
- `--CR` : occurrence, contexte droit ;
- `--RL` : contexte droit, contexte gauche ;
- `--RC` : contexte droit, occurrence.

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

Options de sortie :

- `-H/--html` : produit une concordance au format HTML codée en UTF-8 ; (par défaut) ;
- `-t/--text` : produit une concordance au format texte Unicode ;
- `-g SCRIPT/--glossanet=SCRIPT` : produit une concordance pour GlossaNet au format HTML. Le fichier HTML produit est codé en UTF-8 ;
- `-p SCRIPT/--script=SCRIPT` : produit une concordance au format HTML où les occurrences sont liens décrits par `SCRIPT`. Par exemple, si vous utilisez `-phttp://www.google.com/search?q=`, vous obtiendrez une concordance au format HTML où les occurrences sont des liens vers des requêtes Google ;
- `-i/--index` : produit un index de la concordance, qui comporte les occurrences (avec les sorties des grammaires, s'il y en a), précédées par les positions des occurrences, dans le fichier texte, exprimées en caractères ;

- `-u offsets/--uima=offsets` : produit un index de la concordance relatif fichier texte original, avant toute opération effectuée par Unitex. Offsets est le fichier produit par Tokenize avec l'option `--output_offsets`

- `-e/--xml` : produit un index xml de la concordance ;

- `-w/--xml-with-header` : produit un index xml de la concordance avec une en-tête xml complète ;

- `--lemmatize` : produit un fichier de concordance HTML spécial utilisé par l'interface de lemmatisation de l'interface graphique d'Unitex.

  REMARQUE : les options -e et -w acceptent toutes deux un fichier d'offset, comme l'accepte -u

- `--PRLG=X,Y` : produit une concordance pour des corpus PRLG où chaque ligne est préfixée par l'information extraite avec l'option `--PRLG` de Unxmlize. X est le fichier produit par l'option `--PRLG` de Unxmlize et Y est le fichier produit par l'option `--output_offsets` de Tokenize. Remarquons que si cette option est utilisée en plus avec `-u`, l'argument Y remplace l'argument de `-u` ;

- `-A/--axis` : presque pareil que `--index`, mais les nombres représentent le caractère médian de chaque occurrence. Pour plus d'information, consultez [32] ;

- `-x/--xalign` : un autre fichier index, utilisé par le module d'alignement de texte. Chaque ligne est formée de 3 entiers X Y Z suivi du contenu de l'occurrence. X est numéro de la phrase, partant de 1. Y et Z sont les positions de début et de fin de l'occurrence dans la phrase exprimée en caractères ;

- `-m TXT/--merge=TXT` : indique au programme qu'il doit produire une version modifiée du texte et l'enregistrer dans le fichier dénommé `TXT` (voir section 6.10.4).

- `T--export_csv` : produit un fichier avec le séparateur tabulation export.csv dans l'ordre du texte avec le format suivant :

  A B C D E F, où :

  A=nombre de lignes dans le fichier .csv

  B=nombre de phrases

  C= référence PRLG, si elle existe

  D=la forme fléchie présente dans le texte

  E=le lemme, s'il existe

  F=les codes, s'il y en a

  Pour fonctionner, cette option doit re appelée pour des fichier concord.ind, qui ne contiennent pas de token S ni espace

Autres options :

- `-d DIR/--directory=DIR` : indique au programme qu'il ne doit pas travailler avec le même répertoire que `<index>` mais avec `DIR` ;
- `-a ALPH/--alphabet=ALPH` : fichier alphabet utilisé pour le tri ;
- `-T/--thai` : option à utiliser pour les concordances en Thai.

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`, `--glossanet` ou `--script`, 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 `<a href="X Y Z">`. `X` et `Y` représentent les positions de début et de fin de l'occurrence en caractères dans le fichier `text_name.snt`. `Z` représente le numéro de la phrase dans laquelle apparaît l'occurrence.

14.10 ConcorDiff

```
ConcorDiff [OPTIONS] <concor1> <concor2>
```

Ce programme prend deux fichiers de concordance et produit une page HTML montrant les différences entre ces deux concordances (voir section 6.10.6, page 167). `<concor1>` et `<concor2>` fichiers de concordances (.ind) doivent avoir des noms absolus, car Unitex en déduit le texte sur lequel elles ont été calculées ;

OPTIONS :

- `-o X/--out=X` : page HTML de sortie ;
- `-f FONT/--font=FONT` : police à utiliser dans le page HTML de sortie ;
- `-s N/--size=N` : taille de police à utiliser dans le page HTML de sortie ;
- `-d/--diff_only` : ne pas afficher les séquences identiques ;

14.11 Convert

```
Convert [OPTIONS] <text_1> [<text_2> <text_3> ...]
```

Ce programme permet de transcoder des fichiers textes.

OPTIONS :

- `-s X/--src=X` : encodage d'entrée ;
- `-d X/--dest=X` : encodage de sortie (par défaut=`LITTLE-ENDIAN`) ;

Options de translitération (seulement pour l'arabe) :

- `-F/--delaf` : l'entrée est un DELAF et l'on veut seulement translitérer les formes fléchies et canoniques ;
- `-S/--delas` : l'entrée est un DELAS et l'on veut seulement translitérer les formes canoniques.

Options de sortie :

- `-r/--replace` : la conversion écrase les fichiers source (par défaut) ;
- `-o file/--output=file` : nom du fichier de destination (seulement un fichier à convertir) ;
- `--ps=PFX` : les fichiers sources sont renommés avec le préfixe `PFX` ; (`toto.txt` ⇒ `PFXtoto.txt`);
- `--pd=PFX` : les fichiers destinations sont renommés avec le préfixe `PFX` ;
- `--ss=SFX` : les fichiers sources sont renommés avec le suffixe `SFX` ; (`toto.txt` ⇒ `totoSFX.txt`);
- `--sd=SFX` : les fichiers destinations sont renommés avec le suffixe `SFX`.

Options HTML :
`Convert` offre des options spéciales pour les fichiers HTML. Vous pouvez utiliser une combinaison des options suivantes :

- `--dnc` (Decode Normal Chars) : des séquences comme `é` `x` et `ø` sont décodées comme un unique caractère unicode, sauf si elles representent un caractère de contrôle HTML ;
- `--dcc` (Decode Control Chars) : `<` `>` `&` et `"` sont décodés comme < > & et les quote (de même pour leur représentation décimales et hexadécimales) ;
- `--eac` (Encode All Chars) : chaque caractère non supporté par l'encodage de sortie est représenté par une chaîne comme `ǉ`
- `--ecc` (Encode Control Chars) : < > & et les quote sont encodés par `<` `>` `&` et `"`

Par défaut, toutes les options HTML sont désactivées.

Autres options :

- `-m/--main-names` : imprime la liste des noms principaux des encodage ;
- `-a/--aliases` : imprime la liste des alias d'encodage ;

- `-A/--all-infos` : imprime toutes les information concernant tous les encodages ;
- `-i X/--info=X` : imprime toutes les information concernant l'encodage X.

Les encodages prennent leurs valeurs dans la liste suivante (liste non exhaustive, voir ci-dessous) :

`FRENCH`
`ENGLISH`
`GREEK`
`THAI`
`CZECH`
`GERMAN`
`SPANISH`
`PORTUGUESE`
`ITALIAN`
`NORWEGIAN`
`LATIN` (default latin code page)
`windows-1252` : Microsoft Windows 1252 - Latin I (Western Europe & USA)
`windows-1250` : Microsoft Windows 1250 - Central Europe
`windows-1257` : Microsoft Windows 1257 - Baltic
`windows-1251` : Microsoft Windows 1251 - Cyrillic
`windows-1254` : Microsoft Windows 1254 - Turkish
`windows-1258` : Microsoft Windows 1258 - Viet Nam
`iso-8859-1` : ISO 8859-1 - Latin 1 (Europe de l'ouest & USA)
`iso-8859-15` : ISO 8859-15 - Latin 9 (Western Europe & USA)
`iso-8859-2` : ISO 8859-2 - Latin 2 (Eastern and Central Europe)
`iso-8859-3` : ISO 8859-3 - Latin 3 (Southern Europe)
`iso-8859-4` : ISO 8859-4 - Latin 4 (Northern Europe)
`iso-8859-5` : ISO 8859-5 - Cyrillic
`iso-8859-7` : ISO 8859-7 - Greek
`iso-8859-9` : ISO 8859-9 - Latin 5 (Turkish)
`iso-8859-10` : ISO 8859-10 - Latin 6 (Nordic)
`next-step` : NextStep code page
`LITTLE-ENDIAN`
`BIG-ENDIAN`
`UTF8`

14.12 Dico

```
Dico [OPTIONS] <dic_1> [<dic_2> <dic_3>...]
```

Ce programme applique des dictionnaires à un texte. Le texte doit avoir été découpé en unités lexicales par le programme `Tokenize`.

OPTIONS :

- `-t TXT/--text=TXT` : nom complet du fichier texte `.snt` ;
- `-a ALPH/--alphabet=ALPH` : le fichier alphabet à utiliser ;
- `-m DICS/--morpho=DICS` : ce paramètre optionnel liste les dictionnaires du mode morphologique, si la présence de dictionnaires `.fst2` rend cette information nécessaire. `DICS` représente une liste de fichiers `.bin` (avec leur nom complet) séparés par des points-virgules ;
- `-K/--korean` : indique à `Dico` qu'il travaille sur du coréen ;
- `-s/--semitic` : indique à `Dico` qu'il travaille sur une langue sémitique (nécessaire si `Dico` doit compresser un dictionnaire) ;
- `-u X/--arabic_rules=X` : désigne le fichier de configuration des règles typographiques de l'arabe ;.
- `r X/--raw=X` : indique que `Dico` devrait simplement produire un fichier de sortie X contenant les mots simples et composés, sans exiger un répertoire texte. Si X est omis, les résultats sont affichés sur la sortie standard.

`<dic_i>` représente le chemin d'accès complet à un dictionnaire. Le dictionnaire doit être soit un dictionnaire compressé au format `.bin` (obtenu avec le programme `Compress`) soit un graphe dictionnaire au format `.fst2` (voir section 3.8, page 69). Il est possible de donner des priorités aux dictionnaires. Pour les détails voir section 3.8.1.

Le programme `Dico` produit les 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 ;
- `tags_err` : mots simples inconnus qui ne sont pas reconnus par le fichier `tags.ind` ;
- `tags.ind` : séquences à insérer dans l'automate du texte (see section 3.8.3, page 70) ;
- `stat_dic.n` : fichier contenant les nombres de mots simples, composés et inconnus du texte.

NOTE : Les fichiers`dlf`, `dlc`, `err` and `tags_err` ne sont pas triés. Utilisez le programme `SortTxt` pour le faire.

14.13 DumpOffsets

Ce programme permet d'étudier et d'utiliser les fichiers de correspondance d'Offsets, manipulé par certains outils Unitex comme Unxmlize, Normalize, Fst2Txt, Tokenize, Concord et GrfTest.

```
DumpOffsets --merge -o <fichier_offsets1> <fichier_offsets2>
  -p <fichier_offset12>
```

En entrée, le fichier offsets1 (15.13.10, page 358) contient la correspondant des offsets entre un fichier en version A et un fichier en version B, et offset2 contient la correspondant des offsets entre ce fichier en version B et en version C, le fichier fichier_offset12 résultant aura la correspondance entre les versions A et C.

```
DumpOffsets [OPTIONS] -o <fichier_version1> -n <fichier_Version2>
  <fichier_offset> -p <fichier_dump>
```

OPTIONS :

- `-f/--full` : Inclus des informations plus complètes

En entrée, le fichier fichier_offset contient la correspondant des offsets entre le fichier_version1 et le fichier_version2. En sortie, le fichier texte <fichier_dump> contiendra la comparaison des séquences entre les 2 fichiers et vérifiera leur cohérence. Ce fichier est destiné à une lecture manuelle, afin d'étudier le contenu du fichier d'offset

```
DumpOffsets [OPTIONS] --convert_modified_to_common
  <fichier_offset_différence> -p <fichier_offset_zone_commune>
```

OPTIONS :

- `-s N/--old_size=N` : Contient la taille en caractère de la version d'origine du fichiet texte
- `-S N/--new_size=N` : Contient la taille en caractère de la version d'arrivée du fichiet texte

Il faut obligatoirement spécifier une des deux tailles. Pour un fichier encodé en UTF16BE_BOM, c'est la taille en octets, auquel on retranche 2 pour les 2 octets de signature BOM et que l'on divise ensuite par 2 car chaque caractère unicode prend 2 octets. En UTF8, la correspondance n'est pas immédiate.

Converti un fichier d'offset indiquant les caractères supprimés (tel que fournis par les autres outils Unitex) en fichier indiquant les plages de caractères identiques (15.13.11).

```
DumpOffsets [OPTIONS] --convert_common_to_modified
  <fichier_offset_zone_commune> -p <fichier_offset_différence>
```

OPTIONS :

- `-s N/--old\_size=N` : Contient la taille en caractère de la version d'origine du fichier texte
- `-S N/--new\_size=N` : Contient la taille en caractère de la version d'arrivée du fichier texte

Il faut obligatoirement spécifier les deux tailles.

Converti un fichier d'offset indiquant les plages de caractères identiques en fichier indiquant les caractères supprimés.

OPTIONS :

- `-d/--denormalize` : Denormalize l'output

Typiquement ce program permet de rétablir les espaces, les espaces insecables, les sauts de ligne, les tabulations, etc. effacés par Normalize. Il rétablit aussi le texte qui a été supprimé par le Preprocessing ou par un graphe. Il conserve les ajouts à condition que ceux-ci soient placés entre chevrons.

Le fichier_dump contient les textes du fichier version 1 et les textes entre chevron (<, >) qui étaient ajoutés en fichier version 2.

```
DumpOffsets [OPTIONS] -d -o <fichier_version1>
-n <fichier_version2> <fichier_offset> -p <fichier_dump>
```

Autre Utilisation : `DumpOffsets -o <list_of_position_file_to_read.txt>`

`<list_of_position_file_to_read.txt>` est un fichier avec seulement un nombre par ligne (une position).

Ceci convertit une liste de positions en utilisant le fichier d'offsets. Le fichier créé contient à chaque ligne la nouvelle position suivi d'un + si le caractère à cette position est dans le fichier d'arrivée, suivi d'un - si le caractère a été supprimé.

- `-p <list_to_create> -T <offset_file_to_read>`

Utiliser `-t` à la place de `-T` produit la traduction inverse.

14.14 Elag

```
Elag [OPTIONS] <tfst>
```

Ce programme prend un fichier `.tfst` automate de texte `<tfst>` et lui applique des règles de levée d'ambiguïtés.

OPTIONS :

- `-l LANG/--language=LANG` : Le fichier de configuration ELAG pour la langue considérée
- `-r RULES/--rules=RULES` : le fichier de règles compilées au format `.rul` ;
- `-o OUT/--output=OUT` : l'automate du texte de sortie.

14.15 ElagComp

```
ElagComp [OPTIONS]
```

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

OPTIONS :

- `-r RULES/--rules=RULES` : fichier listant des grammaires ELAG ;
- `-g GRAMMAR/--grammar=GRAMMAR` : une grammaire ELAG donnée ;
- `-l LANG/--language=LANG` : le fichier de configuration ELAG pour la langue considérée ;
- `-o OUT/--output=OUT` : nom du fichier de sortie. Par défaut, le fichier de sortie est identique à `RULES`, sauf pour l'extension qui est `.rul`.

14.16 Evamb

```
Evamb [OPTIONS] <tfst>
```

Ce programme calcule un taux d'ambiguïté moyen sur tout l'automate du texte `<tfst>`, ou juste sur la phrase spécifiée par`N`. Les résultats du calcul sont affichés sur la sortie standard. L'automate du texte n'est pas modifié par ce programme.

OPTIONS :

- `-o OUT/--output=OUT` : nom de fichier optionnel ;
- `-s N/--sentence=N` : numéro de phrase.

14.17 Extract

```
Extract [OPTIONS] <text>
```

Ce programme extrait de ce texte toutes les phrases qui contiennent au moins une des occurrences de la concordance. Le paramètre `<text>` représente le nom complet du fichier texte, sans omettre l'extension `.snt`.

OPTIONS :

- `-y/--yes` : extrait toutes les phrases qui contiennent des séquences reconnues (par défaut) ;
- `-n/--no` : extrait toutes les phrases qui ne contiennent pas de séquence reconnue
- `-o OUT/--output=OUT` : nom du fichier de sortie ;
- `-i X/--index=X` : le fichier `.ind` qui décrit la concordance. Par défaut, `X` est le fichier `concord.ind` situé dans le répertoire du texte

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

14.18 Flatten

```
Flatten [OPTIONS] <fst2>
```

Ce programme prend une grammaire `.fst2` en paramètre, et essaye de la transformer en un transducteur à états finis.

OPTIONS :

- `-f/--fst` : la grammaire est "dépliée" à la profondeur maximum et tronquée si des appels à des sous-graphes existent. Les appels tronqués sont remplacés par des transitions vides. Le résultat est une grammaire `.fst2` qui contient un unique transducteur à états finis ;
- `-r/--rtn` : les appels aux sous-graphes qui subsistent après transformation sont laissés tels quels. Le résultat est un transducteur à états finis dans le cas favorable, et une grammaire optimisée strictement équivalente à l'originale dans le cas contraire (par défaut) ;
- `-d N/--depth=N` : profondeur maximum à laquelle les appels aux graphes devraient être dépliés. La valeur par défaut est 10.

14.19 Fst2Check

```
Fst2Check [OPTIONS] <fst2>
```

Ce programme vérifie si un fichier .fst2 n'a pas d'erreurs Locate.

OPTIONS :

- `-y/--loop_check` : active la vérification d'erreurs (détection de boucles) ;
- `-n/--no_loop_check` : désactive la vérification d'erreurs (par défaut) ;
- `-t/--tfst_check` : vérifie si le graphe donné peut être considéré comme un automate de phrases ou non ;
- `-e/--no_empty_graph_warning` : pas d'émission de warning quand les graphes reconnaissent le mot vide. Cette option est utilisée par `MultiFlex` pour ne pas effrayer les utilisateurs par des messages d'erreurs inadéquats lorsqu'ils construisent une grammaire de flexion qui reconnaît le mot vide

Options de sortie :

- `-o file/--output=file` : fichier de sorties pour les messages d'erreurs ;
- `-a/--append` : ouvre un fichier de message d'erreurs en mode append ;
- `-s/--statistics` : affiche les statistique du fichier `.fst2`.

14.20 Fst2List

```
Fst2List [-o out][-p s/f/d][-[a/t] s/m][-m][-f s/a][-s "Str"]
         [--io_separator "Str"] [--stop_mark "Str"]
         [-r [s/l/x] "Str"] [-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` ;
- `-S` : Affiche le résultat sur la sortie standard. Exclusif avec `-o` ;
- `-[a/t] s/m` : précise si l'on tient compte (`t`) ou non (`a`) 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 `subname`. Ce paramètre peut être utilisé plusieurs fois, afin de spécifier plusieurs graphes d'arrêts

- `-p s/f/d` : `s` produit l'affichage des chemins de chaque sous-graphe de la grammaire ; `f` (par défaut) affiche les chemins globaux de la grammaire ; `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 apparait entre angles par le caractère unicode de code hexadécimal `0xXXXX` ;

- `-s "L[,R]"` : spécifie les délimiteurs gauche (`L`) et droit (`R`) qui entoureront les items. Par défaut, ces délimiteurs sont nuls ;

- `-g/--io_separator "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 `in0 out0 in1 out1` (`a`). La valeur par défaut est `s` ;

- `-q/--stop_mark "stop"` : arrête l'exploration à `"<stop>"`. La valeur par défaut est `null` ;

- `-v` : ce paramètre produit l'affichage de messages d'informations ; (mode verbose) ;

- `-m` : mode spécial pour description avec alphabet ;

- `-rx "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 14.4, voici les résultats que l'on obtient si l'on pose `L="["` et `R="]*"` :

```
il fait [très très]*
il fait très beau
```

FIGURE 14.4 – Graphe avec un cycle

14.21 Fst2Txt

```
Fst2Txt [OPTIONS] <fst2>
```

Ce programme applique un transducteur à un texte en phase de prétraitement, quand le texte n'est pas encore découpé en unités lexicales

OPTIONS :

- `-t TXT/--text=TXT` : le fichier texte à modifier, avec l'extension `.snt` ;
- `-a ALPH/--alphabet=ALPH` : le fichier alphabet de la langue du texte ;
- `-s/--start_on_space` : ce paramètre indique que la recherche va commencer à n'importe quelle position dans le texte, même avant un espace. Ce paramètre ne devrait être utilisé que pour effectuer des recherches morphologiques ;
- `-x/--dont_start_on_space` : interdit au programme de reconnaître des séquences commençant par un espace (par défaut) ;
- `-c/--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 comme le Thaï ;
- `-w/--word_by_word` : fonctionne en mode mot par mot (par défaut) ;
- `--input_offsets=XXX` : fichier offset à utiliser.

Options de sorties :

- `-M/--merge` : ajoute les sorties du transducteur aux séquences reconnues texte d'entrée (par défaut) ;
- `-R/--replace` : remplace les séquences reconnues avec les sorties correspondantes du transducteur.
- `--output_offsets=XXX` : fichier offset à produire

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

14.22 Grf2Fst2

```
Grf2Fst2 [OPTIONS] <grf>
```

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

OPTIONS :

- `-y/--loop_check` : active la vérification d'erreurs (détection de boucles) ;

- `-n/--no_loop_check` : désactive la vérification d'erreurs (par défaut) ;
- `-a ALPH/--alphabet=ALPH` : spécifie le fichier d'alphabet à utiliser pour faire le découpage en unités lexicales du contenu des boîtes de la grammaire.
- `-c/--char_by_char` : le découpage se fait caractère par caractère. Si ni `-c` ni `-a` ne sont utilisés, le découpage s'effectue en prenant des suites de lettres Unicode.
- `-d DIR/--pkgdir=DIR` : définit le répertoire de dépôt à utiliser pour compiler la grammaire (voir section 5.2.2, page 103).
- `-e/--no_empty_graph_warning` : pas d'émission de warning quand les graphes reconnaissent le mot vide. Cette option est utilisée par `MultiFlex` pour ne pas effrayer les utilisateurs par des messages d'erreurs inadéquats lorsqu'ils construisent une grammaire de flexion qui reconnaît le mot vide.
- `-t/--tfst_check` : vérifie si le graphe donné peut être considéré comme un automate de phrases ou non ;
- `-s/--silent_grf_name` : n'affiche pas le nom des graphes (nécessaire pour l'utilisation de fichiers log sur plusieurs systèmes) ;
- `-r XXX/--named_repositories=XXX` : déclaration de noms de répertoires de dépôt. XXX est formé d'une séquence d'un ou plusieurs X=Y, séparés par ';', où X est le nom du répertoire de dépôt désigné par le chemin Y. Vous pouvez utiliser cette option à plusieurs reprises ;
- `--debug` : compile les graphes en mode debug ;
- `-v/check_variables` : vérifier la validité de sortie afin d'éviter des expressions avec variables malformées.

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 `<grf>`.

14.23 GrfDiff

`GrfDiff <grf1> <grf2>: fichier fichiers .grf` à comparer
OPTIONS :

- `--output X` : sauve le résultat éventuel dans X au lieu de l'afficher

Compare les fichier .grf et affiche leurs différence sur la sortie standard. Renvoie 0 s'il sont identiques modulo le réordonnancement des boîtes et des transitions, 1 si ils sont différents, 2 en cas d'erreur.

Voici les indications que GrfDiff peut émettre :

- `P name` : présentation d'une propriété a changé. name= nom propriété name (SIZE, FONT, ...)

- `M a b` : une boîte est déplacée. a=numéro de boîte dans <grf1>, b=numéro de boîte dans <grf2>
- `C a b` : le contenu d'une boîte a changé. a=numéro de boîte dans <grf1>, b=numéro de boîte dans <grf2>
- `A x` : une boîte a été ajoutée. x=numéro de boîte dans <grf2>
- `R x` : une boîte a été supprimée. x=numéro de boîte dans <grf1>
- `T a b x y` : une transition a été ajoutée. a,b=src et dst numéros de boîtes dans <grf1>. x,y=src et dst numéros de boîtes dans <grf2>
- `X a b x y` : transition a été supprimée. a,b=src et dst numéros de boîtes dans <grf1>. x,y=src et dst numéros de boîtes dans <grf2>

Remarquons que les modifications concernant les transitions liées aux boîtes ajoutées ou supprimées sont rapportées.

14.24 GrfDiff3

GrfDiff3 <mine> <base> <other>
<mine> : mon fichier .grf <other> : l'autre fichier .grf qui produit un conflit <base> : fichier .grf ancêtre commun

OPTIONS :

- `--output X` : enregistre le résultat, le cas échéant, dans X et pas sur la sortie
- `--conflicts X` : enregistre la description des conflits, le cas échéant, dans X
- `--only-cosmetic` : signale un conflit de tout changement qui n'est pas purement cosmétique

Essaye de regrouper les <mine> et <other>. En cas de succès, le résultat est imprimé sur la sortie standard et 0 est renvoyé. En cas de conflits non résolus, 1 est renvoyé et rien n'est imprimé. 2 est renvoyé en cas d'erreur.

14.25 ImplodeTfst

```
ImplodeTfst [OPTIONS] <tfst>
```

Ce programme implose l'automate du texte spécifié en fusionnant ensemble les entrées lexicales qui ne diffèrent que par leurs catactéristiques flexionnelles.

OPTIONS :

- `-o OUT/--output=OUT` : fichier de sortie. Par défaut, l'automate du texte est modifiée.

14.26 Locate

```
Locate [OPTIONS] <fst2>
```

Ce programme applique une grammaire à un texte et construit un fichier d'index des occurrences trouvées.

OPTIONS :

- `-t TXT/--text=TXT` : chemin complet du fichier texte, sans omettre l'extension `.snt` ;
- `-a ALPH/--alphabet=ALPH` : chemin d'accès complet au fichier alphabet ;
- `-m DICS/--morpho=DICS` : ce paramètre optionnel indique quels dictionnaires morphologiques sont utilisés, s'ils sont exigés par des dictionnaires `.fst2 DICS` représente une liste de fichiers `.bin` (avec leurs chemins complets) séparés par des points-virgules ;
- `-s/--start_on_space` : ce paramètre indique que la recherche va commencer à n'importe quelle position dans le texte, même avant un espace. Ce paramètre ne devrait être utilisé que pour effectuer des recherches morphologiques ;
- `-x/--dont_start_on_space` : interdit au programme de reconnaitre des séquences commençant par un espace (par défaut) ;
- `-c/--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 comme le Thaï ;
- `-w/--word_by_word` : fonctionne en mode mot par mot (par défaut) ;
- `-d DIR/--sntdir=DIR` : met les fichiers produits dans le répertoire au lieu `DIR` au lieu du répertoire texte. Notez que `DIR` doit se terminer par un séparateur de fichier (\ or /) ;
- `-K/--korean` : indique `Locate` qu'il travaille sur du coréen ;
- `-u X/--arabic_rules=X` : désigne le fichier de configuration des règles typographiques de l'arabe ;
- `-g X/--negation_operator=X` : spécifie l'opérateur de négation à utiliser dans les masques lexicaux. Les deux valeurs possibles de `X` sont `moins` et `tilde` (par défaut). Utiliser `moins` offre une compatibilité descendante avec les versions précédentes de Unitex.

Options de limite de recherche :

- `-l/--all` : recherche toutes les séquences reconnues (par défaut) ;
- `-n N/--number_of_matches=N` : stoppe après les premiers `N` matches.

Options du nombre d'itérations maximum par token :

- `-o N/--stop_token_count=N` : stoppe après N itérations sur un token ;
- `-o N,M/--stop_token_count=N,M` : émet un warning après N itérations sur un token et s'arrête après itérations M.

Options du mode de reconnaissance :

- `-S/--shortest_matches` ;
- `-L/--longest_matches` (par défaut) ;
- `-A/--all_matches`.

Options de sortie :

- `-I/--ignore` : ignore les sorties du transducteur (par défaut) ;
- `-M/--merge` : ajoute les sorties du transducteur avec les séquences reconnues ;
- `-R/--replace` : remplace les séquences reconnues par les sorties correspondantes du transducteur ;
- `-p/--protect_dic_chars` : quand le mode `-M` ou `-R` est utilisé, `-p` protège certains caractères de l'entrée avec un antislash. Ceci est utile quand `Locate` est appelée par `Dico` afin d'éviter la production de mauvaises lignes comme :

  `3,14,.PI.NUM`

- `-v X=Y/--variable=X=Y` : définit une variable de sortie nommé `X` avec un contenu `Y`. Remarquons que Y doit être ASCII.

Options de sortie ambiguës :

- `-b/--ambiguous_outputs` : permet la production de plusieurs matchs avec la même entrée, mais différentes sorties (par défaut) ;
- `-z/--no_ambiguous_outputs` : interdit les sorties ambiguës. Dans le cas de sorties ambiguës, l'une sera arbitrairement choisie, en fonction de l'état interne du programme.

Options d'erreur sur les variables
Ces options n'ont aucun effet si le mode de sortie est réglé avec `--ignore` ; sinon, elles définissent le comportement du programme `Locate` quand une sortie contient une référence à une variable qui n'est pas correctement définie.

- `-X/--exit_on_variable_error` : arrête le programme ;

- `-Y/--ignore_variable_errors` : agit comme si la variable avait un contenu vide (par défaut) ;
- `-Z/--backtrack_on_variable_errors` : arrêter d'explorer le chemin courant de la grammaire.

Injection de variables :

- `-v X=Y/--variable=X=Y` : définit une variable de sortie nommée X avec un contenu Y. Notez que Y doit être ASCII.

Ce programme enregistre les références des occurrences trouvées dans un fichier appelé `concord.ind`. Le nombre d'occurrences, le nombre d'unités appartenant à ces occurrences, ainsi que le pourcentage d'unités reconnues dans le texte sont enregistrées dans un fichier appelé `concord.n`. Ces deux fichiers sont stockés dans le répertoire du texte.

14.27 LocateTfst

```
LocateTfst [OPTIONS] <fst2>
```

Ce programme applique une grammaire à l'automate du texte, et sauve l'indes des séquences reconnues dans un fichier `concord.ind`, comme le fait `Locate`.

OPTIONS :

- `-t TFST/--text=TFST` : chemin complet du fichier texte, sans omettre l'extension ;
- `-a ALPH/--alphabet=ALPH` : chemin d'accès complet au fichier alphabet ;
- `-K/--korean` : indique à `LocateTfst` qu'il travaille sur du coréen ;
- `-g X/--negation_operator=X` : spécifie l'opérateur de négation à utiliser dans les masques lexicaux. Les deux valeurs possibles de `X` sont `moins` et `tilde` (par défaut). Utiliser `moins` offre une compatibilité descendante avec les versions précédentes de Unitex.

Options de limite de recherche :

- `-l/--all` : recherche toutes les séquences reconnues (par défaut) ;
- `-n N/--number_of_matches=N` : stoppe après les premiers `N` matches.

Options du mode de reconnaissance :

- `-S/--shortest_matches` ;
- `-L/--longest_matches` (par défaut) ;

- `-A/--all_matches`.

Options de sortie :

- `-I/--ignore` : ignore les sorties du transducteur (par défaut) ;
- `-M/--merge` : ajoute les sorties du transducteur avec les séquences reconnues ;
- `-R/--replace` : remplace les séquences reconnues par les sorties correspondantes du transducteur ;

Options de sortie ambiguës :

- `-b/--ambiguous_outputs` : permet la production de plusieurs matchs avec la même entrée, mais différentes sorties (par défaut) ;
- `-z/--no_ambiguous_outputs` : interdit les sorties ambiguës. Dans le cas de sorties ambiguës, l'une sera arbitrairement choisie, en fonction de l'état interne du programme.

Options d'erreur sur les variables
Ces options n'ont aucun effet si le mode de sortie est réglé avec `--ignore` ; sinon, elles définissent le comportement du programme `Locate` quand une sortie une référence à une variable qui n'est pas correctement définie.

- `-X/--exit_on_variable_error` : arrête le programme ;
- `-Y/--ignore_variable_errors` : agit comme si la variable avait un contenu vide (par défaut) ;
- `-Z/--backtrack_on_variable_errors` :arrêter d'explorer le chemin courant de la grammaire.

Injection de variables

- `-v X=Y/--variable=X=Y` : définit une variable de sortie nommée `X` avec un contenu `Y`. Notez que Y doit être ASCII.

Option d'étiquetage

- `--tagging` : indique que la concordance doit être tagguée, et contenir les informations supplémentaires sur les états de début et de fin de chaque match.

Ce programme enregistre les références des occurrences trouvées dans un fichier appelé `concord.ind`. Le nombre d'occurrences, et le nombre de sorties produites sont enregistrées dans un fichier appelé `concord_tfst.n`. Ces deux fichiers sont stockés dans le répertoire du texte.

14.28 MultiFlex

```
MultiFlex [OPTIONS] <dela>
```

Ce programme effectue la flexion automatique d'un dictionnaire DELA contenant des formes canoniques 3.1.2) de mots simples ou composés (see chapter 11).

OPTIONS :

- `-o DELAF/--output=DELAF` : fichier DELAF de sortie ;
- `-a ALPH/--alphabet=ALPH` : fichier alphabet ;
- `-d DIR/--directory=DIR` : le répertoire contenant les fichiers `Morphology` et `Equivalences` et des graphes de flexion pour mots simples ou composés ;
- `-K/--korean` : indique à `MultiFlex` qu'il travaille sur du coréen ;
- `-s/--only-simple-words` : le programme tiendra compte des mots composés comme des erreurs ;
- `-c/--only-compound-words` : le programme tiendra compte des mots simples comme des erreurs ;
- `-p DIR/--pkgdir=DIR` : indique le répertoire des graphes.
- `-rXXX/--named_repositories=XXX` : déclaration des dépôts nommés. XXX est formée d'une séquence ou plus X=Y , séparés par ; où X est le nom de dépôt désigné par le chemin Y. Vous pouvez utiliser cette option à plusieurs reprises ;

Remarquons que les transducteurs de flexion `.fst2` sont automatiquement construits à partir des fichiers `.grf` correspondants en cas d'absence ou de fichiers `.grf` plus anciens.

14.29 Normalize

```
Normalize [OPTIONS] <text>
```

Ce programme effectue une normalisation des séparateurs de texte. Les séparateurs sont l'espace, la tabulation, et le saut de ligne. Chaque séquence de séparateurs qui contient au moins un saut de ligne est remplacé par un saut de ligne unique. Toutes les autres séquences de séparateurs sont remplacées par un seul 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}`, le marqueur `{STOP}`, soit une ligne de DELAF valide (`{aujourd'hui,.ADV}`).

Le paramètre `<text>` 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`.

OPTIONS :

- `-n/--no_carriage_return` : chaque séquence de séparateurs sera transformée en un espace unique ;
- `--input_offsets=XXX` : fichier offset à utiliser.
- `--output_offsets=XXX` : fichier offset à produire
- `-r XXX/--replacement_rules=XXX` : indique la règle de normalisation à utiliser. Voir section 15.13.6 Pour plus de détails sur le format de ce fichier. Par défaut, le programme ne remplace que `{` and `}` par `[` et `]`.
- `--no_separator_normalization` : n'applique que des règles de remplacement spécifiées par -r

ATTENTION : si vous spécifiez un fichier de règles de normalisation, ces règles seront appliquées avant toute autre chose. Donc, il faut être très prudent si vous manipulez les séparateurs dans ces règles.

14.30 PolyLex

```
PolyLex [OPTIONS] <list>
```

Ce programme prend en paramètre un fichier de mots inconnus `<list>` 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`.

OPTIONS :

- `-a ALPH/--alphabet=ALPH` : le fichier alphabet à utiliser ;
- `-d BIN/--dictionary=BIN` : le dictionnaire .bin à utiliser ;
- `-o OUT/--output=OUT` : désigne le fichier dans lequel les lignes de dictionnaire produites doivent être enregistrées, si ce fichier existe déjà, les lignes sont ajoutées à la fin du fichier ;
- `-i INFO/--info=INFO` : désigne un fichier texte dans lequel les informations relatives à l'analyse a été réalisée.

Options de langue :

- `-D/--dutch`
- `-G/--german`

- `-N/--norwegian`
- `-R/--russian`

NOTE : pour les mots hollandais ou norvégiens, le programme tente de lire un fichier texte contenant une liste de mots interdits. Ce fichier est supposé s'appeler `ForbiddenWords.txt` (voir section 15.13.7) et être stocké dans le même répertoire que `BIN`.

14.31 RebuildTfst

```
RebuildTfst <tfst>
```

Ce programme reconstruit l'automate du texte `<tfst>` en tenant compte des modifications manuelles. Si le programme trouve un fichier `sentenceN.grf` dans le même répertoire que `<tfst>`, il remplace l'automate de la phrase `N` par celle représentée par `sentenceN.grf`. L'automate du texte donné entrée est modifié.

14.32 Reconstrucao

```
Reconstrucao [OPTIONS] <index>
```

Le programme génère une grammaire de normalisation destinée à être appliquée avant la construction d'un automate pour un texte en langue portugaise. Le fichier `<index>` représente une concordance qui doit être produite en mode MERGE to the considered text a grammar that extracts all forms to be normalized. Cette grammaire est nommée `V-Pro-Suf`, et est stockée dans le répertoire `/Portuguese/Graphs/Normalization`.

OPTIONS :

- `-a ALPH/--alphabet=ALPH` : le fichier alphabet à utiliser ;
- `-r ROOT/--root=ROOT` : le dictionnaire inversé `.bin` à utiliser pour retrouver les formes au futur et au conditionnel à partir des formes canoniques. Il a été obtenu en compressant le dictionnaire des verbes au futur et au conditionnel avec le paramètre `--flip` (voir section 14.8) ;
- `-d BIN/--dictionary=BIN` : le dictionnaire `.bin` à utiliser ;
- `-p PRO/--pronoun_rules=PRO` : la grammaire `.fst2` de réécriture des pronoms ;
- `-n PRO/--nasal_pronoun_rules=PRO` : la grammaire `.fst2` de réécriture des pronoms nasaux ;
- `-o OUT/--output=OUT` : le nom du graphe `.grf` à générer

14.33 Reg2Grf

```
Reg2Grf <txt>
```

Ce programme construit un fichier `.grf` correspondant à l'expression rationnelle contenue dans le fichier `<txt>`. Le paramètre `<txt>` 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 `<txt>`.

14.34 Seq2Grf

```
Seq2Grf [OPTIONS] <snt>
```

ce programme construit un fichier `.grf` qui correspond aux séquences contenues dans le fichier `<snt>`.

OPTIONS :

- `-a ALPH/--alphabet=ALPH` : le fichier alphabet à utiliser ;
- `-o XXX/--output=XXX` : le fichier graphe de sortie ;
- `-s/--only-stop` : ne considérer que les séquences séparées par `{STOP}` ;
- `-b/--beautify` : appliquer au graphe l'algorithme beautify ;
- `-n/--no_beautify` : ne pas appliquer au graphe l'algorithme beautify ; (par défaut) ;
- `--case-sensitive` : respect de la casse (par défaut) ;
- `--case-insensitive` : non respect de la casse ;
- `-w x` : nombre de jokers ;
- `-i x` : nombre d'insertions ;
- `-r x` : nombre de remplacement ;
- `-d x` : nombre de délitions ;

Construire l'automate des séquences : un unique automate qui reconnaît toutes les séquences du SNT. Les séquences doivent être délimitées par l'étiquette `{STOP}`;. Le fichier `.grf` produit est stocké dans le répertoire Graphs de l'utilisateur. Les autres fichiers, nommés `text.tfst`, `text.tind` se trouvent dans le répertoire text.

14.35 SortTxt

```
SortTxt [OPTIONS] <txt>
```

Ce programme effectue un tri lexicographique des lignes du fichier `<txt>`. `<txt>` doit représenter le chemin d'accès complet au fichier à trier.

OPTIONS :

- `-n/--no_duplicates` : supprime les doublons (par défaut) ;
- `-d/--duplicates` : conserve les doublons ;
- `-r/--reverse` : trie en ordre décroissant ;
- `-o XXX/--sort_order=XXX` : trie en utilisant l'ordre alphabétique défini par le fichier `XXX`. Si ce paramètre est abscent, le tri est effectué selon l'ordre des caractères Unicode ;
- `-l XXX/--line_info=XXX` : sauvegarde le nombre de lignes du fichier résultat dans le fichier `XXX` ;
- `-t/--thai` : option à utiliser pour trier un texte Thai.
- `-f/--factorize_inflectional_codes` : transforme les deux entrées XXX,YYY.ZZZ :A et XXX,YYY.ZZZ :B en l'entrée unique XXX,YYY.ZZZ :A :B

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.

14.36 Stats

```
Stats [OPTIONS] <ind>
```

Ce programme calcule des statistiques à partir du fichier d'index de concordances `<ind>`.

OPTIONS :

- `-m MODE/--mode=MODE` : spécifie la sortie à produire :
  - `0` = séquence reconnue avec contexte gauche et droit + nombre d'occurrences ;
  - `1` = cooccurrences + nombre d'occurrences ;
  - `2` = cooccurrences + nombre d'occurrences + z-score.
- `-a ALPH/--alphabet=ALPH` : fichier alphabet à utiliser ;
- `-o OUT/--output=OUT` : fichier de sortie ;

- `-l N/--left=N` : longueur du contexte gauche en tokens ;
- `-r N/--right=N` : longueur du contexte droit en tokens ;
- `-c N/--case=N` : traitement de la casse : `0` = non respect de la casse, `1` = respect de la casse (par défaut).

14.37 Table2Grf

```
Table2Grf [OPTIONS] <table>
```

Ce programme génère automatiquemient des graphes à partir de la table de lexique- grammaire `<table>` et d'un graphe patron

OPTIONS :

- `-r GRF/--reference_graph=GRF` : nom du graphe patron ;
- `-o OUT/--output=OUT` : nom du graphe résultant principal ;
- `-s XXX/--subgraph_pattern=XXX` : si ce paramètre optionnel est spécifié, tous les sous-graphes produits seront nommés en fonction de ce motif. Afin d'avoir des noms non ambigus, nous vous recommandons d'inclure `@%` dans le paramètre (rappelons que `@%` sera remplacé par le numéro de ligne de l'entrée dans la table). Par exemple, si vous définissez le paramètre par le motif '`subgraph-@%.grf`', les noms de sous-graphe seront de la forme '`subgraph-0013.grf`'. Par défaut, les noms de sous-graphe ressemblent à '`result_0013.grf`', où '`result.grf`' est le graphe résultant principal.

14.38 Tagger

```
Tagger [OPTIONS] <tfst>
```

L'entrée de ce programme est l'automate du texte spécifié dans `.tfst`. Le programme applique l'algorithme de Viterbi et produit un automate linéaire. L'automate est élagué de façon probabiliste selon un modèle de Markov caché de second ordre. Si fichier tagger indiqué contient des tuples de type "cat" , le tagger élague les transitions sur la base des codes grammaticaux, syntaxiques et sémantiques (par exemple, `that.DET+Ddem` versus `that.PRO+Pdem`). Par contre si le fichier contient des tuples de type "morph", le tagger élague les transitions sur la base des codes grammaticaux, syntaxiques, sémantiques et flexionnels (`the.DET+Ddef:s` versus `the.DET+Ddef:p`). Dans le cas où, l'automate doit être développé avant d'applique le processus d'étiquetage un fichier tagset doit re indiqué avec l'option `-t` ci-dessous.

OPTIONS :

- `-a ALPH/--alphabet=ALPH` : fichier alphabet.

- `-o OUT/--output=OUT` : automate du texte en sortie.
- `-t TAGSET/--tagset=TAGSET` : nom du fichier tagset.
- `-d DATA/--data=DATA` : un fichier de donné tagger .bin qui contient le nombre d'occurences d' unigramme, de bigrammes et de trigrammes afin de calculer des probabilités. ce fichier est fournit avec le programme `TrainingTagger` (voir section 15.10.2).

14.39 TagsetNormTfst

```
TagsetNormTfst [OPTIONS] <tfst>
```

Ce programme normalise l'automate de texte `.tfst` selon un fichier de jeu d'étiquettes, en supprimmant les codes dictionnaire non déclarés et les entrées lexicales incohérentes. Les caractéristiques flexionnelles ne pas sont factorisées afin que `{rouge,.A:fs:ms}` soit divisé en deux étiquettes `{rouge,.A:fs}` et `{rouge,.A:ms}`.

OPTIONS :

- `-o OUT/--output=OUT` : automate de texte résultant. Par défaut, l'automate du texte donné en entrée est modifié ;
- `-t TAGSET/--tagset=TAGSET` : nom du fichier de description du jeu d'étiquettes

14.40 TEI2Txt

```
TEI2Txt [OPTIONS] <xml>
```

Produit un fichier de texte brut à partir du fichier TEI `<xml>`.

OPTIONS :

- `-o TXT/--output=TXT` : nom du fichier de texte de sortie. Par défaut, le fichier de sortie porte le même nom que celui d'entrée, remplaçant `.xml` by `.txt`.

14.41 Tfst2Grf

```
Tfst2Grf [OPTIONS] <tfst>
```

Ce programme extrait un automate de phrase en format `.grf` format à partir d'un automate du texte donné.

OPTIONS :

- `-s N/--sentence=N` : le nombre de phrases à extraire ;

- `-o XXX/--output=XXX` : motif utilisé pour nommer le fichier de sortie `XXX.grf`, `XXX.txt` et `XXX.tok` (defaut=`cursentence`);
- `-f FONT/--font=FONT` : définit la police à utiliser en sortie `.grf` (default=`Times new Roman`);
- `-z N/--fontsize=N` : définit la taille de police (defaut=10).

Le programme produit les fichiers suivants et les enregistre dans le répertoire du texte :

- `cursentence.grf` : graphe représentant l'automate de la phrase ;
- `cursentence.txt` : fichier texte contenant la phrase ;
- `cursentence.tok` : fichier texte contenant le nombre de token qui compose la phrase ;

14.42 Tfst2Unambig

```
Tfst2Unambig [OPTIONS] <tfst>
```

Ce programme prend un automate de texte `.tfst` et produit le fichier texte équivalent si celui-ci est linéaire (i.e. sans ambiguïté). Voir section 7.6, page 200.

OPTIONS :

- `-o TXT/--out=TXT` : fichier de sortie.

14.43 Tokenize

```
Tokenize [OPTIONS] <txt>
```

Ce programme découpe le texte en unités lexicales. `<txt>` le chemin d'accès complet au fichier texte, sans omettre l'extension `.snt` extension.

OPTIONS :

- `-a ALPH/--alphabet=ALPH` : alphabet file ;
- `-c/--char_by_char` : indique que le programme est appliqué caractère par caractère à l'exception du délimiteur de phrase `{S}`, du marqueur `{STOP}` et d'étiquettes lexicales comme `{today,.ADV}` qui sont considérées comme des unités simples ;
- `-w/--word_by_word` : Avec cette option, le programme considère qu'une unité est soit une séquence de lettres (ces lettres sont définies dans le fichier `alphabet`), ou un caractère qui n'est pas une lettre, ou le délimiteur de phrase `{S}`, ou une étiquette lexicale comme `{aujourd'hui,.ADV}`. C'est le mode par défaut.

- `-t TOKENS/--tokens=TOKENS` : désigne un fichier a `tokens.txt` à charger et à modifier, au lieu d'en créer un nouveau à partir de zéro.

Options d'offsets :

- `input_offsets` : fichier offsets d'entrée ;
- `output_offsets` : fichier offsets à produire ;

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 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.

14.44 TrainingTagger

```
TrainingTagger [OPTIONS] <txt>
```

Ce programme génère automatiquement deux fichiers de données Tagger à partir d'un corpus étiqueté. Ils sont utilisés par le programme `Tagger` afin de calculer les probabilités et linéariser l'automate texte. Le fichier corpus étiqueté doit suivre le format décrit à la section 15.10.1. Ces fichiers contiennent des tuples (unigrammes, bigrammes et trigrammes), formées par des balises et des mots. Dans le premier fichier de données, les étiquettes sont de type "cat" (i.e. des codes grammaticaux, syntaxiques et sémantiques). Dans le second fichier de données, les étiquettes sont de type "morph" (i.e. des codes grammaticaux, syntaxiques, sémantiques et flexionnels).

OPTIONS :

- `-a/--all` : indique que le programme doit produire tous les fichiers de données (par défaut) ;
- `-c/--cat` : indique que le programme ne doit produire que les fichiers de données avec "cat" ;
- `-m/--morph` : indique que le programme ne doit produire que les fichiers de données avec "morph" ;
- `-n/--no_binaries` : indique que le programme ne doit pas compresser les fichiers de données en fichiers `.bin`, seulement dans ce cas les fichiers de données `.dic` sont genérés ;
- `-b/--binaries` : indique que le programme doit compresser les fichiers de données en fichiers `.bin` files (par défaut) ;
- `-o XXX/--output=XXX` : motif utilisé pour nommer les fichiers de sortie du tagueur `XXX_data_cat.bin` ; et `XXX_data_morph.bin` (par défaut : nom de fichier sans extension corpus de textes) ;
- `-s/--semitic` : indique que l'algoritme de compression sémitique doit être utilisé ;

14.45 Txt2Tfst

`Txt2Tfst [OPTIONS] <txt>` Ce programme construit l'automate du texte. Le paramètre `<txt>` doit représenter le chemin d'accès complet au fichier texte, sans omettre l'extension `.snt`

OPTIONS :

- `-a ALPH/--alphabet=ALPH` : fichier alphabet ;
- `-c/--clean` : indique que la règle de conservation des meilleurs chemins (voir section 7.2.4) doit être utilisée ;
- `-n XXX/--normalization_grammar=XXX` : nom de la grammaire de normalisation qui doit être appliquée à l'automate de texte ;
- `-t TAGSET/--tagset=TAGSET` : fichier de jeu d'étiquete Elag pour la normalisation des entrées du dictionnaire ;
- `-K/--korean` : indique à `Txt2Tfst` qu'il traite du coréen.

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.tfst` qui est sauvegardé dans le répertoire du texte. Un autre fichier `text.tind` est aussi produit.

NOTE : Ce programme essaye également d'utiliser le fichier `tags.ind` s'il existe (voir section 15.7.4).

14.46 Uncompress

```
Uncompress [OPTIONS] <bin>
```

Ce programme décompresse un dictionnaire `.bin` en un fichier texte `.dic`.

OPTIONS :

- `-o OUT/--output=OUT` : nom du fichier de sortie optionnel (par défaut : `file.bin` > `file.dic`).

14.47 Untokenize

```
Untokenize [OPTIONS] <txt>
```

Untokenize et reconstruit le texte orgininal. La liste des token est stockée dans le fichier `tokens.txt` et le texte codé dans `text.cod`. Le fichier `enter.pos` contient la position en tokens de tous les retours à la ligne Ces fichiers se trouvent dans le répertoire XXX_snt où XXX est sans son extension <txt>.

OPTIONS :

- `-d X/--sntdir=X` : utilise le répertoire X au lieu du répertoire texte ; remarquez que X doit se terminer par un antislash
- `-n N/--number_token=N` : ajoute le numéro de token chaque N token ;
- `-r N/--range=N` : émet seulement les tokens du numéro N à la fin ;
- `-r N,M/--range=N,M` : émet seulement les tokens du numéro N à M.

14.48 UnitexTool

```
UnitexTool <utilities>
```

Ce programme permet d'exécuter tous les programmes externes d'Unitex. Avec lui, on peut enchaîner une séquence de programmes afin qu'ils soient exécutés dans un même processus, ce qui accélère le traitement (voir aussi le chapitre 13). Cela se fait en invoquant des commandes imbriquées entre accolades comme ceci :

```
UnitexTool { SelectOutput [OPTIONS] }
               { cmd #1+args }
               { cmd #2+args }
               etc.
```

Par exemple, si vous souhaitez faire un Locate et construire la concordance, vous pouvez utiliser la commande suivante :

```
UnitexTool { Locate "-tD:\My Unitex\English\Corpus\ivanhoe.snt"
"D:\My Unitex\English\regexp.fst2"
"-aD:\My Unitex\English\Alphabet.txt" -L -I -n200
"--morpho=D:\Unitex2.0\English\Dela\dela-en-public.bin" -b -Y }
{ Concord "D:\My Unitex\English\Corpus\ivanhoe_snt\concord.ind"
"-fCourier new" -s12 -l40 -r55 --CL --html
"-aD:\My Unitex\English\Alphabet_sort.txt" }
```

OPTIONS :

- `-o [on/off]/--output=[on/off]` : activer (on) ou désactiver (off) la sortie standard
- `-e [on/off]/--error=[on/off]` : activer (on) ou désactiver (off) la sortie erreur standard

Par exemple :

```
UnitexTool { SelectOutput -o off -e off } { Normalize
Unitex\English\Corpus\ivanhoe.txt }
```

14.49 UnitexToolLogger

```
UnitexToolLogger <utilities>
```

Ce programme est un sur-ensemble d'UnitexTool. Il permet d'exécuter à nouveau un fichier de log .ulp. Il peut également enregistrer une session d'UnitexTool en cours d'exécution et créer un fichier de log .ulp. Si UnitexToolLogger est utilisé comme UnitexTool (avec uniquement les paramètres contenant des lignes de commande pour des programmes Unitex externes), et qu'un fichier contenant un chemin et nommé unitex_logging_parameters_count.txt est présent dans le répertoire courant, alors un fichier de log .ulp pour la session en cours sera créé. Le fichier .ulp est un fichier zip comprimé (compatible avec unzip), qui peut être utile pour le débogage.

```
UnitexToolLogger RunLog [OPTIONS] <ulp>
```

OPTIONS after RunLog :

- `-m/--quiet` : n'émet pas de messages lors de l'exécution ;
- `-v/--verbose` : émet des messages lors de l'exécution ;
- `-d DIR/--rundir=DIR` : chemin où le fichier log est exécuté ;
- `-r newfile.ulp/--result=newfile.ulp` : nom du fichier ulp résultat créé ;
- `-c/--clean` : supprime le fichier de travail après l'exécution ;
- `-k/--keep` : conserve le fichier de travail après l'exécution ;
- `-s file.txt/--summary=file.txt` : fichier ** avec comparaison de log
- `-e file.txt/--summary-error=file.txt` : fichier de synthèse avec comparaison des erreurs ;
- `-b/--no-benchmark` : ne pas enregistrer le temps d'exécution dans les fichiers log ;
- `-n/--cleanlog` : supprime le résultat ulp après exécution ;
- `-l/--keeplog` : garde le résultat ulp après exécution ;
- `-o NameTool/--tool=NameTool` : lance seulement les log pour `NameTool` ;
- `-i N/--increment=N` : incrémenter le nom de fichier <ulp> de 0 à N ;
- `-t N/--thread=N` : créer N thread ;
- `-a N/--random=N` : choisir N fois un fichier log aléatoire dans la liste (dans chaque thread) ;
- `-f N/--break-after=N` : l'utilisateur annule après N exécutions (avec seulement un seul thread) ;
- `-u PATH/--unfound-location=PATH` : prend le dictionnaire et le FST2 à partir de PATH s'il est absent du fichier log ;

Une autre utilisation d'UnitexToolLogger est d'utiliser l'option MzRepairUlp pour réparer un fichier ulp abîmé (souvent, un log de crash) :

```
UnitexToolLogger MzRepairUlp [OPTIONS] <ulpfile>
```

OPTIONS après MzRepairUlp :

- `-t X/--temp=X` : utilise X comme nom de fichier temporaire (<ulpfile>.build par défaut) ;
- `-o X/--output=X` : utilise X comme nom de fichier .ulp (<ulpfile>.repair par défaut) ;
- `-m/--quiet` : n'émet pas de message lors de l'exécution ; ;

- `-v/--verbose` : émet un message lors de l'exécution ;

Une autre utilisation de UnitexToolLogger est d'utiliser l'option CreateLog (avec des accolades) pour créer un fichier log d'exécutions de programmes Unitex, comme :

```
UnitexToolLogger { CreateLog [OPTIONS] } cmd args

UnitexToolLogger { CreateLog [OPTIONS] } { cmd #1+args } { cmd #2+args } etc.
```

Par exemple,

```
UnitexToolLogger { CreateLog --log_file=my_run_normalize.ulp }
             Normalize "C:\My Unitex\French\Corpus\80jours.txt"


UnitexToolLogger { CreateLog --directory=c:\logs }
                 { Compress c:\dela\mydela.dic }
                 { CheckDic --delaf c:\dela\mydela.inf }
```

OPTIONS après CreateLog :

- `-g/--no_create_log` : ne pas créer de fichier log. Incompatible avec toutes les autres options ;
- `-p XXX/--param_file=XXX` : charge un fichier de paramètres comme unitex_logging_parameters.txt. Incompatible avec toutes les autres options ;
- `-d XXX/--directory=XXX` : Emplacement du répertoire où le fichier log est créé ;
- `-l XXX/--log_file=XXX` : nom du fichier log à créer ;
- `-i/--store_input_file` : enregistre le fichier d'entrée dans log (par défaut) ;
- `-n/--no_store_input_file` : n'enregistre pas le fichier d'entrée dans log (empêche de relancer le fichier log) ;
- `-o/--store_output_file` : enregistre le fichier de sortie dans log ;
- `-u/--no_store_output_file` : n'enregistre pas le fichier de sortie dans log (par défaut) ;
- `-s/--store_list_input_file` : enregistre la liste de fichiers d'entrée dans log (par défaut) ;
- `-t/--no_store_list_input_file` : n'enregistre pas la liste de fichiers d'entrée dans log ;
- `-r/--store_list_output_file` : enregistre la liste de fichiers de sortie dans log (par défaut) ;

- `-f/--no_store_list_output_file` : n'enregistre pas la liste de fichiers de sortie dans log.

```
UnitexToolLogger { SelectOutput [OPTIONS] }
                 { cmd #1+args }
                 { cmd #2+args }
                 etc.
```

OPTIONS après SelectOutput :

- `-o [on/off]/--output=[on/off]` : activer (on) ou désactiver (off) la sortie standard
- `-e [on/off]/--error=[on/off]` : activer (on) ou désactiver (off) la sortie erreur standard

Par exemple :

```
UnitexToolLogger { SelectOutput -o off -e off } { Normalize
Unitex\English\Corpus\ivanhoe.txt }
```

14.50 Unxmlize

Ce programme supprime tous les tags xml d'un fichier .xml ou .html donné pour produire un fichier texte traitable par Unitex. `Unxmlize [OPTIONS] <file>`

OPTIONS :

- `-o TXT/--output=TXT` : fichier de sortie. Par défaut, foo.xml => foo.txt
- `--output_offsets=XXX` : spécifie le fichier offset à produire
- `--PRLG=XXX` : extrait dans le fichier XXX des informations utilisées dans le projet PRLG du grec ancien (exige `--output_offsets`)
- `-t/--html` : considère le fichier comme un fichier html (ne tient pas compte de l'extension)
- `-x/--xml` : considère le fichier comme un fichier xml (ne tient pas compte de l'extension)
- `-l/--tolerate` : essayez tolérer des malformations de balisage
- `--comments=IGNORE` : chaque commentaire est supprimé (par défaut)

- `--comments=SPACE` : chaque commentaire est remplacé par un simple espace
- `--scripts=IGNORE` : chaque script block is removed
- `--scripts=SPACE` : chaque commentaire est remplacé par un simple espace (par défaut pour .html)

Note : par défaut, balises de script sont traitées comme des balises normales (par défaut pour .xml).

- `--normal_tags=IGNORE` : chaque tag différent est supprimé (par défaut pour .xml)
- `--normal_tags=SPACE` : chaque tag différent est remplacé est remplacé par un unique espace (par défaut pour .html)

14.51 XMLizer

```
XMLizer [OPTIONS] <txt>
```

Ce programme prend un fichier texte brut `<txt>` et produit le fichier équivalent au format TEI ou XML. La différence entre TEI et XML est que les fichier TEI contiennent une en-tête de type TEI.

OPTIONS :

- `-x/--xml` : produit un fichier a XML ;
- `-t/--tei` : produit un fichier TEI (par défaut) ;
- `-n XXX/--normalization=XXX` : désigne le fichier de règles de normalisation à utiliser (voir section 15.13.6) ;
- `-o OUT/--output=OUT` : nom optionnel du de fichier de sortie (par défaut : `file.txt > file.xml`);
- `-a ALPH/--alphabet=ALPH` : fichier alphabet ;
- `-s SEG/--segmentation_grammar=SEG` : grammaire de délimitation de phrase à utiliser. Cette grammaire devrait ressembler à la grammaire `Sentence.grf` utilisée lors du prétraitement d'un corpus, mais elle peut comporter l'étiquette spéciale `{P}` pour indiquuer les limites de paragraphe.

Chapitre 15

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.

15.1 Codage Unicode

Par défaut, les fichiers textes manipulés par Unitex doivent être en Unicode Little-Endian. Unitex accepte aussi des fichiers Unicode Big-Endian ou UTF-8. 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 Little-Endian, Big-Endian or UTF-8 commence par le caractère spécial (Unicode Byte Order Mark - BOM) de valeur hexadécimale `FF FE` (Little-Endian), `FE FF` (Big-Endian) ou `EF BB BF` (UTF-8). Parce que UTF-8 n'a pas d'ordre d'octet, l'ajout d'un BOM UTF-8 est optionnel ; pour UTF-16 c'est obligatoire. Les symboles de saut de ligne doivent être codés par les deux caractères `0D 00` et `0A 00` (Little-Endian), `00 0D` et `00 0A` (Big-Endian), ou `0D` and `0A` (UTF-8).

Considérons le texte suivant :

```
Unitex¶
β-version¶
```

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

BOM header	U	n	i	t	e	x	¶	β
FF FE	55 00	6E 00	69 00	74 00	65 00	78 00	0D 00 0A 00	B2 03
-	v	e	r	s	i	o	n	¶
2D 00	76 00	65 00	72 00	73 00	69 00	6F 00	6E 00	0D 00 0A 00

TABLE 15.1 – Représentation hexadécimale d'un texte Unicode Little-Endian

Voici sa représentation en Unicode Big-Endian :

BOM header	U	n	i	t	e	x	¶	β
FE FF	00 55	00 6E	00 69	00 74	00 65	00 78	00 0D 00 0A	03 B2
-	v	e	r	s	i	o	n	¶
00 2D	00 76	00 65	00 72	00 73	00 69	00 6F	00 6E	00 0D 00 0A

TABLE 15.2 – Représentation hexadécimale d'un texte Unicode Big-Endian

Voici sa représentation Unicode en UTF-8 :

BOM header	U	n	i	t	e	x	¶	β
EF BB BF	55	6E	69	74	65	78	0D 0A	CE B2
-	v	e	r	s	i	o	n	¶
2D	76	65	72	73	69	6F	6E	0D 0A

TABLE 15.3 – Représentation hexadécimale d'un texte Unicode UTF-8

En Unicode Little-Endian, les octets de poids fort et de poids faible ont été inversés, ce qui explique que le caractère d'en-tête soit codé par `FF FE` au lieu de `FE FF`, idem pour `00 0D` et `00 0A` qui sont devenus respectivement `0D 00` and `0A 00`.

15.2 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 *alphabet*, et le second sous celui *alphabet de tri*.

15.2.1 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 :

- `#`XY : un dièse suivi de 2 caractères X and 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 ;
- `Aa` : 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.
- `ᐁ` : 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. For example, `é`, qui peut avoir comme majuscule soit `E` ou `É`. Pour exprimer cela, il suffit d'utiliser plusieurs lignes. L'inverse est également vrai : à une majuscule peuvent correspondre plusieurs minuscules. Ainsi, `E` peut être la majuscule de `e`, `é`, `è`, `ë` ou `ê`. Voici l'extrait du fichier alphabet du français qui définit les différentes lettres `e` :

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

15.2.2 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 `eéèëê` le mot `ébahi` sera considéré comme plus petit que `estuaire`, lui-même plus petit que `été`. Comme les lettres qui suivent `e` et `é` permettaient 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 `eéèëê`, 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.

15.3 Graphes

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

15.3.1 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 le 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'entête.

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 15.4 donne le codage des deux séquences spéciales qui ne sont pas codées telles quelles dans les fichiers `.grf` :

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 and *Y* représentent les coordonnées de la boîte en pixels. La figure 15.1 montre comment ces coordonnées sont interprétées par Unitex.

Séquence dans l'éditeur de graphe	Séquence dans le fichier `.grf`
"	\"
\"	\\\"

TABLE 15.4 – Codage des séquences spéciales

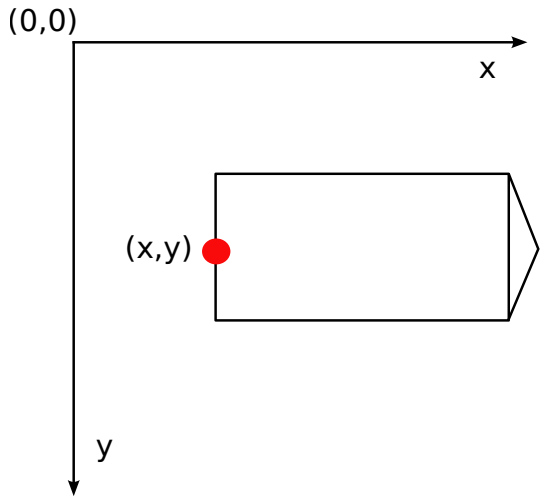


FIGURE 15.1 – Interprétation des coordonnées des boîtes

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.

15.3.2 Format .fst2

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

```
0000000002¶
-1 NP¶
: 1 1 ¶
: 2 2 -2 2 ¶
: 3 3 ¶
t ¶
f ¶
-2 Adj¶
```

```
: 6 1 5 1 4 1 ¶
t ¶
f ¶
%<E>¶
%the/DET¶
%<A>/ADJ¶
%<N>¶
%nice¶
@pretty¶
%small¶
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 NP` 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 et d'un retour à la ligne.

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 recherché 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 : `the/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.

15.4 Textes

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

15.4.1 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` and `000A`.

15.4.2 Fichiers .snt

`.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}`).

15.4.3 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.

15.4.4 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 contenues 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 unitée 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.

15.4.5 Fichier 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.

15.4.6 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.

15.5 Automate du texte

15.5.1 Fichier text.tfst

Le fichier `text.tfst` représente l'automate du texte. C'est un fichier texte qui commence par une ligne comportant dix chiffres qui indiquent le nombre de phrases contenues dans l'automate. Ensuite, pour chaque phrase, on dispose de l'en-tête suivante :

- `$XXX¶` : `XXX` = numéro de la phrase ;
- `foo foo foo...¶` : texte de la phrase ;
- `a/b c/d e/f g/h...¶` : pour chaque token de la phrase, il y a une paire `x/y` : `x` est l'index du token dans le fichier `tokens.txt`, `y` est sa longueur en caractères ;
- `X_Y¶` : `X` est l'offset du premier token de la phrase, en tokens depuis le début du texte ; `Y` est identique mais l'offset représente le nombre de caractères.

Ensuite, tous les états de l'automate sont codés, un par ligne. Si l'état est final, la ligne commence par `t`. Sinon, elle commence par `:`. Toutes les transitions sont écrites sous la forme de paires `x y`, `x` étant le nombre de tag, `y` étant le nombre d'états de destination. Remarquons que contrairement au format `.fst2`, les lignes doivent finir par un espace. La dernière ligne de la liste d'états contient `f`.

Enfin, tous les les tags sont codés. Par convention, le premier tag est toujours épsilon : `@<E>¶`

`.¶`

D'autres étiquettes doivent être soit des unités lexicales ou des entrées au format DELAF entre accolades. Elles sont codées comme suit :

`@STD¶`
`@`*content*`¶`
`@`*a*`.`*b*`.`*c*`-`*x*`.`*y*`.`*z*`¶`
`.¶`

contenu est le contenu du tag. Les informations *a.b.c-x.y.z* décrivent la zone du texte couverte par le tag :

- *a* : offset de début en tokens depuis le début de la phrase ;

- b : offset de début en caractères depuis le début du premier ; token du tag ;
- c : offset de début en lettres logiques depuis le premier caractère du tag. Ces informations sont utiles pour le coréen, parce qu'un tag représent une séquence de caractères Jamo qui apparaissent à l'intérieur d'un Hangul. L'offset en caractères n'est donc pas assez précis ;
- x : offset de fin en tokens depuis le début de la phrase ;
- y : offset de fin en caractères depuis le début du dernier token du tag ;
- z : de fin en lettres logiques depuis le dernier caractère du the tag. Dans des automates de phrase coréen, des formes de surface vides peuvent correspondre à des mots vides du texte. Dans ces cas, z a la valeur -1.

La définition de tag se termine par une ligne qui contient `f`.

Exemple : Voici le fichier correspondant au texte *He is drinking orange juice.*

```
0000000001¶
$1¶
He is drinking orange juice. ¶
0/2 1/1 2/2 1/1 3/8 1/1 4/6 1/1 5/5 6/1 1/1¶
0_0¶
: 2 1 1 1¶
: 4 2 3 2¶
: 7 3 6 3 5 3¶
: 10 5 9 4 8 4¶
: 12 5 11 5¶
: 13 6¶
t¶
f¶
@<E>¶
.¶
@STD¶
@{He,he.N:s:p}¶
@0.0.0-0.1.0¶
.¶
@STD¶
@{He,he.PRO+Nomin:3ms}¶
@0.0.0-0.1.0¶
.¶
@STD¶
@{is,be.V:P3s}¶
@2.0.0-2.1.0¶
.¶
```

```
@STD¶
@{is,i.N:p}¶
@2.0.0-2.1.0¶
.¶
@STD¶
@{drinking,drinking.A}¶
@4.0.0-4.7.0¶
.¶
@STD¶
@{drinking,drinking.N:s}¶
@4.0.0-4.7.0¶
.¶
@STD¶
@{drinking,drink.V:G}¶
@4.0.0-4.7.0¶
.¶
@STD¶
@{orange,orange.A}¶
@6.0.0-6.5.0¶
.¶
@STD¶
@{orange,orange.N:s}¶
@6.0.0-6.5.0¶
.¶
@STD¶
@{orange juice,orange juice.N+XN+z1:s}¶
@6.0.0-8.4.0¶
.¶
@STD¶
@{juice,juice.N+Conc:s}¶
@8.0.0-8.4.0¶
.¶
@STD¶
@{juice,juice.V:W:P1s:P2s:P1p:P2p:P3p}¶
@8.0.0-8.4.0¶
.¶
@STD¶
@.¶
@9.0.0-9.0.0¶
.¶
f¶
```

15.5.2 Fichier text.tind

Le fichier `text.tind` utilisé pour sauter à l'octet d'offset correct dans le fichier `text.tfst` quand on veut charger une phrase donnée. C'est un fichier binaire qui contient $4 \times N$ octets,où N est le nombre de phrases. Il donne l'offset de départ de chaque phrase sous la forme d'une suite de 4 octets en little-endian.

15.5.3 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`.

NOTE : les sorties des boîtes sont utilisées pour coder les offsets, tels que définis dans `.tfst`. Les offsets sont séparés par des espaces. Voici, par exemple, quelques lignes qui representent la première phrase d'*Ivanhoe* :

```
"Ivanhoe/0 0 0 0 6 0" 100 200 2 3 4 ¶
"{by,by.PART}/2 0 0 2 1 0" 220 150 2 5 6 ¶
"{by,by.PREP}/2 0 0 2 1 0" 220 50 2 5 6 ¶
"{Sir,sir.N+Hum:s}/4 0 0 4 2 0" 310 200 1 7¶
```

15.5.4 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.

un tel graphe contient des offsets dans les sorties des boîtes du graphe (voir note section 15.5.3).

15.5.5 Fichier cursentence.txt

Lors de l'extraction de l'automate phrase, le texte de la phrase est enregistré dans le fichier appelé `cursentence.txt`. Ce fichier est utilisé par Unitex pour afficher le texte de la phrase sous l'automate. Ce fichier contient le texte de la phrase, suivie d'un saut de ligne.

15.5.6 The cursentence.tok file

Lors de l'extraction de l'automate phrase, les numéros de tokens qui composent la phrase sont enregistrés dans un fichier nommé `cursentence.tok`. Ce fichier contient une ligne par token, chaque ligne étant composée de 2 entiers `x y` : `x` est le numéro de token, `y` est sa longueur en caractères.

Voici le contenu de ce fichier pour la première phrase d'*Ivanhoe* :

```
0 7¶        Ivanhoe
1 1¶        ␣
2 2¶        by
```

```
1 1¶          ␣
3 3¶          Sir
1 1¶          ␣
4 6¶          Walter
1 1¶          ␣
5 5¶          Scott
1 1¶          ␣
```

15.5.7 Fichiers tfst_tags_by_freq.txt et tfst_tags_by_alph.txt

Ces fichiers contiennent tous les tags qui apparaissent dans l'automate du texte classés par fréquence et par ordre alphabétique.

15.6 Concordances

15.6.1 Fichier concord.ind

Le fichier `concord.ind` est l'index des occurrences trouvées par les programmes `Locate` ou `LocateTfst` 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¶
59.0.0 63.3.0 the[ADJ= greater] part¶
67.0.0 71.4.0 the beautiful hills¶
87.0.0 91.3.0 the pleasant town¶
123.0.0 127.4.0 the noble seats¶
157.0.0 161.5.0 the fabulous Dragon¶
189.0.0 193.3.0 the Civil Wars¶
455.0.0 459.11.0 the feeble interference¶
463.0.0 467.6.0 the English Council¶
566.0.0 570.10.0 the national convulsions¶
590.0.0 594.5.0 the inferior gentry¶
626.0.0 630.11.0 the English constitution¶
696.0.0 700.4.0 the petty kings¶
813.0.0 817.5.0 the certain hazard¶
896.0.0 900.5.0 the great Barons¶
938.0.0 942.3.0 the very edge¶
```

La première ligne indique dans quel mode de transduction la concordance a été calculée. Les trois 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 correspondent aux offsets définis dans le fichier tag `.tfst` (voir 15.5.1).

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.

15.6.2 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.

15.6.3 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 :

```
<a href="X Y Z">
```

`X` et `Y` représentent les positions de début et de fin de l'occurrence en caractères dans le fichier `name_of_text.snt`. `Z` représente le numéro de la phrase dans laquelle apparaît cette occurrence.

Tous les espaces sont codés comme des espaces insécables (` ` in 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 ://unitexgramlab.org).

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>¶
<table border="0" cellpadding="0" width="100%"
       style="font-family: 'Arial Unicode MS'; font-size: 12">¶
<font face="Courier new" size=3>¶
on, there <a href="116 124 2">extended</a> i <br>¶
 extended <a href="125 127 2">in</a> ancient <br>¶
 Scott {S}<a href="32 34 2">IN</a> THAT PL <br>¶
STRICT of <a href="61 66 2">merry</a> Engl <br>¶
S}IN THAT <a href="40 48 2">PLEASANT</a> D <br>¶
 which is <a href="84 91 2">watered</a> by <br>¶
</font>¶
</td></table></body>¶
</html>¶
```

La figure 15.2 montre la page correspondant au fichier ci-dessus.

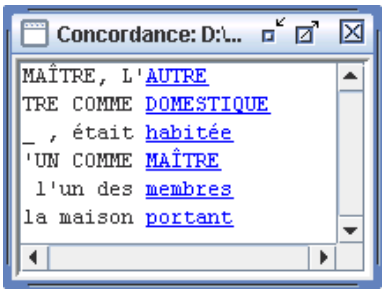


FIGURE 15.2 – Exemple de concordance

15.6.4 Fichier diff.html

Le fichier `diff.html` est une page `HTML` qui montre les différences entre deux concordances. Ce fichier est encodé en UTF-8. Voici un exemple de fichier (des retours à la ligne ont été introduits pour la mise en page) :

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
```

```
<style type="text/css">
a.blue {color:blue; text-decoration:underline;}
a.red {color:red; text-decoration:underline;}
a.green {color:green; text-decoration:underline;}
</style>
</head>
<body>
<h4>
<font color="blue">Blue:</font> identical sequences<br>
<font color="red">Red:</font> similar but different sequences<br>
<font color="green">Green:</font> sequences that occur in only
one of the two concordances<br>
<table border="1" cellpadding="0" style="font-family: Courier new;
font-size: 12">
<tr><td width="450"><font color="blue">ed in ancient times
<u>a large forest</u>, covering the greater par</font></td>
<td width="450"><font color="blue">ed in ancient times
<u>a largeforest</u>, covering the greater par</font></td>
</tr>
<tr><td width="450"><font color="green">ge forest, covering
<u>the greater part</u> of the beautiful hills </font>
</td>
<td width="450"><font color="green"></font></td>
</tr>
</table>
</body>
</html>
```

15.7 Dictionnaires du texte

Le programme `Dico` produit plusieurs fichiers qui représentent les dictionnaires.

15.7.1 dlf et dlc

`dlf` et `dlc` sont des dictionnaires de mots simples et composés au format DELAF format (voir section 3.1.1).

15.7.2 err

Ce fichier contient les mots inconnus, un par ligne.

15.7.3 tags_err

Ce fichier contient les mots inconnus, un par ligne. La différence avec le fichier `err` est que dans celui-ci les mots simples reconnus dans le fichier `tags.ind` n'apparaissent pas.

15.7.4 tags.ind

Ce fichier a le même format que `concord.ind` il s'obtient en mode MERGE ou REPLACE mais son en-tête est `#T`. Remarquons que les sorties ne commence pas par un slash.

15.8 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.

15.8.1 Fichier .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 codé 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 50106ème , octet sera représenté par la séquence hexadécimale `004100C3BA`.

Par convention, le premier état de l'automate est l'état initial.

15.8.2 Fichier.inf

Un fichier `.inf` est un fichier texte décrivant les formes comprimées associées à un fichier `.bin`. Voici un exemple de fichier `.inf` file :

```
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 fléchies 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 :

```
0-1.N:p
```

qui est la sortie pour `battle-axes,battle-axe.N:p`

Cela permet de conserver une certaine visibilité dans le fichier `.inf` lorsque le dictionnaire contient des mots composés.

Lorsque 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.

15.8.3 Fichier information sur un dictionnaire

Dans le cadre "Apply lexical resources", il est possible d'obtenir quelques informations sur un dictionnaire par click droit. Ces informations sont associées aux dictionnaires `biniou.bin` ou `biniou.fst2` à l'aide d'un texte brut nommé `biniou.txt`, situé dans le même répertoire.

15.8.4 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 quatre 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: unexpected end of line
garden,N:s
```

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
abracadabra,INTJ
supercalifragilisticexpialidocious,.INTJ
damned,. INTJ
Paul,.N+Hum+Hum
eat,.V:W:P1s:Ps:P1p:P2p:P3p
```

on obtient le fichier `CHECK_DIC.TXT` suivant :

```
Line 1: unprotected comma in lemma¶
1,2 et 3!,.INTJ ¶
Line 2: unexpected end of line¶
abracadabra,INTJ ¶
Line 5: duplicate semantic code¶
Paul,.N+Hum+Hum¶
Line 6: an inflectional code is a subset of another¶
eat,.V:W:P1s:Ps:P1p:P2p:P3p¶
-----------------------------------¶
-------------  Stats  -------------¶
-----------------------------------¶
File: D:\My Unitex\English\Dela\axe.dic¶
Type: DELAF¶
6 lines read¶
2 simple entries for 2 distinct lemmas¶
0 compound entry for 0 distinct lemma¶
-----------------------------------¶
----  All chars used in forms  ----¶
-----------------------------------¶
```

```
a (0061)¶
c (0063)¶
d (0064)¶
e (0065)¶
f (0066)¶
g (0067)¶
i (0069)¶
l (006C)¶
m (006D)¶
n (006E)¶
o (006F)¶
p (0070)¶
r (0072)¶
s (0073)¶
t (0074)¶
u (0075)¶
x (0078)¶
------------------------------------------------------------¶
----    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  -----¶
-----------------------------------------------------¶
```

Remarquons que les codes flexionnels de `eat` ne sont pas signalés, puisque une erreur s'est produite dans cette ligne.

15.9 Fichiers ELAG

15.9.1 Fichier tagset.de

See section 7.3.6, page 186.

15.9.2 Fichiers .lst

LES FICHIERS .LST NE SONT PAS CODÉS EN UNICODE.

Un fichier `.lst` contient une liste de noms de fichiers `.grf`. Si le nom d'un fichier n'est pas absolu, il est relatif à l'emplacement du fichier `elag.lst`. 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¶
```

15.9.3 .elg files

Les fichiers `.elg` contiennent des règles ELAG compilées. Ces fichiers sont au format `.fst2`.

15.9.4 Fichier .rul

LES FICHIERS .RUL NE SONT PAS CODÉS EN UNICODE.

Un fichier `.rul` contient 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` . Les noms de fichiers `.elg` sont entres 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>¶
```

15.10 Fichier taggeur

Cette section présente les fichiers produits et utilisés par les programmes TrainingTagger et Tagger.

15.10.1 Fichier corpus.txt

Ce fichier est utilisé par le programme TrainingTagger afin de calculer les statistiques pour le programme Tagger. Il contient des phrases où chaque mot est représenté sur une ligne séparée.

Chaque ligne représentant un mot est constituée d'un mot, simple ou composé, suivie d'une barre oblique et de l'étiquette du mot.

Cette étiquette est composée d'un code grammatical, parfois suivi d'une `'+'` et de codes syntaxiques ou sémantiques. Les codes flexionnels sont spécifiés après un `':'`. Si le mot est un composé, les mots simples qui y figurent doivent être séparés par un `'_'`. Voici un exemple d'un fichier corpus.txt :

```
The/DET+Ddef:s¶
GATT/N:s¶
had/V:I3s¶
formerly/ADV¶
a/DET+Dind:s¶
political/A¶
assessment/N:s¶
of/PREP¶
the/DET+Ddef:s¶
behavior/N:s¶
of/PREP¶
foreign_countries/N:p¶
./PONCT¶
¶
She/PRO+Nomin:3fs¶
closed/V:I3s¶
easily/ADV¶
her/DET+Poss3fs:p¶
eyes/N:p¶
when/CONJ¶
some/DET+Dadj:p¶
infractions/N:p¶
might/V:I3p¶
appear/V:W¶
justified/V:K¶
against/PREP¶
higher/A¶
interests/N:p¶
./PONCT¶
¶
```

REMARQUE : Les phrases doivent être délimitées par des lignes vides.

Le format `.txt` peut également être utilisé (voir section 15.4.1). Chaque mot du texte doit être représenté par une étiquette lexicale valide (`{aujourd'hui,.ADV}`) et les phrases sont délimitées par `{S}`. Voici l'exemple précédent dans le format `.txt` :

```
{The,.DET+Ddef:s} {GATT,.N:s} {had,.V:I3s} {formerly,.ADV}
{a,.DET+Dind:s} {political,.A} {assessment,.N:s} {of,.PREP}
{the,.DET+Ddef:s} {behavior,.N:s} {of,.PREP} {foreign countries,.N:p}
```

```
{.,.PONCT} {S} {She,.PRO+Nomin:3fs} {closed,.V:I3s} {easily,.ADV}
{her,.DET+Poss3fs:p} {eyes,.N:p} {when,.CONJ} {some,.DET+Dadj:p}
{infraction,.N:p} {might,.V:I3p} {appear,.V:W} {justified,.V:K}
{against,.PREP} {higher,.A} {interests,.N:p} {.,.PONCT} {S}
```

15.10.2 Le fichier de données du taggueur

The TrainingTagger programme genère deux fichiers de données (par défaut) utilisé par le programme Tagger afin de calculer un modèle de Markov caché d'ordre 2. Ces fichiers contiennent des tuples unigram, bigramme et trigramme extraits du corpus étiqueté corpus.txt. Les tuples sont composés soit d'une séquence de 2 ou 3 tags (pour calculer la probabilité de transition) ou d'un mot précédé par 0 ou 1 tag (pour calculer la probabilité émise). Les unités dans un tuple doivent être séparées par une tabulation. Ces tuples sont suivis par la séquence de délimiteurs ",." et ensuite un nombre entier représentant le nombre d'occurrences de ce tuple dans le corpus.

Les noms de fichiers sont suffixés par "cat" ou "morph". Dans la premier, les tuples sont composés tags de codes grammaticaux, syntaxiques et sémantiques. Dans le second, les tuples sont composés de tags de codes grammaticaux, syntaxiques et sémantiques parfois suivis par un `':'` et des codes flexionnels. Voici un exemple d'un fichier de données avec des tags de type "cat" :

```
the,.9630¶
those,.236¶
eyes,.32¶
DET+Ddef  the,.9630¶
DET+Ddem  those,.140¶
PRO+Pdem  those,.96¶
N    eyes,.32¶
DET  N,.62541¶
PREP DET  N,.25837¶
¶
```

Voici un exemple d'un fichier de données avec des tags de type "morph" :

```
the,.9630¶
those,.236¶
eyes,.32¶
DET+Ddef:s  the,.4437¶
DET+Ddef:p  the,.5193¶
DET+Ddem:p  those,.140¶
PRO+Pdem:p  those,.96¶
N:p    eyes,.32¶
DET:s  N:s,.18489¶
PREP   DET:s  N:s,.6977¶
¶
```

Une ligne spécifique est ajoutée à des fichiers de données afin de déterminer si le fichier contient des tags de type "cat" ou "morph".

Cette ligne contient `CODE FEATURES` suivie soit le nombre 0 pour "cat" tags ou 1 pour "morph".

REMARQUE : À l'étape finale, TrainingTagger comprime ces deux fichiers de données au format `.bin`.

15.11 Fichier de configuration

15.11.1 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 (l'ordre des lignes peut varier) :

```
#Unitex configuration file of 'paumier' for 'English'¶
#Fri Oct 10 15:18:06 CEST 2008¶
TEXT\ FONT\ NAME=Courier New¶
TEXT\ FONT\ STYLE=0¶
TEXT\ FONT\ SIZE=10¶
CONCORDANCE\ FONT\ NAME=Courier new¶
CONCORDANCE\ FONT\ HTML\ SIZE=12¶
INPUT\ FONT\ NAME=Times New Roman¶
INPUT\ FONT\ STYLE=0¶
INPUT\ FONT\ SIZE=10¶
OUTPUT\ FONT\ NAME=Arial Unicode MS¶
OUTPUT\ FONT\ STYLE=1¶
OUTPUT\ FONT\ SIZE=12¶
DATE=true¶
FILE\ NAME=true¶
PATH\ NAME=false¶
FRAME=true¶
RIGHT\ TO\ LEFT=false¶
BACKGROUND\ COLOR=-1¶
FOREGROUND\ COLOR=-16777216¶
AUXILIARY\ NODES\ COLOR=-3289651¶
COMMENT\ NODES\ COLOR=-65536¶
SELECTED\ NODES\ COLOR=-16776961¶
PACKAGE\ NODES\ COLOR=-2302976¶
CONTEXT\ NODES\ COLOR=-16711936¶
CHAR\ BY\ CHAR=false¶
ANTIALIASING=false¶
HTML\ VIEWER=¶
MAX\ TEXT\ FILE\ SIZE=2097152¶
```

```
ICON\ BAR\ POSITION=West¶
PACKAGE\ PATH=D\:\\repository¶
MORPHOLOGICAL\ DICTIONARY=D\:\\MyUnitex\\English\\Dela\\zz.bin¶
MORPHOLOGICAL\ NODES\ COLOR=-3911728¶
MORPHOLOGICAL\ USE\ OF\ SPACE=false¶
```

Les deux premières lignes sont des lignes de commentaires. Les trois lignes suivantes 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.

The `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 15.5 décrit ces correspondances.

Paramètres dans le fichier `Config` file	Paramètres dans un fichier `.grf` file
DATE	DDATE
FILE NAME	DFILE
PATH NAME	DDIR
FRAME	DFRAME
RIGHT TO LEFT	DRIG
BACKGROUND COLOR	BCOLOR
FOREGROUND COLOR	FCOLOR
AUXILIARY NODES COLOR	ACOLOR
COMMENT NODES COLOR	SCOLOR
SELECTED NODES COLOR	CCOLOR

TABLE 15.5 – Signification des paramètres

Le paramètre `PACKAGE NODES` définit la couleur des appels à des sous-graphes du répertoire de dépôt.

Le paramètre `CONTEXT NODES` définit la couleur des boîtes correspondant à des débuts ou fins de contextes.

Le paramètre `CONTEXT NODES` indique si la langue courante doit être traitée en mode caractère par caractère ou non.

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.

Le paramètre `MAX TEXT FILE SIZE` n'est plus utlisé.

Le paramètre `ICON BAR POSITION` définit la position de la barre d'icônes dans les fenêtres de graphes.

Le paramètre `PACKAGE PATH` définit le répertoire de dépôt à utiliser pour cette langue.

Le paramètre `MORPHOLOGICAL DICTIONARY` indique la liste des dictionnaires du mode morphologique, séparés par des points virgules.

Le paramètre `MORPHOLOGICAL NODES COLOR` définit la couleur des étiquettes du mode morphologique `$<` et `$>`.

Le paramètre `MORPHOLOGICAL USE OF SPACE` indique si le programme `Locate` peut commencer en reconnaissant les espaces. (par défaut c'est non)

15.11.2 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` file. Les dictionnaires du système doivent se trouver dans le répertoire système Unitex, à l'intérieur du sous-répertoire `(langue courante)/Dela`. Voici un exemple de fichier :

```
delacf.bin¶
delaf.bin¶
```

15.11.3 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 travail.

15.11.4 Fichiers (nom d'utilisateur).cfg et .unitex.cfg

Sous Linux et Mac OS, Unitex considère que le répertoire personnel de travail se nomme `unitex` et qu'il se trouve dans le répertoire racine de l'utilisateur (`$HOME`). Si vous voulez changer cet emplacement par défaut, un fichier `.unitex.cfg` est créé dans votre répertoire racine, et il contient le chemin vers votre répertoire de travail Unitex. Ce fichier est un fichier

UTF8. Si `.unitex.cfg` ne contient pas un chemin Linux valide vers un répertoire existant, il est ignoré [1].

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 de travail. Ce fichier est sauvegardé sous le nom `(nom d'utilisateur).cfg` dans le sous-répertoire `Users` du répertoire système Unitex. Si l'utilisateur n'a pas les droits pour écrire dans ce répertoire, un fichier `.unitex.cfg` est sauvegardé dans le répertoire du profil utilisateur :

- dans `Documents and Settings\(user login)` sous Windows XP
- dans `Users\(user login)` sous WindowsVista ou une version plus récente.

ATTENTION : CE FICHIER N'EST PAS EN UNICODE ET LE CHEMIN DU RÉPERTOIRE PERSONNEL DE TRAVAIL N'EST PAS SUIVI PAR UN RETOUR À LA LIGNE.

15.12 Fichiers CasSys

15.12.1 Fichiers de configuration CasSys csc

Pour mémoriser la liste des transducteurs d'une cascade CasSys, nous utilisons un fichier texte (csc) dans lequel chaque ligne contient le chemin vers un transducteur suivi du mode de sortie (fusionner / remplacer) à appliquer à ce transducteur. Le format d'une ligne du fichier csc est : Name_and_path_of_transducer Merge Voici un exemple de fichier de cascade csc :

```
"C:\apps\my_unitex\French\Graphs\grf1.fst2" Merge
"C:\apps\my_unitex\French\Graphs\grf2.fst2" Replace
```

15.13 Plusieurs autres fichiers

Pour chaque texte, Unitex crée plusieurs fichiers contenant des informations destinées à être affichées dans l'interface graphique. Cette section décrit ces différents fichiers.

15.13.1 Fichier dlf.n, dlc.n, err.n et tags_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`, `err` et `tags_err`. Ces nombres sont suivis par un retour à la ligne.

1. Cela permet de lancer Unitex tantôt sous Linux, tantôt sous Windows, sur des fichiers partagés : le chemin Windows vers le répertoire personnel de travail Unitex est indiqué dans `.unitex.cfg`, et Unitex l'ignore quand on le lance sous Linux.

15.13.2 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` and `err`.

15.13.3 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).

15.13.4 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.

15.13.5 Fichier concord_tfst.n

Le fichier `concord_tfst.n` est un fichier texte qui se trouve dans le répertoire du texte. Il contient des informations sur la dernière recherche sur l'automate du texte et ressemble à ce qui suit :

```
23 matches(45 outputs)¶
```

15.13.6 Fichier règles de normalisation

Ce fichier est utilisé par les programmes `Normalization` et `XMLizer`. Il représente règles de normalisation. Chaque ligne représente une règle, selon le format suivant (⟼ représente le caractère de tabulation) :

```
input sequence ⟼ output sequence
```

Si vous souhaitez utiliser la tabulation ou le newline, vous devez les déspécialiser avec un antislash comme ceci :

```
123\
⟼ ONE_TWO_THREE_NEW_LINE
```

15.13.7 Fichier de mots interdits

Le programme `PolyLex` requiert un de mots interdits pour le hollandais et le norvégien. Ce fichier texte brut est censé s'appeler `ForbiddenWords.txt` . Il doit se trouver dans le répertoire `Dela` correspondant à la langue courante. Chaque ligne est censée contenir un mot interdit.

15.13.8 Fichier de log

Le programme `UnitexToolLogger`, si le fichier `unitex_logging_parameters.txt` est trouvé avec un chemin (pour enregistrer les fichiers log) crée un fichier .ulp de log de l'outil Unitex en cours d'exécution choisi.

Il crée un fichier `unitex_logging_parameters_count.txt` qui contient seulement le numéro du dernier fichier log créé.

Un fichier log (avec l'extension .ulp) est un fichiers zip non comprimés, compatibles avec unzip et tous les outils unzip standards. On peut le recréer avec zip d'Infozip (avec les options -0 -X). Il contient ces fichiers :

- `test_info/command_line.txt` : une liste de paramètres de la ligne de commande utilisée pour exécuter l'outil. Il y a un paramètre sur chaque ligne. La première ligne contient la valeur de retour, la deuxième ligne le nombre de paramètres ;
- `test_info/command_line_synth.txt` : une simple ligne avec un résumé de la ligne de commande utilisée pour exécuter l'outil ;
- `test_info/list_file_in.txt` : une liste des fichiers lus par l'outil. La première colonne est la taille du fichier, la seconde est crc32, la troisième le nom du fichier ;
- `test_info/list_file_out.txt` : une liste des fichiers créés par l'outil. La première colonne est la taille du fichier, la seconde est crc32, la troisième le nom du fichier ;
- `test_info/std_out.txt` : le contenu de sortie standard de la console ;

- `test_info/std_err.txt` : le contenu de sortie erreurs de la console ;
- `src/xxx` : une copie du fichier lu par l'outil (nécessaire pour faire fonctionner à nouveau le log) ;
- `dest/xxx` : une copie du fichier créé par l'outil

Si la seconde ligne de unitex_logging_parameters.txt contient 0, ces fichiers ne sont pas enregistrés ; si cette ligne contient 1, ils sont enregistrés ;

15.13.9 Règles typographiques de l'arabe : arabic_typo_rules.txt

Pour l'arabe, la recherche dans le dictionnaire peut être paramétrée avec un fichier de configuration nommé `Arabic/arabic_typo_rules.txt`, qui autorise certaines variations typographiques (section 3.8.5). Ce fichier est constitué de lignes comme celle-ci :

```
fatha omission=YES
```

où `fatha omission` est le nom d'une règle prédéfinie. Toutes les règles disponibles sont décrites dans le fichier `Arabic.h` dans les sources du programme.

15.13.10 Fichier d'offsets de différence

Les fichiers d'offsets de différence sont lu et écrit par l'outils Unxmlize(14.50), DumpOffsets(14.13), Normalize(14.29), Fst2Txt(14.21), Tokenize(14.43), Concord(14.9) et GrfTest, et lu par Tokenize(14.43). Ces fichiers textes sont constituées de lignes contenant 4 entiers A B C

D. Chaque ligne correspond à une modification du texte, exprimée de la façon suivante : l'intervalle [A ;B[du texte *** avant tout traitement *** est remplacé par l'intervalle [C ;D[

après traitement, A, B, C et D étant des positions en caractères dans les fichiers textes.

Par exemple, si on applique le programme Normalize sur le texte "Hello world" (avec deux espaces entre les deux mots), on aura une ligne comme ceci :

```
5 7 5 6
```

signifiant qu'une séquence de deux caractères (les 2 espaces) a été remplacée par une séquence d'un seul caractère.

Le principe est donc de produire un nouveau fichier d'offsets pour chaque application de programme modifiant le texte, en prenant en entrée le fichier d'offsets produit par le programme précédent. Ainsi, en regardant le dernier fichier d'offsets produit, on sait que pour chaque ligne A B C D, l'intervalle [C ;D[dans le fichier .snt correspond à l'intervalle [A ;B[dans le fichier .txt de départ

15.13.11 fichier d'offsets de zone commune

Les fichiers d'offsets de zone commune sont lu et écrit par DumpOffsets. Ces fichiers textes sont constituées de lignes contenant 4 entiers A B C D. Chaque ligne correspond à une modification du texte, exprimée de la façon suivante : l'intervalle [A ;B[du texte original correspond à l'intervalle [C ;D[après traitement, A, B, C et D étant des positions en caractères dans les fichiers textes. Sur chaque ligne, B-A=D-C.

Par exemple, si on applique le programme Normalize sur le texte "Hello world" (avec deux espaces entre les deux mots), on aura une ligne comme ceci :

```
0 5 0 5
7 12 6 11
```

signifiant que les caractère de 0 (inclus) à 5 (non inclus) des deux fichiers contiennent exactement le même texte, et que ceux de 7 (inclus) à 12 (non inclus) du premier texte contiennent exactement le même texte que ceux de 6 (inclus) à 11 (non inclus) du second.

15.13.12 fichier d'offsets uima

Les fichiers d'offsets uima sont écrit par Tokenize et lu par Concord (avec les options `--uima=`, `--xml-with-header=` ou `--xml=`). Ces fichiers établissent la correspondance entre chaque token successif et une position dans le fichier d'origine.

Ces fichiers textes sont constituées de lignes contenant 3 entiers A B C et de texte entre < et >.

Chaque ligne correspond à un token, exprimée de la façon suivante : Le token numéro A correspond au texte de la position B (inclue) à la position C (non inclus) du fichier d'origine, et le texte de ce token est mentionné entre < et >. Le numéro de token A correspond au numéro de ligne de tokens.txt où figure ce token (après avoir ajouté 1 pour la ligne d'en-tête de tokens.txt(15.4.4))

Annexe A - GNU Lesser General Public License

Cette licence peut également être trouvée ici [36].

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

[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!

Annexe B - Licences du type BSD à 2 clauses

B.1. TRE

Voici la licence, la note de copyright et la clause de non-responsabilité pour TRE, une bibliothèque de manipulation d'expressions régulières.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met :

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES ; LOSS OF USE, DATA, OR PROFITS ; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

B.2. wingetopt

Voici la licence, la note de copyright et la clause de non-responsabilité pour wingetopt, une bibliothèque getopt pour compilateurs Windows.

Sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F39502-99-1-0512.

Annexe C - Licence Apache de Xerces2

Voici la licence, la note de copyright et la clause de non-responsabilité pour Xerces2 Java Parser, un analyseur syntaxique XML utilisé par XAlign ([68]).

Apache License, Version 2.0, January 2004, http ://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.
"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.
"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.
"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.
"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.
"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.
"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.
"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix at the following address : http ://www.apache.org/licenses
"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."
"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions :

1. You must give any other recipients of the Work or Derivative Works a copy of this License ; and

2. You must cause any modified files to carry prominent notices stating that You changed the files ; and

3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works ; and

4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places : within a NOTICE text file distributed as part of the Derivative Works ; within the Source form or documentation, if provided along with the Derivative Works ; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Annexe D - Licence MIT de LibYAML

Voici la licence, la note de copyright et la clause de non-responsabilité pour la bibliothèque open source LibYAML d'analyse syntaxique YAML 1.1 écrite en C.

Annexe E - Licence open source TMate de SVNKit

Voici la licence, la note de copyright et la clause de non-responsabilité pour SVNKit, une bibliothèque open source de TMate Software.

This license applies to all portions of TMate SVNKit library which are not externally-maintained libraries (e.g. Ganymed SSH library).

All the source code and compiled classes in package org.tigris.subversion.javahl except SvnClient class are covered by the license in JAVAHL-LICENSE file

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met :

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Redistributions in any form must be accompanied by information on how to obtain complete source code for the software that uses SVNKit and any accompanying software that uses the software that uses SVNKit. The source code must either be included in the distribution or be available for no more than the cost of distribution plus a nominal fee, and must be freely redistributable under reasonable conditions. For an executable file, complete source code means the source code for all modules it contains. It does not include source code for modules or files that typically accompany the major components of the operating system on which the executable file runs.

4. Redistribution in any form without redistributing source code for software that uses SVNKit is possible only when such redistribution is explictly permitted by TMate Software. Please, contact TMate Software at support@svnkit.com to get such permission.

Annexe F - Lesser General Public License For Linguistic Resources

Cette licence a été conçue par l'Université de Marne-la-Vallée. Elle a reçu l'approbation de la Free Software Foundation ([1]) et figure sur la liste de licences open source (à partir de la version 2.1) du projet Software Package Data Exchange (SPDX) de la Fondation Linux.

Preamble

The licenses for most data are designed to take away your freedom to share and change it. By contrast, this License is intended to guarantee your freedom to share and change free data–to make sure the data are free for all their users.

This license, the Lesser General Public License for Linguistic Resources, applies to some specially designated linguistic resources – typically lexicons, grammars, thesauri and textual corpora.

Terms and Conditions For Copying, Distribution and Modification

0. This License Agreement applies to any Linguistic Resource 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 for Linguistic Resources (also called "this License"). Each licensee is addressed as "you".

   A "linguistic resource" means a collection of data about language prepared so as to be used with application programs.

   The "Linguistic Resource", below, refers to any such work which has been distributed under these terms. A "work based on the Linguistic Resource" means either the Linguistic Resource or any derivative work under copyright law : that is to say, a work containing the Linguistic Resource 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".)

   "Legible form" for a linguistic resource means the preferred form of the resource for making modifications to it.

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 Linguistic Resource is not restricted, and output from such a program is covered only if its contents constitute a work based on the Linguistic Resource (independent of the use of the Linguistic Resource in a tool for writing it). Whether that is true depends on what the program that uses the Linguistic Resource does.

1. You may copy and distribute verbatim copies of the Linguistic Resource 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 Linguistic Resource.

   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 Linguistic Resource or any portion of it, thus forming a work based on the Linguistic Resource, 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 linguistic resource.

   (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.

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Linguistic Resource, 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 Linguistic Resource, 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 Linguistic Resource.

   In addition, mere aggregation of another work not based on the Linguistic Resource with the Linguistic Resource (or with a work based on the Linguistic Resource) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. A program that contains no derivative of any portion of the Linguistic Resource, but is designed to work with the Linguistic Resource (or an encrypted form of the Linguistic Resource) by reading it or being compiled or linked with it, is called a "work that uses

the Linguistic Resource". Such a work, in isolation, is not a derivative work of the Linguistic Resource, and therefore falls outside the scope of this License.

However, combining a "work that uses the Linguistic Resource" with the Linguistic Resource (or an encrypted form of the Linguistic Resource) creates a package that is a derivative of the Linguistic Resource (because it contains portions of the Linguistic Resource), rather than a "work that uses the Linguistic Resource". If the package is a derivative of the Linguistic Resource, you may distribute the package under the terms of Section 4. Any works containing that package also fall under Section 4.

4. As an exception to the Sections above, you may also combine a "work that uses the Linguistic Resource" with the Linguistic Resource (or an encrypted form of the Linguistic Resource) to produce a package containing portions of the Linguistic Resource, and distribute that package under terms of your choice, provided that the terms permit modification of the package for the customer's own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the package that the Linguistic Resource is used in it and that the Linguistic Resource and its use are covered by this License. You must supply a copy of this License. If the package during execution displays copyright notices, you must include the copyright notice for the Linguistic Resource 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 package with the complete corresponding machine-readable legible form of the Linguistic Resource including whatever changes were used in the package (which must be distributed under Sections 1 and 2 above) ; and, if the package contains an encrypted form of the Linguistic Resource, with the complete machine-readable "work that uses the Linguistic Resource", as object code and/or source code, so that the user can modify the Linguistic Resource and then encrypt it to produce a modified package containing the modified Linguistic Resource.

   (b) Use a suitable mechanism for combining with the Linguistic Resource. A suitable mechanism is one that will operate properly with a modified version of the Linguistic Resource, if the user installs one, as long as the modified version is interface-compatible with the version that the package was made with.

   (c) Accompany the package with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 4a, above, for a charge no more than the cost of performing this distribution.

   (d) If distribution of the package 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.

   If the package includes an encrypted form of the Linguistic Resource, the required form of the "work that uses the Linguistic Resource" must include any data and uti-

lity programs needed for reproducing the package 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 proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Linguistic Resource together in a package that you distribute.

5. You may not copy, modify, sublicense, link with, or distribute the Linguistic Resource except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Linguistic Resource 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.

6. 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 Linguistic Resource or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Linguistic Resource (or any work based on the Linguistic Resource), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Linguistic Resource or works based on it.

7. Each time you redistribute the Linguistic Resource (or any work based on the Linguistic Resource), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Linguistic Resource 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.

8. 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 Linguistic Resource at all. For example, if a patent license would not permit royalty-free redistribution of the Linguistic Resource 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 Linguistic Resource.

   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 resource distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of data 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 resources 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.

9. If the distribution and/or use of the Linguistic Resource is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Linguistic Resource 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.

10. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License for Linguistic Resources 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 Linguistic Resource 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 Linguistic Resource does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

11. If you wish to incorporate parts of the Linguistic Resource into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission.

NO WARRANTY

12. BECAUSE THE LINGUISTIC RESOURCE IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LINGUISTIC RESOURCE, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LINGUISTIC RESOURCE "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 LINGUISTIC RESOURCE IS WITH YOU. SHOULD THE LINGUISTIC RESOURCE PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

13. 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 LINGUISTIC RESOURCE 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 LINGUISTIC RESOURCE (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 LINGUISTIC RESOURCE 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

Bibliographie

[1] Free Software Foundation. http://www.fsf.org. 15.13.12

[2] 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. 3.9

[3] Jean-Claude ANSCOMBRE. Pourquoi un moulin à vent n'est pas un ventilateur. *Langue Française*, 86, 1990. 11.1

[4] Laurie BAUER. *English Word-Formation*. Cambridge University Press, 1983. 11.1

[5] Emile BENVENISTE. *Fondements syntaxiques de la composition nominale. Formes nouvelles de la composition nominale*, pages 145–176. Gallimard, Paris, 1974. 11.1

[6] Olivier BLANC and Anne DISTER. Automates lexicaux avec structure de traits. In *Actes RECITAL 2004*, 2004. 7.3

[7] Xavier BLANCO. Noms composés et traduction français-espagnol. *Lingvisticæ Investigationes*, 21(1), 1997. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.9

[8] Xavier BLANCO. Les dictionnaires électroniques de l'espagnol (DELASs et DELACs). *Lingvisticæ Investigationes*, 23(2), 2000. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.9

[9] Jean-Paul BOONS, Alain GUILLET, and Christian LECLÈRE. La structure des phrases simples en français : classes de constructions transitives. Technical report, LADL, Paris, 1976. 9.1

[10] Jean-Paul BOONS, Alain GUILLET, and Christian LECLÈRE. *La structure des phrases simples en français : constructions intransitives*. Droz, Genève, 1976. 9.1

[11] Firefox. Web browser. http://www.mozilla.com/firefox/. 4.8.2

[12] Netscape. Web browser. http://www.netscape.com. 4.8.2

[13] Pierre CADIOT. A entre deux noms : vers la composition nominale. *Lexique*, 11 :193–240, 1992. 11.1

[14] Folker CAROLI. Les verbes transitifs à complément de lieu en allemand. *Lingvisticæ Investigationes*, 8(2) :225–267, 1984. Amsterdam-Philadelphia : John Benjamins Publishing Company. 9.1

[15] A. CHROBOT, B. COURTOIS, M. HAMMANI-MC CARTHY, M. GROSS, and K. ZELLAGUI. Dictionnaire electronique DELAC anglais : noms composés. Technical Report 59, LADL, Université Paris 7, 1999. 3.9

[16] Unicode Consortium. http://www.unicode.org. 2.2

[17] Matthieu CONSTANT and Anastasia YANNACOPOULOU. 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. 3.9

[18] Danielle CORBIN. Hypothèses sur les frontières de la composition nominale. *Cahiers de grammaire*, 17 :26–55, 1992. Université de Toulouse Le Mirail. 11.1

[19] Blandine COURTOIS. Formes ambiguës de la langue française. *Lingvisticæ Investigationes*, 20(1) :167–202, 1996. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.9

[20] Blandine Courtois and Max Silberztein, editors. *Les dictionnaires électroniques du français*. Larousse, Langue française, vol. 87, 1990. 3.9, 11.2.1, 11.2.2

[21] 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. 2.5.2

[22] Pamela DOWNING. On the Creation and Use of English Compound Nouns. In *Proceedings of CICLING-2002*, volume 53, pages 810–842. Linguistic Society of America, 1977. 11.1

[23] Dana-Marina DUMITRIU and Sébastien PAUMIER. Requêtes linguistiques sur alignements multilingues. In *Directia Terminologie si Inginerie Lingvistica (DTIL'08)*, February 2008. ISBN : 978-9-291220-37-3. 10

[24] Inkscape. Vector Graphics Editor. http://www.inkscape.org. 5.4.1

[25] Samuel ELEUTERIO, Elisabete RANCHHOD, Helena FREIRE, and Jorge BAPTISTA. A system of electronic dictionaries of portuguese. *Lingvisticæ Investigationes*, 19(1) :57–82, 1995. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.9

[26] Anibale ELIA. *Le verbe italien. Les complétives dans les phrases à un complément*. Schena/Nizet, Fasano/Paris, 1984. 9.1

[27] Anibale ELIA. *Lessico-grammatica dei verbi italiani a completiva. Tavole e indice generale*. Liguori, Napoli, 1984. 9.1

[28] 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. 3.9

[29] 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. 3.9

[30] Vassiliki Foufi. Les noms composés A(A)N du Grec Moderne et leurs variantes. In Kakoyianni Doa [57]. 11.2

[31] Nathalie FRIBURGER. *Reconnaissance automatique des noms propres : application à la classification automatique de textes journalistiques*, 2002. Thèse de doctorat. Université de Tours. 12

[32] A Simple English Axis Generator. http://nlp.cs.nyu.edu/GMA/docs/HOWTO-axis. 14.9

[33] Jacqueline GIRY-SCHNEIDER. Syntax and lexicon : Blessure (wound), noeud (knot), caresse (caress)... *SMIL, Journal of Linguistic Calculus*, 3-4 :55–72, 1978. 9.1

[34] Jacqueline GIRY-SCHNEIDER. *Les nominalisations en français. L'opérateur faire dans le lexique*. Droz, Genève-Paris, 1978. 9.1

[35] Jacqueline GIRY-SCHNEIDER. *Les prédicats nominaux en français. Les phrases simples à verbe support*. Droz, Genève-Paris, 1987. 9.1

[36] GNU. Lesser General Public License. http://www.gnu.org/licenses/lgpl.html. 1.1, 15.13.12

[37] Gaston GROSS. Définition des noms composés dans un lexique-grammaire. *Langue Française*, 87, 1990. 11.1

[38] Gaston GROSS. *Les expressions figées en français. Noms composés et autres locutions*. Ophrys, Paris, 1996. 3.9, 11.1

[39] Maurice GROSS. *Méthodes en syntaxe*. Hermann, Paris, 1975. 9.1

[40] Maurice GROSS. Sur quelques groupes nominaux complexes. In J.-C. Chevalier et M. Gross, editor, *Méthodes en grammaire française*, pages 97–119. Paris : Klincksieck, 1976. 9.1

[41] Maurice GROSS. Taxonomy in syntax. *SMIL, Journal of Linguistic Calculus*, 3-4 :73–96, 1978. 9.1

[42] Maurice GROSS. Simple sentences. Discussion of Fred W. Householder's paper (analysis, synthesis and improvisation). In Sture Allen, editor, *Text Processing, Proceedings of Nobel Symposium 51*, pages 297–315. Stockholm :Almqvist Wiksell, 1982. 9.1

[43] Maurice GROSS. On structuring the lexicon. *Quaderni di Semantica*, 4(1) :107–120, 1983. 9.1

[44] Maurice GROSS. Lexicon-grammar and the syntactic analysis of french. In *Proceedings of the 10 th International Conference on Computational Linguistics (COLING'84)*. Stanford, California, 1984. 9.1

[45] Maurice GROSS. A linguistic environment for comparative romance syntax. In Ph. Baldi, editor, *Papers from the XIIth Linguistic Symposium on Romance Languages*, volume IV(26) of *Amsterdam studies in the theory and history of linguistic science*, pages 373–446. Amsterdam/Philadelphia : Benjamins, 1984. 9.1

[46] Maurice GROSS. *Grammaire transformationnelle du français. 3 - Syntaxe de l'adverbe*. ASSTRIL, Paris, 1986. 3.9, 9.1

[47] Maurice GROSS. Lexicon-grammar. the representation of compound words. In *COLING-1986 Proceedings*, pages 1–6. Bonn, 1986. 9.1

[48] Maurice GROSS. Methods and tactics in the construction of a lexicon-grammar. In *Linguistics in the Morning Calm 2, Selected papers from SICOL*, pages 177–197. Seoul : Hanshin, 1986. 9.1

[49] Maurice GROSS. Linguistic representations and text analysis. In *Linguistic Unity and Linguistic Diversity in Europe*, pages 31–61. London : Academia Europaea, 1991. 9.1

[50] Maurice GROSS. Constructing lexicon-grammars. In Atkins and Zampolli, editors, *Computational Approaches to the Lexicon*, pages 213–263. Oxford Univ. Press, 1994. 9.1

[51] Maurice GROSS. The lexicon-grammar of a language : Application to french. In R.E. Asher, editor, *The Encyclopedia of Language and Linguistics*, volume 4, pages 2195–2205. Oxford/NewYork/Seoul/Tokyo : Pergamon, 1994. 9.1

[52] Alain GUILLET and Christian LECLÈRE. *La structure des phrases simples en français : les constructions transitives locatives*. Droz, Genève, 1992. 9.1

[53] Benoît HABERT and Christian JACQUEMIN. Noms composés, termes, dénominations complexes : problématiques linguistiques et traitements automatiques. *Traitement Automatique des Langues*, 2 :5–41, 1993. 11.1

[54] IGM. Lesser General Public License for Linguistic Resources. `http://igm.univ-mlv/~unitex/lgpllr.html`. 1.1

[55] Text Encoding Initiative. `http://www.tei-c.org`. 10.1

[56] Christian JACQUEMIN. *Spotting and Discovering Terms through Natural Language Processing*. MIT Press, 2001. 11.2.3

[57] Fryni Kakoyianni Doa, editor. *Penser le lexique-grammaire : perspectives actuelles*. Editions Honoré Champion, Paris, France, 2014. 30, 79

[58] Gaby KLARSFLED and Mary HAMMANI-MC CARTHY. Dictionnaire électronique du ladl pour les mots simples de l'anglais (DELASa). Technical report, LADL, Université Paris 7, 1991. 3.9

[59] Cvetana KRSTEV, Duško VITAS, and Agata SAVARY. Prerequisites for a Comprehensive Dictionary of Serbian Compounds. *LNCS*, 4139 :552–563, 2006. 11.2

[60] Tita KYRIACOPOULOU. *Les dictionnaires électroniques : la flexion verbale en grec moderne*, 1990. Thèse de doctorat. Université Paris 8. 3.9

[61] 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. 3.9

[62] Tita KYRIACOPOULOU, Safia MRABTI, and AnastasiaYANNACOPOULOU. Le dictionnaire électronique des noms composés en grec moderne. *Lingvisticæ Investigationes*, 25(1) :7–28, 2002. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.9

[63] Jacques LABELLE. Le traitement automatique des variantes linguistiques en français : l'exemple des concrets. *Lingvisticæ Investigationes*, 19(1) :137–152, 1995. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.9

[64] Eric LAPORTE and Anne MONCEAUX. Elimination of lexical ambiguities by grammars : The ELAG system. *Lingvisticæ Investigationes*, 22 :341–367, 1998. Amsterdam-Philadelphia : John Benjamins Publishing Company. 7.3

[65] Ville LAURIKARI. TRE home page. http://laurikari.net/tre/. 1, 4.7

[66] Christian LECLÈRE. The lexicon-grammar of french verbs : a syntactic database. In Kawaguchi Y. et alii, editor, *Linguistic Informatics - State of the Art and the Future*, pages 29–45. Amsterdam/Philadelphia : Benjamins, 2005. 9.1

[67] Judith N. LEVI. *The Syntax and Semantics of Complex Nominals*. Academic Press, New York-London, 1978. 11.1

[68] XAlign(Alignement multilingue) LORIA (2006). http://led.loria.fr/outils/ALIGN/align.html. 10, 15.13.12

[69] Annie MEUNIER. *Nominalisation d'adjectifs par verbes supports*, 1981. Thèse de doctorat. Université Paris 7. 9.1

[70] Sun Microsystems. Java. http://java.sun.com. 1.2

[71] Christian MOLINIER and Françoise LEVRIER. *Grammaire des adverbes : description des formes en -ment*. Droz, Genève, 2000. 9.1

[72] Anne MONCEAUX. Le dictionnaire des mots simples anglais : mots nouveaux et variantes orthographiques. Technical Report 15, IGM, Université de Marne-la-Vallée, 1995. 3.9

[73] Marcello C. M. MUNIZ, Maria das Graças V. NUNES, and Eric LAPORTE. UNITEX-PB, a set of flexible language resources for Brazilian Portuguese. In *Proceedings of the Workshop on Technology of Information and Human Language*, 2005. São Leopoldo (Brazil) : Unisinos. 3.9

[74] Alexis NEME. A lexicon of arabic verbs constructed on the basis of semitic taxonomy and using finite-state transducers. In *Proceedings of the International Workshop on Lexical Resources (WoLeR) at ESSLLI*. Ljubljana, Slovenia, 2011. 3.6

[75] OpenOffice.org. http://www.openoffice.org. 2.2, 9.2.2

[76] Dong-Ho PAK. *Lexique-grammaire comparé français-coréen. Syntaxe des constructions complétives*. PhD thesis, UQAM, Montréal, 1996. 9.1

[77] Soun-Nam PARK. *La construction des verbes neutres en coréen*, 1996. Thèse de doctorat. Université Paris 7. 9.1

[78] Sébastien PAUMIER and Dana-Marina DUMITRIU. Editable text alignments and powerful linguistic queries. In Matthieu Constant, Takuya Nakamura, Michele De Gioia, and Sara Vecchiato, editors, *27th International Conference on Lexis and Grammar (LGC'08)*, pages 117–125, September 2008. 10, 10.2

[79] Sébastien PAUMIER and Jee-Sun NAM. Un système de dictionnaire de mots simples du coréen. In Kakoyianni Doa [57], pages 481–490. 6.9.3

[80] Adam PRZEPIÓRKOWSKI and Marcin WOLIŃSKI. The Unbearable Lightness of Tagging : A Case Study in Morphosyntactic Tagging of Polish. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora, EACL 2003*, 2003. 11.1.1, 11.2.2

[81] Roger-Bruno RABENNILAINA. *Le verbe malgache*. AUPELF-UREF et Université Paris 13, Paris, 1991. 9.1

[82] Elisabete RANCHHOD. Frozen adverbs. comparative forms como c in portuguese. *Lingvisticæ Investigationes*, 15(1) :141–170, 1991. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.9, 9.1

[83] Elisabete RANCHHOD. Ressources linguistiques du portugais implémentées sous intex. In C. Fairon, editor, *Analyse Lexicale et Syntaxique : Le système INTEX*, Lingvisticae Investigationes, pages 263–277. Amsterdam-Philadelphia : John Benjamins Publishing Company, 1998. 3.9

[84] Elisabete RANCHHOD. Problèmes de traduction automatique des constructions à verbes supports. *Lingvisticæ Investigationes*, 23(2) :253–267, 2001. Amsterdam-Philadelphia : John Benjamins Publishing Company. 9.1

[85] Elisabete RANCHHOD and Michele DE GIOIA. Comparative romance syntax. frozen adverbs in italian and in portuguese. *Lingvisticæ Investigationes*, 20(1) :33–85, 1996. Amsterdam-Philadelphia : John Benjamins Publishing Company. 9.1

[86] Elisabete RANCHHOD and Samuel ELEUTERIO. Construção de dicionários electrónicos do português. problemas teóricos e metodológicos. In *Actas do Congresso Internacional sobre o Português*, pages 265–282, 1996. Lisboa, Colibri. 3.9

[87] Morris SALKOFF. Verbs of mental states. In *Lexique, syntaxe et lexique-grammaire. Papers in honour of Maurice Gross*, volume 24 of *Lingvisticæ Investigationes Supplementa*, pages 561–571. Amsterdam/Philadelphia : Benjamins, 2004. 9.1

[88] Agata SAVARY. *Recensement et description des mots composés - méthodes et applications*, 2000. Thèse de doctorat. Université de Marne-la-Vallée. 3.9, 11.1.1, 11.1.2

[89] Agata SAVARY. A formalism for the computational morphology of multi-word units. *Archives of Control Sciences*, 15(3) :437–449, 2005. 11, 11.1.2, 11.2

[90] Max SILBERZTEIN. Les groupes nominaux productifs et les noms composés lexicalisés. *Lingvisticæ Investigationes*, 27(2) :405–426, 1999. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.9, 11.1

[91] Carlos SUBIRATS-RÜGGEBERG. *Sentential complementation in Spanish. A lexico-grammatical study of three classes of verbs*. John Benjamins, Amsterdam/Philadelphia, 1987. 9.1

[92] Thomas TREIG. Complétives en allemand. classification. Technical Report 7, LADL, 1977. 9.1

[93] 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. 9.1

[94] 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. 3.9

[95] Duško VITAS, Svetla KOEVA, Cvetana KRSTEV, and Ivan OBRADOVIĆ. Tour du monde through the dictionaries. In Matthieu Constant, Takuya Nakamura, Michele De Gioia, and Sara Vecchiato, editors, *27th International Conference on Lexis and Grammar (LGC'08)*, pages 249–256, September 2008. 10

Index