

UNITEX 2.1

USER MANUAL



Université Paris-Est Marne-la-Vallée

<http://www-igm.univ-mlv.fr/~unitex>
unitex@univ-mlv.fr

Sébastien Paumier

English translation of version 1.2 by the local grammar group at the
CIS, Ludwig-Maximilians-Universität, Munich - Oct 2003
(Wolfgang Flury, Franz Guenther, Friederike Malchok, Clemens Marschner, Sebastian
Nagel, Johannes Stiehler)

<http://www.cis.uni-muenchen.de/>

Contents

Introduction	11
What's new from version 2.0 ?	12
Content	13
Unitex contributors	14
If you use Unitex in research projects...	15
1 Installation of Unitex	17
1.1 Licenses	17
1.2 Java runtime environment	17
1.3 Installation on Windows	18
1.4 Installation on Linux	18
1.5 Installation on MacOS X	19
1.5.1 Using the Apple Java 1.6 runtime	19
1.5.2 SoyLatte	20
1.5.3 How to compile Unitex C++ programs on a Macintosh	24
1.5.4 How to makes all files visible on Mac OS	26
1.6 First use	27
1.7 Adding new languages	27
1.8 Uninstalling Unitex	28
1.9 Unitex for developpers	28
2 Loading a text	31
2.1 Selecting a language	31
2.2 Text formats	31
2.3 Editing text files	34
2.4 Opening a text	34
2.5 Preprocessing a text	34
2.5.1 Normalization of separators	36
2.5.2 Splitting into sentences	37
2.5.3 Normalization of non-ambiguous forms	39
2.5.4 Splitting a text into tokens	41
2.5.5 Applying dictionaries	42
2.5.6 Analysis of compound words in Dutch, German, Norwegian and Russian	44

2.6	Opening a tagged text	44
3	Dictionaries	47
3.1	The DELA dictionaries	47
3.1.1	The DELAF format	47
3.1.2	The DELAS Format	50
3.1.3	Dictionary Contents	51
3.2	Checking dictionary format	53
3.3	Sorting	54
3.4	Automatic inflection	56
3.4.1	Inflection of simple words	56
3.4.2	Inflection of compound words	60
3.4.3	Inflection of semitic languages	61
3.5	Compression	61
3.6	Applying dictionaries	63
3.6.1	Priorities	63
3.6.2	Application rules for dictionaries	64
3.6.3	Dictionary graphs	64
3.6.4	Morphological dictionary graphs	66
3.7	Bibliography	68
4	Searching with regular expressions	71
4.1	Definition	71
4.2	Tokens	71
4.3	Lexical masks	72
4.3.1	Special symbols	72
4.3.2	References to information in the dictionaries	73
4.3.3	Grammatical and semantic constraints	73
4.3.4	Inflectional constraints	74
4.3.5	Negation of a lexical mask	74
4.4	Concatenation	76
4.5	Union	77
4.6	Kleene star	77
4.7	Morphological filters	77
4.8	Search	79
4.8.1	Search configuration	79
4.8.2	Presentation of the results	80
4.8.3	Statistics	85
5	Local grammars	89
5.1	The local grammar formalism	89
5.1.1	Algebraic grammars	89
5.1.2	Extended algebraic grammars	90
5.2	Editing graphs	90
5.2.1	Creating a graph	90

<i>CONTENTS</i>	5
5.2.2 Sub-Graphs	93
5.2.3 Manipulating boxes	96
5.2.4 Transducers	97
5.2.5 Using Variables	98
5.2.6 Copying lists	99
5.2.7 Special Symbols	100
5.2.8 Toolbar Commands	100
5.3 Display options	101
5.3.1 Sorting the lines of a box	101
5.3.2 Zoom	102
5.3.3 Antialiasing	102
5.3.4 Box alignment	104
5.3.5 Display options, fonts and colors	105
5.4 Exporting graphs	107
5.4.1 Inserting a graph into a document	107
5.4.2 Printing a Graph	108
6 Advanced use of graphs	109
6.1 Types of graphs	109
6.1.1 Inflection transducers	109
6.1.2 Preprocessing graphs	110
6.1.3 Graphs for normalizing the text automaton	111
6.1.4 Syntactic graphs	112
6.1.5 ELAG grammars	112
6.1.6 Parameterized graphs	113
6.2 Compilation of a grammar	113
6.2.1 Compilation of a graph	113
6.2.2 Approximation with a finite state transducer	113
6.2.3 Constraints on grammars	114
6.2.4 Error detection	116
6.3 Contexts	117
6.3.1 Right contexts	118
6.3.2 Left contexts	120
6.4 The morphological mode	124
6.4.1 Why ?	124
6.4.2 The rules	124
6.4.3 Morphological dictionaries	125
6.4.4 Dictionary entry variables	126
6.5 Exploring grammar paths	127
6.6 Graph collections	129
6.7 Rules for applying transducers	130
6.7.1 Insertion to the left of the matched pattern	130
6.7.2 Application while advancing through the text	131
6.7.3 Priority of the leftmost match	131
6.7.4 Priority of the longest match	132

6.7.5	Transducer outputs with variables	132
6.8	Output variables	135
6.9	Operations on variables	136
6.9.1	Testing variables	136
6.9.2	Comparing variables	137
6.10	Applying graphs to texts	137
6.10.1	Configuration of the search	137
6.10.2	Advanced search options	139
6.10.3	Concordance	142
6.10.4	Modification of the text	143
6.10.5	Extracting occurrences	144
6.10.6	Comparing concordances	144
7	Text automaton	147
7.1	Displaying text automaton	147
7.2	Construction	149
7.2.1	Construction rules for text automata	149
7.2.2	Normalization of ambiguous forms	150
7.2.3	Normalization of clitical pronouns in Portuguese	151
7.2.4	Keeping the best paths	153
7.3	Resolving Lexical Ambiguities with ELAG	157
7.3.1	Grammars For Resolving Ambiguities	157
7.3.2	Compiling ELAG Grammars	158
7.3.3	Resolving Ambiguities	160
7.3.4	Grammar collections	162
7.3.5	Window For ELAG Processing	162
7.3.6	Description of the tag sets	163
7.3.7	Grammar Optimization	169
7.4	Linearizing text automaton with the tagger	170
7.4.1	Compatibility of the tagset	171
7.4.2	Use of the Tagger	172
7.4.3	Creation of a new tagger	172
7.5	Manipulation of text automata	173
7.5.1	Displaying sentence automata	173
7.5.2	Modifying the text automaton	174
7.5.3	Display configuration	175
7.6	Converting the text automaton into linear text	175
7.7	Searching patterns in the text automaton	176
7.8	Table display	178
7.9	The special case of Korean	179
8	Lexicon-grammar	183
8.1	Lexicon-grammar tables	183
8.2	Conversion of a table into graphs	184
8.2.1	Principle of parameterized graphs	184

8.2.2	Format of the table	184
8.2.3	Parameterized graphs	185
8.2.4	Automatic generation of graphs	186
9	Text alignment	191
9.1	Loading texts	191
9.2	Aligning texts	193
9.3	Pattern matching	195
10	Compound word inflection	199
10.1	Multi-Word Units	199
10.1.1	Formal Description of the Inflectional Behavior of Multi-word Units	200
10.1.2	Lexicalized vs. Grammar-Based Approach to Morphological Description	201
10.2	Formalism for the Computational Morphology of MWUs	202
10.2.1	Morphological Features of the Language	202
10.2.2	Decomposition of a MWU into Units	204
10.2.3	Inflection paradigm of a MWU	205
10.3	Integration in Unitex	210
10.3.1	Complete Example in English	211
10.3.2	Complete Example in French	214
10.3.3	Complete Example in Serbian	217
11	Cascade of Transducers	227
11.1	Applying a cascade of transducers with CasSys	227
11.1.1	Creating the list of transducers	227
11.1.2	Editing the list of transducers	228
11.1.3	Launching a cascade	229
11.1.4	Displaying the results of a cascade	229
11.2	Details on Cassys	230
11.2.1	Type of graphs used	231
11.2.2	The Unitex rules used for the cascade	231
11.2.3	A special way to mark up patterns with CasSys	231
11.2.4	Interest of a cascade of transducer	233
11.2.5	The longest pattern	233
11.2.6	Files resulting from CasSys	234
12	Use of external programs	235
12.1	Creating log files	236
12.2	The console	236
12.3	Text file encoding parameters	237
12.4	BuildKrMwuDic	237
12.5	Cassys	238
12.6	CheckDic	239
12.7	Compress	240

12.8	Concord	240
12.9	ConcorDiff	242
12.10	Convert	243
12.11	Dico	245
12.12	Elag	246
12.13	ElagComp	246
12.14	Evamb	246
12.15	Extract	247
12.16	Flatten	247
12.17	Fst2Check	247
12.18	Fst2List	248
12.19	Fst2Txt	249
12.20	Grf2Fst2	250
12.21	ImplodeTfst	251
12.22	Locate	251
12.23	LocateTfst	253
12.24	MultiFlex	254
12.25	Normalize	255
12.26	PolyLex	256
12.27	RebuildTfst	256
12.28	Reconstruca	257
12.29	Reg2Grf	257
12.30	SortTxt	257
12.31	Stats	258
12.32	Table2Grf	259
12.33	Tagger	259
12.34	TagsetNormTfst	260
12.35	TEI2Txt	260
12.36	Tfst2Grf	260
12.37	Tfst2Unambig	261
12.38	Tokenize	261
12.39	TrainingTagger	262
12.40	Txt2Tfst	263
12.41	Uncompress	263
12.42	Untokenize	264
12.43	UnitexTool	264
12.44	UnitexToolLogger	265
12.45	XMLizer	267
13	File formats	269
13.1	Unicode encoding	269
13.2	Alphabet files	270
13.2.1	Alphabet	270
13.2.2	Sorted alphabet	271
13.3	Graphs	272

13.3.1	Format .grf	272
13.3.2	Format .fst2	275
13.4	Texts	276
13.4.1	.txt files	276
13.4.2	.snt files	277
13.4.3	File text.cod	277
13.4.4	The tokens.txt file	277
13.4.5	The tok_by_alph.txt and tok_by_freq.txt files	277
13.4.6	The enter.pos file	277
13.5	Text Automaton	278
13.5.1	The text.tfst file	278
13.5.2	The text.tind file	280
13.5.3	The cursentence.grf file	281
13.5.4	The sentenceN.grf file	281
13.5.5	The cursentence.txt file	281
13.5.6	The cursentence.tok file	281
13.5.7	The tfst_tags_by_freq.txt and tfst_tags_by_alph.txt files	282
13.6	Concordances	282
13.6.1	The concord.ind file	282
13.6.2	The concord.txt file	283
13.6.3	The concord.html file	283
13.6.4	The diff.html file	284
13.7	Text dictionaries	285
13.7.1	dlf and dlc	285
13.7.2	err	285
13.7.3	tags_err	285
13.7.4	tags.ind	285
13.8	Dictionaries	286
13.8.1	The .bin files	286
13.8.2	The .inf files	286
13.8.3	Dictionary information file	288
13.8.4	The CHECK_DIC.TXT file	288
13.9	ELAG files	290
13.9.1	tagset.def file	290
13.9.2	.lst files	290
13.9.3	.elg files	290
13.9.4	.rul files	291
13.10	Tagger files	291
13.10.1	The corpus.txt file	291
13.10.2	The tagger data file	292
13.11	Configuration files	293
13.11.1	The Config file	293
13.11.2	The system_dic.def file	295
13.11.3	The user_dic.def file	296
13.11.4	The user.cfg and .unitex.cfg files	296

13.12	Cassys files	296
13.12.1	Cassys configuration files csc	296
13.13	Various other files	296
13.13.1	The dlf.n, dlc.n, err.n et tags_err.n files	296
13.13.2	The stat_dic.n file	297
13.13.3	The stats.n file	297
13.13.4	The concord.n file	297
13.13.5	The concord_tfst.n file	297
13.13.6	Normalization rule file	298
13.13.7	Forbidden word file	298
13.13.8	Log file	298
13.13.9	Arabic typographic rules: arabic_typo_rules.txt	299
Appendix A - GNU Lesser General Public License		301
Appendix B - TRE's 2-clause BSD License		311
Appendix C - Lesser General Public License For Linguistic Resources		313

Introduction

Unitex is a collection of programs developed for the analysis of texts in natural language by using linguistic resources and tools. These resources consist of electronic dictionaries, grammars and lexicon-grammar tables, initially developed for French by Maurice Gross and his students at the Laboratoire d'Automatique Documentaire et Linguistique (LADL). Similar resources have been developed for other languages in the context of the RELEX laboratory network.

The electronic dictionaries specify the simple and compound words of a language together with their lemmas and a set of grammatical (semantic and inflectional) codes. The availability of these dictionaries is a major advantage compared to the usual utilities for pattern searching as the information they contain can be used for searching and matching, thus describing large classes of words using very simple patterns. The dictionaries are presented in the DELA formalism and were constructed by teams of linguists for several languages (French, English, Greek, Italian, Spanish, German, Thai, Korean, Polish, Norwegian, Portuguese, etc.)

The grammars used here are representations of linguistic phenomena on the basis of recursive transition networks (RTN), a formalism closely related to finite state automata. Numerous studies have shown the adequacy of automata for linguistic problems at all descriptive levels from morphology and syntax to phonetic issues. Grammars created with Unitex carry this approach further by using a formalism even more powerful than automata. These grammars are represented as graphs that the user can easily create and update.

Lexicon-grammar tables are matrices describing properties of some words. Many such tables have been constructed for all simple verbs in French as a way of describing their relevant syntactic properties. Experience has shown that every word has a quasi-unique behavior, and these tables are a way to present the grammar of every element in the lexicon, hence the name lexicon-grammar for this linguistic theory. Unitex offers a way to automatically build grammars from lexicon-grammar tables.

Unitex can be viewed as a tool in which one can put linguistic resources and use them. Its technical characteristics are its portability, modularity, the possibility of dealing with languages that use special writing systems (e.g. many Asian languages), and its openness, thanks to its open source distribution. Its linguistic characteristics are the ones that have motivated the elaboration of these resources: precision, completeness, and the taking into

account of frozen expressions, most notably those which concern the enumeration of compound words.

What's new from version 2.0 ?

Here are the main new features:

- introduction of `LocateTfst` that performs locate operations on the text automaton (7.7, 12.23)
- `LocateTfst` fully supports Korean (7.9)
- new search options (6.10.2)
- introduction of `Stats` that can compute statistics from a concordance file (4.8.3)
- introduction of a statistical tagger that can trim text automata to make them linear (7.4)
- introduction of the transducer cascade system `CasSys` (chapter 11)
- introduction of output variables that can be used to catch outputs emitted by grammars (6.8)
- introduction of operators to test and compare variables (6.9)
- supporting semitic inflection (3.4.3), with a full support of Arabic typographical variations (13.13.9)
- new options to apply dictionary graphs (3.6.3)
- introduction of `Uncompress` that can rebuild a `.dic` dictionary from a `.bin` compressed one (12.41)
- introduction of `Untokenize` that can rebuild a `.snt` text file from `text.cod` and `tokens.txt` (12.42)
- introduction of ancient Georgian
- the console has changed: you can view error messages emitted by previous commands (12.2)
- there is a tutorial for installing Unitex on MacOS X (1.5)
- new text automaton format `.tfst` (13.5.1)
- Unitex programs have Unix style options
- Unitex is now pure LGPL (1.1)
- Unitex can be compiled as a dynamic library (`.dll` or `.so`) (1.9)

- Some Unix programs can run without memory leak when compiled with preprocessor macro `UNITEX_RELEASE_MEMORY_AT_EXIT` or `UNITEX_LIBRARY`. This include `Concord`, `Convert`, `Dico`, `Elag`, `Evamb`, `Extract`, `Flatten`, `Fst2Txt`, `Grf2Fst2`, `ImplodeTfst`, `LocateTfst`, `MultiFlex`, `Normalize`, `PolyLex`, `Reg2Grf`, `SortTxt`, `TagsetNormTfst`, `TEI2Txt`, `Tfst2Grf`, `Tfst2Unambig`, `Tokenize`, `Txt2Tfst`, `XMLizer`.
- Unix code is thread-safe
- Encoding of Unicode text file can be specified (12.3)
- introduction of `UnitexTool` that can be used to launch several Unix programs in one command to make scripting easier (12.43)
- introduction of `UnitexToolLogger` that can be used to create and run again log of Unix programs execution. (12.44)

IMPORTANT: as some file formats changed and some new files were introduced, we recommend that you reprocess your existing text files, especially if you work with text automata.

Content

Chapter 1 describes how to install and run Unix.

Chapter 2 presents the different steps in the analysis of a text.

Chapter 3 describes the formalism of the DELA electronic dictionaries and the different operations that can be applied to them.

Chapters 4 and 5 present different means for making text searches more effective. Chapter 5 describes in detail how to use the graph editor.

Chapter 6 is concerned with the different possible applications of grammars. The particularities of each type of grammar are presented.

Chapter 7 introduces the concept of text automaton and describes the properties of this notion. This chapter also describes operations on this object, in particular, how to disambiguate lexical items with the ELAG program.

Chapter 8 contains an introduction to lexicon-grammar tables, followed by a description of the method of constructing grammars based on these tables.

Chapter 9 describes the text alignment module, based on the XAlign tool.

Chapter 10 describes the compound word inflection module, as a complement of the simple word inflection mechanism presented in chapter 3.

Chapter 11 describes the CasSys cascade of transducer system.

Chapter 12 contains a detailed description of the external programs that make up the Unitex system.

Chapter 13 contains descriptions of all file formats used in the system.

The reader will find in appendix the LGPL license under which the Unitex source code is released, as well as the LGPLLR license which applies for the linguistic data distributed with Unitex. There is also the 2-clause BSD licence that applies to the TRE library, used by Unitex for morphological filters.

Unitex contributors

Unitex was born as a bet on the power of Open Source philosophy in the academic world (see http://igm.univ-mlv.fr/~unitex/why_unitex.html), relying on the assumption that people would be interested in sharing their knowledge and skill into such an open project. The following list sounds like Open Source is good for science:

- Olivier Blanc: has integrated the ELAG system into Unitex, originally designed by Eric Laporte, Anne Monceaux and some of their students, has also written `RebuildTfst` (previously known as `MergeTextAutomaton`)
- Matthieu Constant: author of `Grf2Fst2`
- Julien Decretton: author of the text editor integrated in Unitex, has also designed the undo functionality in the graph editor
- Claude Devis: introduction of morphological filters, based on the TRE library
- Hyun-Gue Huh: author of the tools used to generate Korean dictionaries
- Claude Martineau: had worked on the simple word inflection part of `MultiFlex`
- Sebastian Nagel: has optimized many parts of the code, has also adapted `PolyLex` for German and Russian
- Alexis Neme: has optimized `Dico` and `Tokenize`, has also merged `Locate` into `Dico` in order to allow dictionary graphs
- Aljosa Obuljen: author of `Stats`
- Sébastien Paumier: main developer
- Agata Savary: author of `MultiFlex`
- Gilles Vollant: author of `UnitexTool`, has optimized many aspects of Unitex code (memory, speed, multi-compiler compliance, etc)

- Patrick Watrin: author of XMLizer, has worked on the integration of XAlign
- Anthony Sigogne: author of Tagger and TrainingTagger
- Nathalie Friburger: author of Cassys

Moreover, Unitex would be useless without all the precious linguistic resources it contains. All those resources are the result of hard work done by people that shall not be forgotten. Some are mentioned in disclaimers that come with dictionaries, and complete information is available on:

http://igm.univ-mlv.fr/~unitex/linguistic_data_bib.html

If you use Unitex in research projects...

Unitex has been used in several research projects. Some are listed in the “Related works” section of Unitex home page. If you did some work with Unitex (resources, project, paper, thesis, ...) and if you want it to be referenced in the web site, just send a mail to unitex@univ-mlv.fr. The more visible, the more cited!

Chapter 1

Installation of Unitex

Unitex is a multi-platform system that runs on Windows as well as on Linux or MacOS. This chapter describes how to install and how to launch Unitex on any of these systems. It also presents the procedures used to add new languages and to uninstall Unitex.

1.1 Licenses

Unitex is a free software. This means that the sources of the programs are distributed with the software, and that anyone can modify and redistribute them. The code of the Unitex programs is under the LGPL licence ([35]), except for the TRE library for dealing with regular expressions from Ville Laurikari ([63]), which is under 2-clause BSD licence which is more permissive than the LGPL. The LGPL Licence is more permissive than the GPL licence, because it makes it possible to use LGPL code in nonfree software. From the point of view of the user, there is no difference, because in both cases, the software can freely be used and distributed.

All the data that go with Unitex are distributed under the LGPL license ([53]).

Full text versions of LGPL, 2-clause BSD and LGPL license can be found in the appendices of this manual.

1.2 Java runtime environment

Unitex consists of a graphical interface written in Java and external programs written in C/C++. This mixture of programming languages is responsible for a fast and portable application that runs on different operating systems.

Before you can use the graphical interface, you first have to install the runtime environment, usually called Java virtual machine or JRE (Java Runtime Environment).

For the graphical mode, Unitex needs Java version 1.6 (or newer). If you have an older version of Java, Unitex will stop after you have chosen the working language.

You can download the virtual machine for your operating system for free from the Sun Microsystems web site ([68]) at the following address: <http://java.sun.com>.

If you are working under Linux or MacOS, or if you are using a Windows version with personal user accounts, you have to ask your system administrator to install Java.

1.3 Installation on Windows

If Unitex is to be installed on a multi-user Windows machine, it is recommended that the systems administrator performs the installation. If you are the only user on your machine, you can perform the installation yourself.

Decompress the file `unitex_2.1.zip` (You can download this file from the following address: <http://www-igm.univ-mlv.fr/~unitex>) into a directory `Unitex` that should preferably be created within the `Program Files` folder.

After decompressing the file, the `Unitex` directory contains several subdirectories one of which is called `App`. This directory contains a file called `Unitex.jar`. This file is the Java executable that launches the graphical interface. You can double click on this icon to start the program. To facilitate launching Unitex, you may want to add a shortcut to this file on the desktop.

1.4 Installation on Linux

In order to install Unitex on Linux, it is recommended to have system administrator permissions. Uncompress the file `Unitex2.1.zip` in a directory named `Unitex`, by using the following command:

```
unzip Unitex2.1.zip -d Unitex
```

Within the directory `Unitex/Src/C++/build`, start the compilation of Unitex with the command:

```
make install
```

or with the following if you have a 64 bits computer:

```
make install 64BITS=yes
```

You can then create an alias in the following way:

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

1.5 Installation on MacOS X

NOTE: this short tutorial will tell you how to install and run Unitex on Mac OS X. Your questions, comments, suggestions, corrections are more than welcome.

Contact: cedrick.fairon@uclouvain.be

There is an official Java 1.6 for MacOS X 10.5, 64-bit Intel (Core 2 Duo), but there is no official solution for older OS X (10.4 or older), PowerPC and 32-bit Intel (Core Duo). So,

1. if you have OS X 10.5, and 64-bit Intel MacOS, you can just get the Apple JRE 1.6. The only problem is that this version does not start by default. See section "Java for Mac OS X 10.5 Update 2", at <http://developer.apple.com/java/>
2. if you have an older OS X, 32-bit Intel or a PowerPC, you must try SoyLatte (see below)

How to know if my processor is a 32 or 64-bit one ?

In the Apple menu, click on "About this Mac". If you see something like: "Processor : x.xx Ghz Intel Core Duo", your processor is a 32-bit one.

If you see "Processor : x.xx Ghz Intel Core 2 Duo", or if you processor is another Intel one (like Xeon), then you have a 64-bit processor.

1.5.1 Using the Apple Java 1.6 runtime

If you are running Mac OS X 10.5 (or later) on 64-Bits Intel processor, you can just use the Java 1.6 from Apple. You can get it from <http://www.apple.com/support/downloads/javaformacosx105update1.html>.

You can just start Application -> Utilities -> Java Preferences to verify the status of Java 1.6. First, be sure that "Java SE 6" is on "Java Applications" list

Option 1 : modify the default runtime for Java Applications

If you don't use other Java application that need Java 1.5, you can just put "Java SE 6" at the top of the "Java Applications" list on Java Preference Utility.

Option 2 : Create an alias to start Java 1.6

If you don't want modify Java global parameters, you can create an alias:

```
alias jre6="/System/Library/Frameworks/JavaVM.framework/Versions/1.6/Commands/  
jre6 -jar Unitex.jar
```

Then just run Unitex from Terminal.

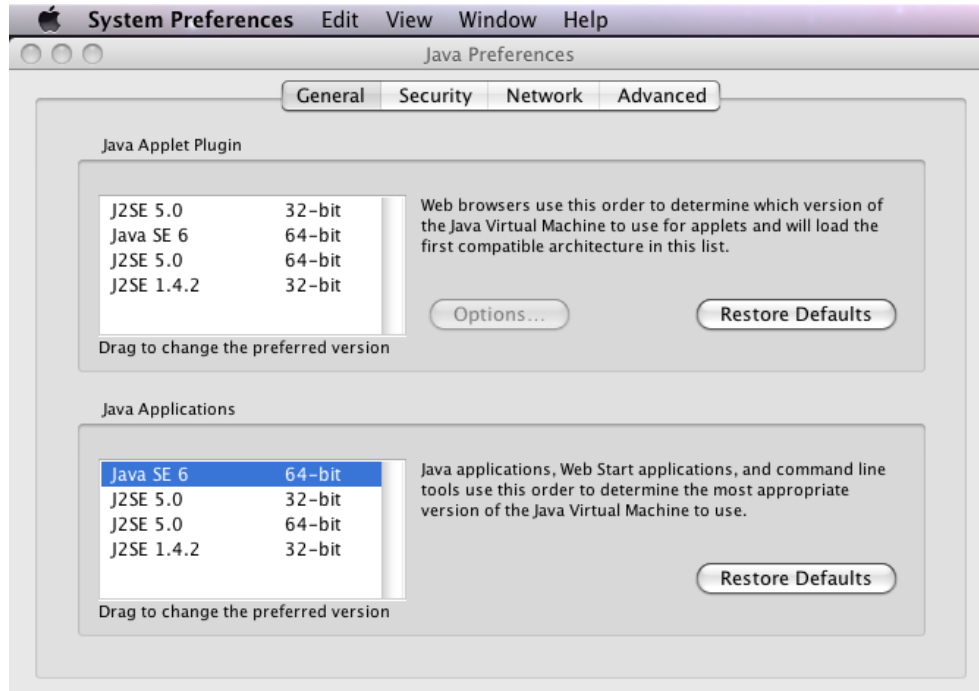


Figure 1.1: Checking and modify Java Preferences

1.5.2 SoyLatte

SoyLatte is a functional, X11-based port of the FreeBSD Java 1.6 patchset to Mac OS X Intel machines. SoyLatte is initially focused on supporting Java 6 development; however, the long-term view far more captivating: open development of Java 7 for Mac OS X, with a release available in concert with the official Sun release, supported on all recent versions of Mac OS X.

Before you start

Try it at your own risks ;-). This tutorial tells you what I have done to have Unitex running on my MacBook Pro (Mac OS X 10.5.7), but it offers no guarantee that it will work on your computer and it does not even guarantee that you can install it safely on your computer.

If you are not familiar with software installation, you should ask for help. As Windows would put it "Contact your Administrator" ;-).

It works only on Intel based Macintosh. If you do not know whether your Macintosh is an Intel machine or not, select in the Apple Menu the item "About this Macintosh", then, click on "More info" and in the next panel, click on "Hardware", as shown on Figure 1.2.

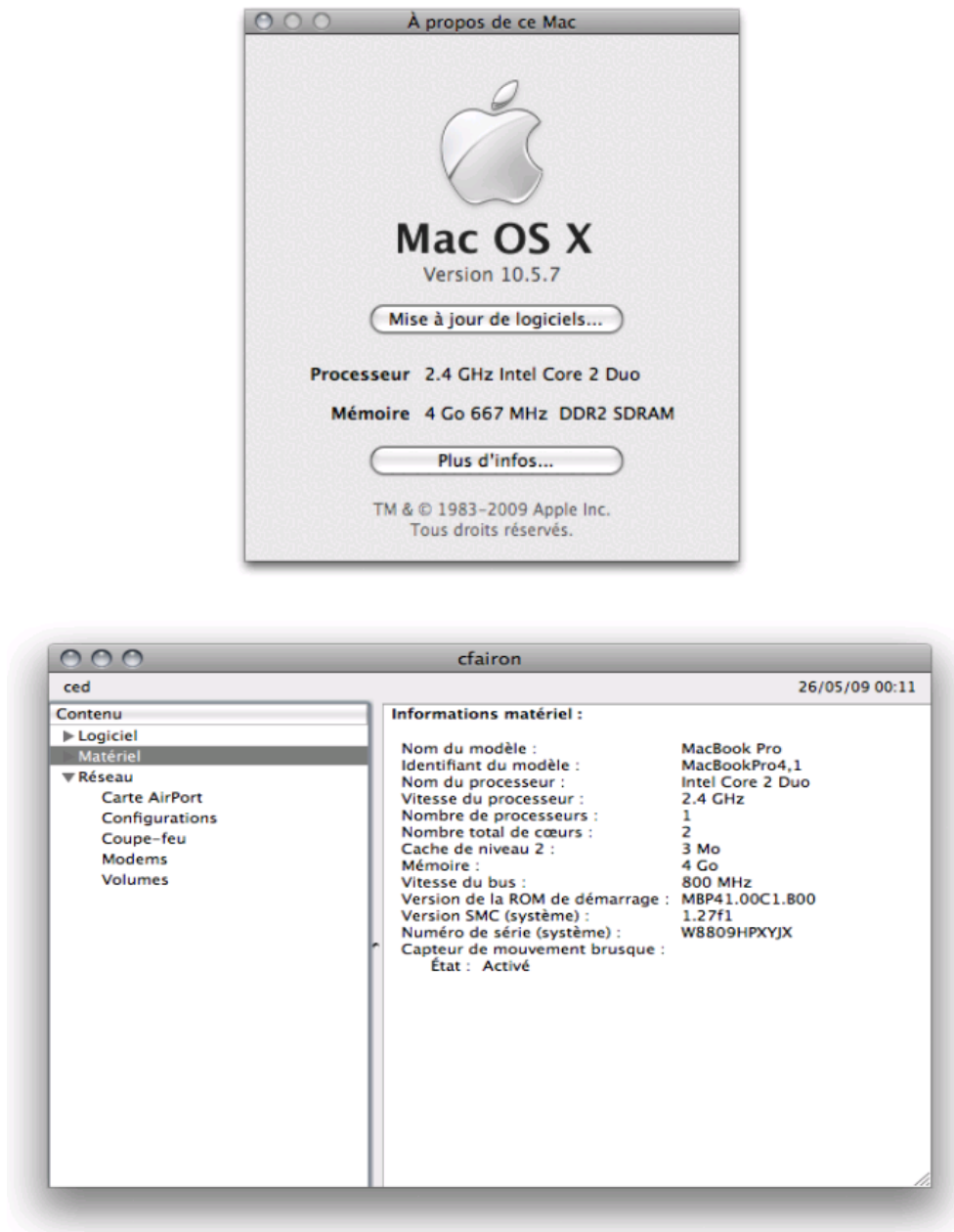


Figure 1.2: Getting information on your computer

Installation

Some of the following installation steps will require the use of the terminal application. On MacOS, the terminal application is located in `/Applications/Utilities/Terminal.app`. When you start this application (double click) a window is displayed. You will have to type in this window some commands for moving, editing and installing files.

Install X11.app

X11 is available as an optional install on the Mac OS X v10.3 Panther, and Mac OS X v10.4 Tiger install disks (the disks you received when you bought your computer). Run the Installer, select the X11 option, and follow the instructions.

After installation X11 will be available in `/Applications/Utilities/`

Download and install Unitex as usual

Download Unitex and uncompress it in the same way that for Linux. See section 1.5.3 for instructions on compiling C++ programs.

Download SoyLatte (the Java 1.6 port)

It is available from <http://landonf.bikemonkey.org/static/soylatte/>

Unless you know why you need another package, you will chose the 32-Bit Binaries distribution (32-bit JDK for Mac OS X 10.4 and 10.5) When prompted, enter the username and password:

```
Username: jrl
Password: I am a Licensee in good standing
```

Install SoyLatte

Either in command line mode:

- Open the terminal (`/Applications/Utilities/Terminal`)
- Find the SoyLatte archive. If it is on your desktop, change the current directory by typing the following command in the terminal (the character `~` means 'home directory'):

```
cd ~/Desktop/
```

- Move (`mv`) the archive anywhere on your file system (where you want to install it). If you chose `/usr/local`, like me, it requires the administrative privileges (use `sudo` and type your password when prompted). Note, if the directory does not exist, you can create it before you move the file:

```
sudo mkdir /usr/local          (only if it does not exist)
sudo mv soylatte16-i386-1.0.3.tar.bz2 /usr/local/
```

- Uncompress the archive:

```
sudo tar -jxvf /usr/local/soylatte16-i386-1.0.3.tar.bz2
```

or, using the graphical interface (finder):

- Move (drag and drop) the archive
`soylatte16-i386-1.0.3.tar.bz2` into the `/Applications` folder
- Double-click on it... wait... it's done ;-)

Configure your system

There are two approaches: you could either decide to use SoyLatte only with Unitex (and keep using the former version of Java that is already installed on your computer for all the others Java applications) or decide to make SoyLatte the default Java distribution on your machine.

As I have no indication that SoyLatte is bug free and fully functional with all applications, I chose to use SoyLatte only with Unitex and kept my Java (and I recommend the same for you). If you want to follow this advice: DO NOT modify your PATH variable as suggested in the SoyLatte installation procedure. Instead add an ALIAS in the configuration file of your shell. Here is how to do that:

- Edit the bash configuration file, which is in your home directory (`/Users/your_dir_name`); The easiest way to do it is to use a command line editor like `vim` or `pico` (note: `.bashrc` is a hidden file. So, normally, you will not see this file in the Finder):

```
pico ~/.bashrc
```

- In the file, type the following lines :

```
alias java6='/usr/local/soylatte16-i386-1.0.3/bin/java'  
alias unitex='cd /Applications/Unitex21beta/App ; java6 -jar  
Unitex.jar'
```

The first line creates an alias that makes it easy to run SoyLatte-Java and the second line creates an alias that makes it easy to start Unitex.

- Then, quit `pico` (type `CTRL+x`, answer "yes" to the question "Do you want to save?" and then press `Enter` to proceed).

Changes will be active next time the `.bashrc` file is loaded (quit and re-launch `Terminal.app` or more simply, type `bash` to open a new bash session). Once you have a new bash session, type the command `unitex` and let's start using Unitex on your Mac !!

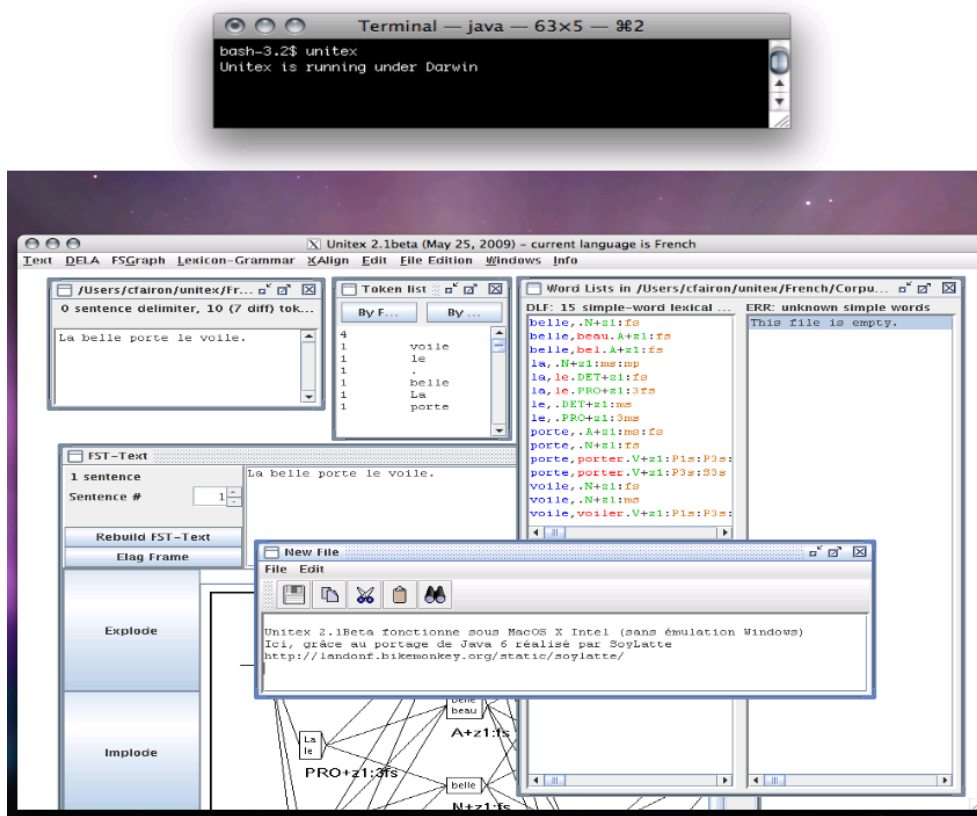


Figure 1.3: Running Unitex on Mac

1.5.3 How to compile Unitex C++ programs on a Macintosh

In order to install Unitex on Mac OS, you will have to compile Unitex C++ sources. This is not a problem if you have already the common (`gcc`) development tools installed on your computer (but of course, they are not in the standard installation).

If you don't know if these tools are present, don't think too long about it... just give a try; open a shell window, move to the Unitex sources directory (`cd /path/to/Src/C++/build`) and run the command that starts the compilation: `make install`

If the compilation does not start and you get an error message stating that the command `make` cannot be found, you probably need to install development programs. On Macintosh the "all inclusive development bundel" is "Xcode" (current version is 2.2 and it includes a lot of stuff among which). You can donwload it from the Appel developer Web site.

The Xcode application includes a full-featured code editor, a debugger, compilers, and a linker. The Xcode application provides a user interface to many industry-standard and open-source tools, including `gcc`, the Java compilers `javac` and `jikes`, and `gdb`. It pro-

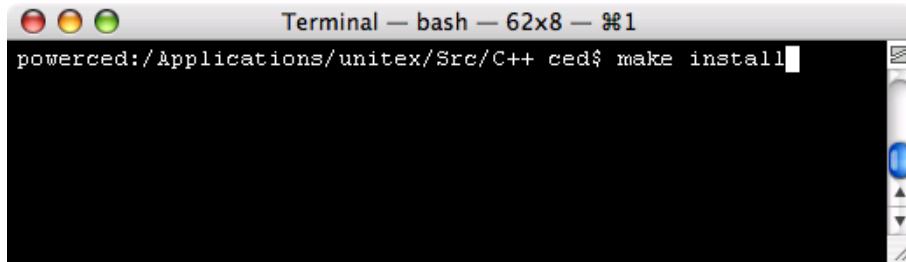


Figure 1.4: Compiling Unitex C++ programs

vides all of the facilities you need to build a program for Mac OS X, whether it's an application, kernel extension, or command-line tool.

The only problem with Xcode is that it is HUGE to download (800 Mb). Of course, you will not need all the stuff included in the package... but all what you need is in it.

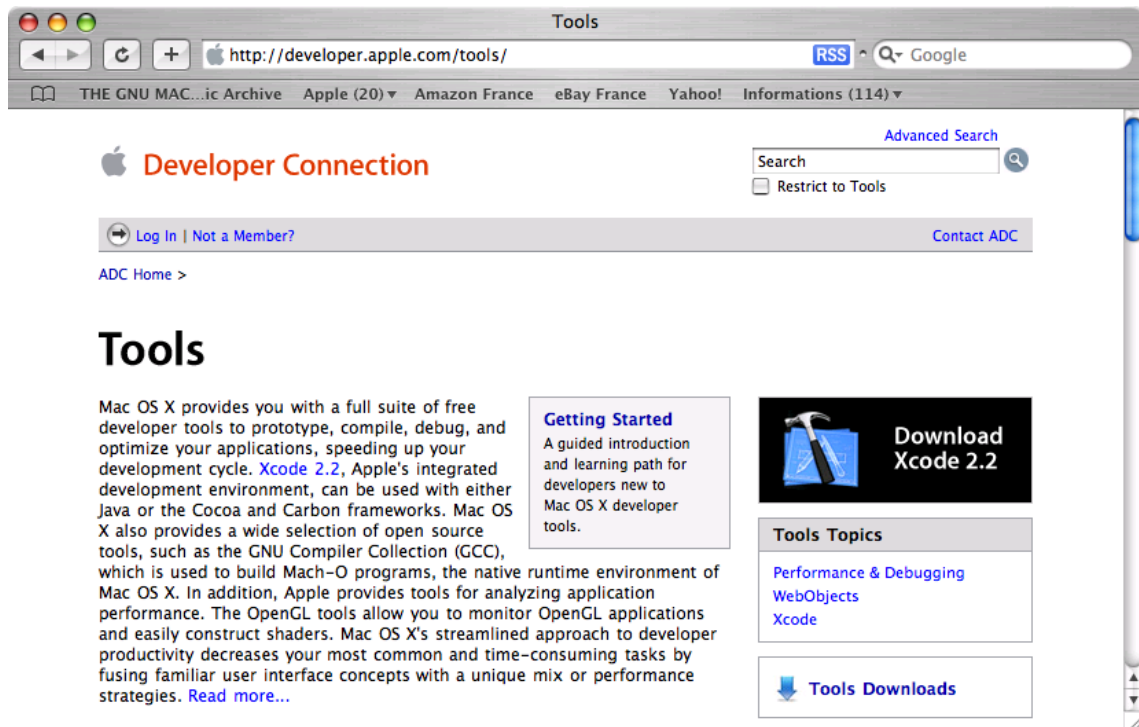


Figure 1.5: Xcode

Once on your computer the Xcode package looks like the following:

Double-click on the `XCodeTools.mpkg` icon to install all the programs.

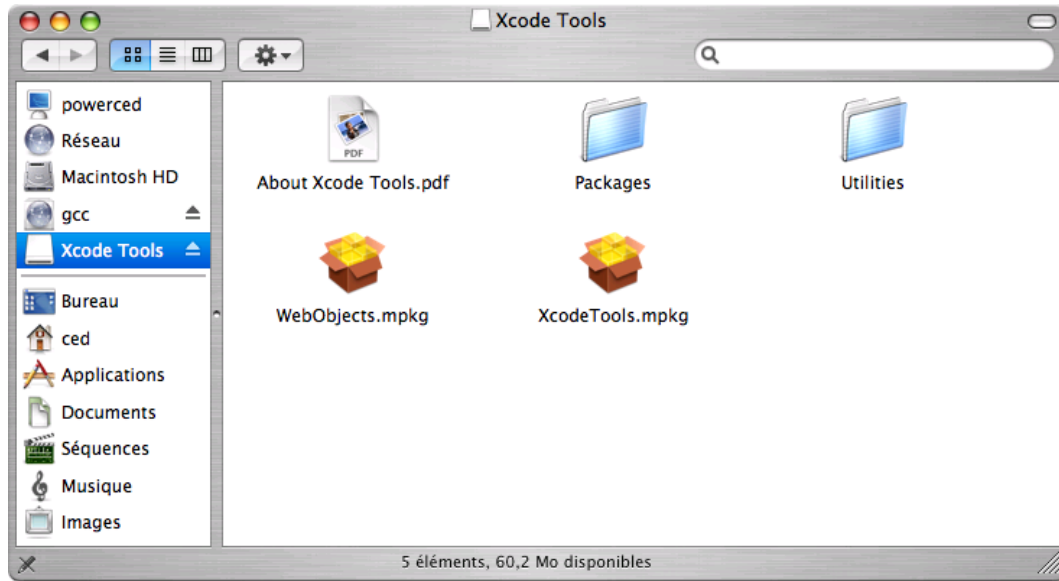


Figure 1.6: Xcode package

1.5.4 How to makes all files visible on Mac OS

See <http://www.macworld.com/article/51830/2006/07/showallfinder.html>.

Or try it right away... Type:

```
defaults write com.apple.Finder AppleShowAllFiles ON
```

Then restart the Finder:

```
killall Finder
```

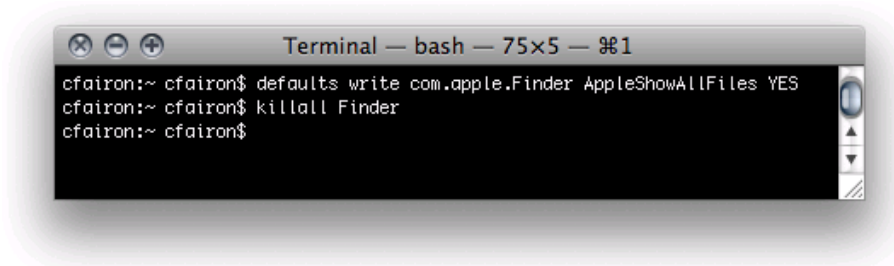


Figure 1.7: Restarting the Finder

To get back to the original configuration, type:

```
defaults write com.apple.Finder AppleShowAllFiles OFF
```

1.6 First use

If you are working on Windows, the program will ask you to choose a personal working directory, which you can change later in "Info>Preferences...>Directories". To create a directory, click on the icon showing a file (see figure 1.10).

If you are using Linux or MacOS, the program will automatically create a `/unitex` directory in your `$HOME` directory. This directory allows you to save your personal data. For each language that you will be using, the program will copy the root directory of that language to your personal directory, except the dictionaries. You can then modify your copy of the files without risking to damage the system files.

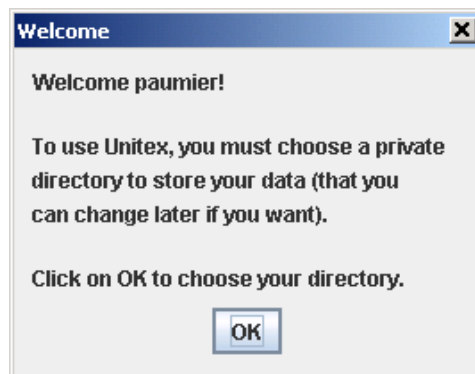


Figure 1.8: First use under Windows



Figure 1.9: First use under Linux

1.7 Adding new languages

There are two different ways to add languages. If you want to add a language that is to be accessible by all users, you have to copy the corresponding directory to the `Unitex` system

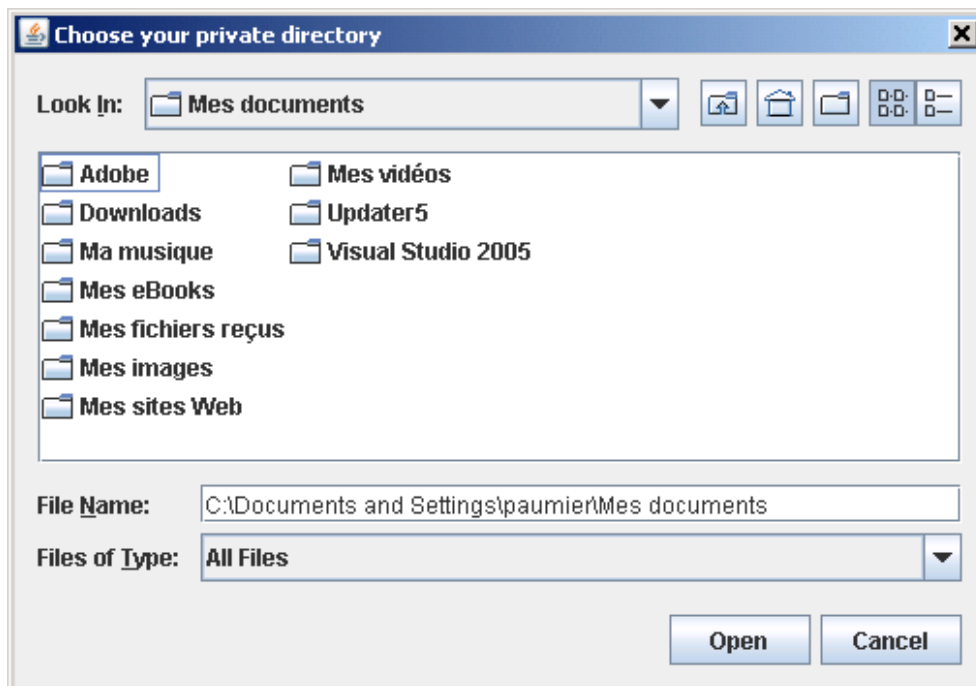


Figure 1.10: Creating the personal work directory

directory, for which you will need to have the access rights (this might mean that you need to ask your system administrator to do it). On the other hand, if the language is only used by a single user, he can also copy the directory to his working directory. He can work with this language without this language being shown to other users.

1.8 Uninstalling Unitex

No matter which operating system you are working with, it is sufficient to delete the `Unitex` directory to completely delete all the program files. Under Windows you may have to delete the shortcut to `Unitex.jar` if you have created one on your desktop. The same has to be done on Linux, if you have created an alias.

1.9 Unitex for developpers

If you are a programmer, you may be interested in linking your code with Unitex C++ sources. To facilitate such operation, you can compile Unitex as a dynamic library that contains all Unitex functions, except `mains`, of course. Under Linux/MacOS, type:

```
make LIBRARY=yes
```

and you will obtain a library named `libunitex.so`. If you want to produce a Windows DLL named `unitex.dll`, use the following commands:

Windows: `make SYSTEM=windows LIBRARY=yes`

Cross-compiling with mingw32: `make SYSTEM=mingw32 LIBRARY=yes`

In all cases, you will also obtain a program named `Test_lib(.exe)`. If everything worked fine, this program should display the following:

```
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
```


Chapter 2

Loading a text

One of the main functionalities of Unitex is to search a text for expressions. To do that, texts have to undergo a set of preprocessing steps that normalize non-ambiguous forms and split the text in sentences. Once these operations are performed, the electronic dictionaries are applied to the texts. Then one can search more effectively in the texts by using grammars.

This chapter describes the different steps for text preprocessing.

2.1 Selecting a language

When starting Unitex, the program asks you to choose the language in which you want to work (see figure 2.1). The languages displayed are the ones that are present in the `Unitex` system directory and those that are installed in your personal working directory. If you use a language for the first time, Unitex copies the system directory for this language to your personal directory, except for the dictionaries in order to save disk space.

WARNING: If you already have a personal directory for a given language, Unitex won't try to copy system data into it. So, if an update has modified a resource file other than a dictionary, you will have to copy by yourself this file, or to delete your personal directory for this language, and let Unitex rebuild it properly.

Choosing the language allows Unitex to find certain files, for example the alphabet file. You can change the language at any time by choosing "Change Language..." in the "Text" menu. If you change the language, the program will close all windows related to the current text, if there are any. The active language is indicated in the title bar of the graphical interface.

2.2 Text formats

Unitex works with Unicode texts. Unicode is a standard that describes a universal character code. Each character is given a unique number, which allows for representing texts without having to take into account the proprietary codes on different machines and/or operating

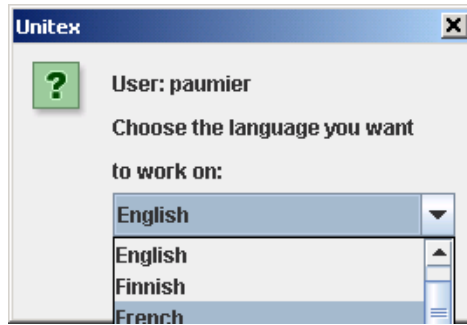


Figure 2.1: Language selection when starting Unitex

systems. Unitex uses a two-byte representation of the Unicode 3.0 standard, called Unicode Little-Endian (for more details, see [16]).

Texts that come with Unitex are already in Unicode format. If you try to open a text that is not in Unicode, the program proposes to convert it (see figure 2.2). This conversion is based on the current language: if you are working in French, Unitex proposes to convert your text¹ assuming that it is coded using a French code page. By default, Unitex proposes to either replace the original text or to rename the original file by inserting `.old` at the beginning of its extension. For example, if one has an ASCII file named `biniou.txt`, the conversion process will create a copy of this ASCII file named `biniou.old.txt`, and will replace the contents of `biniou.txt` with its equivalent in Unicode.



Figure 2.2: Automatic conversion of a non-Unicode text

If the encoding suggested by default is not correct or if you want to rename the file differently than with the suffix `.old`, you must use the "More options..." button. This allows you to choose source and target encodings of the documents to be converted (see figure 2.3). By default, the selected source encoding is that which corresponds to the current language and the destination encoding is Unicode Little-Endian. You can modify these choices by selecting any source and target encodings. Thus, if you wish, you can convert your data into other

¹Unitex also proposes to automatically convert graphs and dictionaries that are not in Unicode Little-Endian.

encodings, as for example UTF-8 in order for instance to create web pages. The button "Add Files" enables you to select the files to be converted. The button "Remove Files" makes it possible to remove a list of files erroneously selected. The button "Transcode" will start the conversion of all the selected files. If an error occurs with a file is processed (for example, a file which is already in Unicode), the conversion continues with the next file.

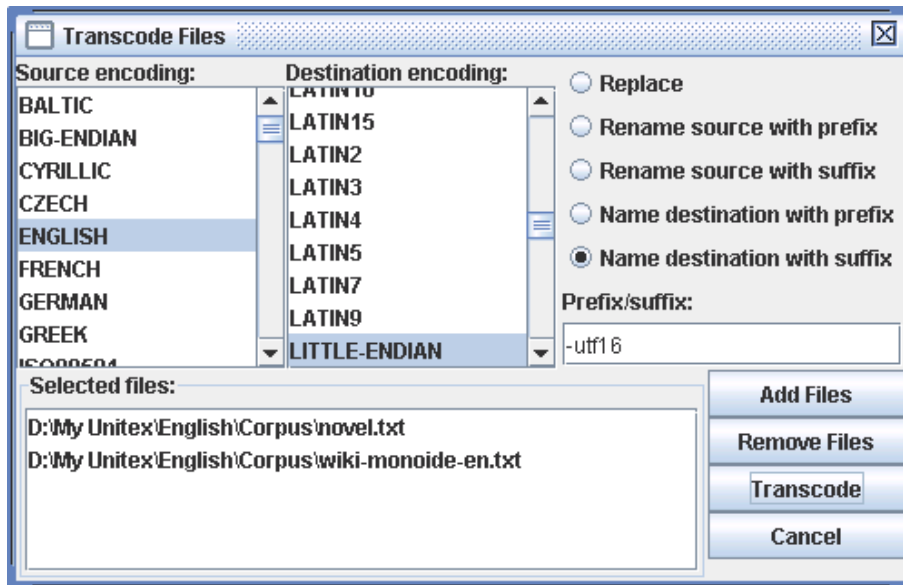


Figure 2.3: Transcoding files

To obtain a text in the right format, you can also use a text processor like the free software from OpenOffice.org ([72]) or Microsoft Word, and save your document with the format "Unicode text". In OpenOffice Writer, you have to choose the "Coded Text (*.txt)" format and then select the "Unicode" encoding in the configuration window as shown on figure 2.4.

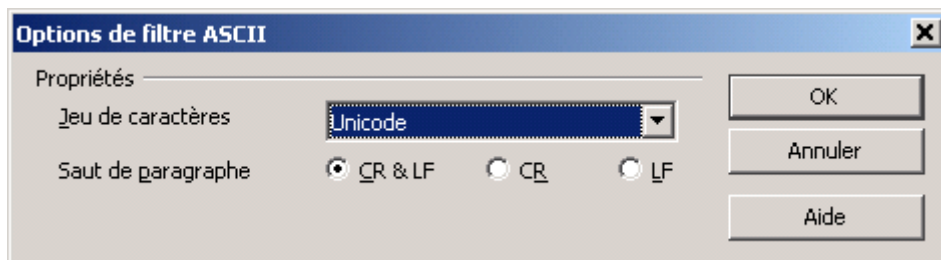


Figure 2.4: Saving in Unicode with OpenOffice Writer

By default, the encoding proposed on a PC is always Unicode Little-Endian. The texts thus obtained do not contain any formatting information anymore (fonts, colors, etc.) and are ready to be used with Unitex.

2.3 Editing text files

For small texts, you also have the possibility of using the text editor integrated into Unitex, accessible via the "Open..." command in the "File Edition" menu". This editor offers search and replace functionalities for the texts and dictionaries handled by Unitex. To use it, click on the "Find" icon. You will then see a window divided into three parts. The "Find" part corresponds to the usual search operations. If you open a text split into sentences, you can base your search on sentence numbers in the "Find Sentence" part. Lastly, the "Search Dictionary" part, visible in figure 2.5, enables you to carry out operations concerning the electronic dictionaries. In particular, you can search by specifying if it concerns inflected forms, lemmas, grammatical and semantic and/or inflectional codes. Thus, if you want to search for all the verbs which have the semantic feature t , which indicates transitivity, you just have to search for t by clicking on "Grammatical code". You will get the matching entries without confusion with all the other occurrences of the letter t .

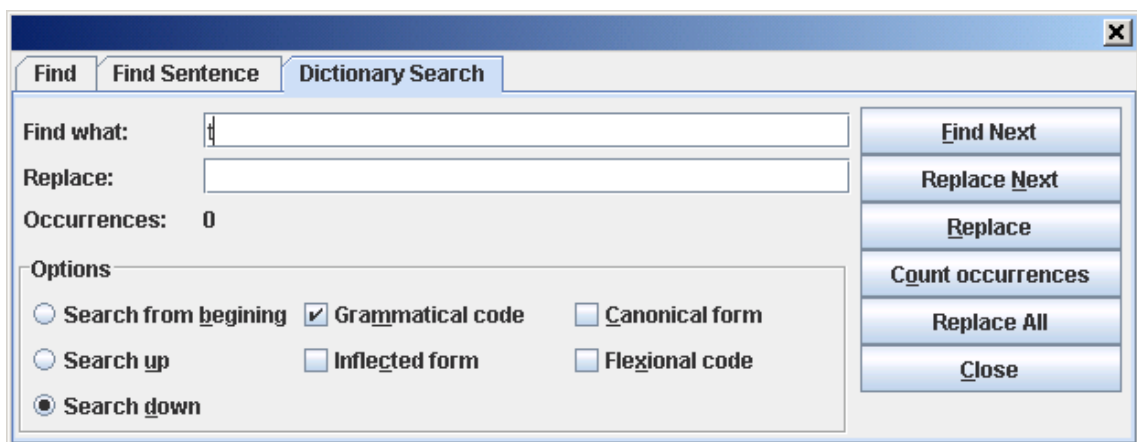


Figure 2.5: Searching an electronic dictionary for the semantic feature t

2.4 Opening a text

Unitex deals with two types of text files. The files with the extension `.snt` are text files pre-processed by Unitex which are ready to be manipulated by the different system functions. The files ending with `.txt` are raw files. To use a text, open the `.txt` file by clicking on "Open..." in the "Text" menu. Choose the file type "Raw Unicode Texts" and select your text.

2.5 Preprocessing a text

After a text is selected, Unitex offers to preprocess it. Text preprocessing consists of performing the following operations: normalization of separators, splitting into sentences, normalization of non-ambiguous forms, tokenization and application of dictionaries. If you choose

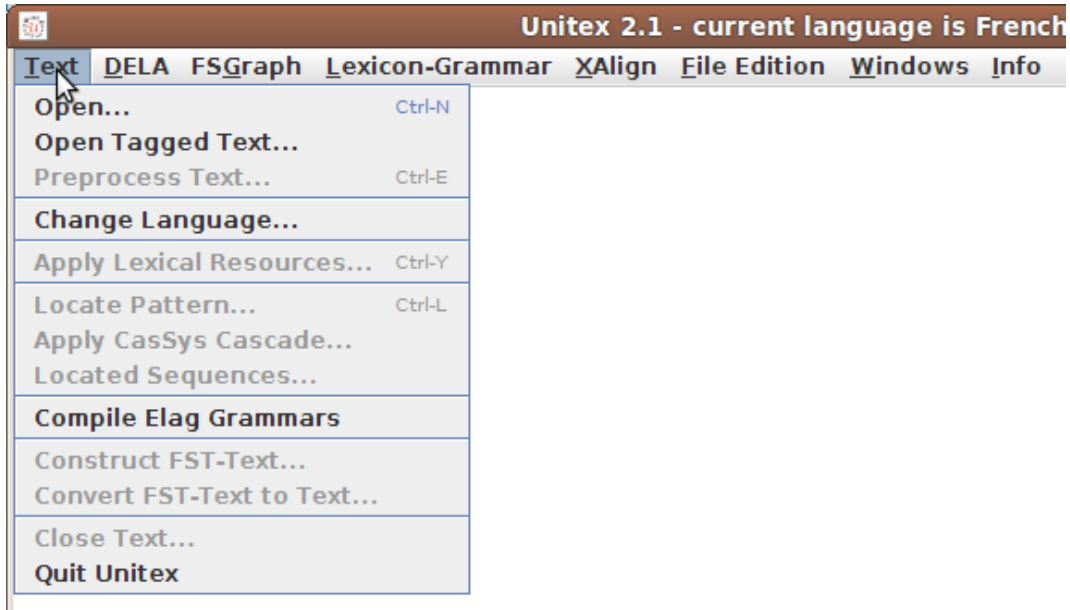


Figure 2.6: Text Menu

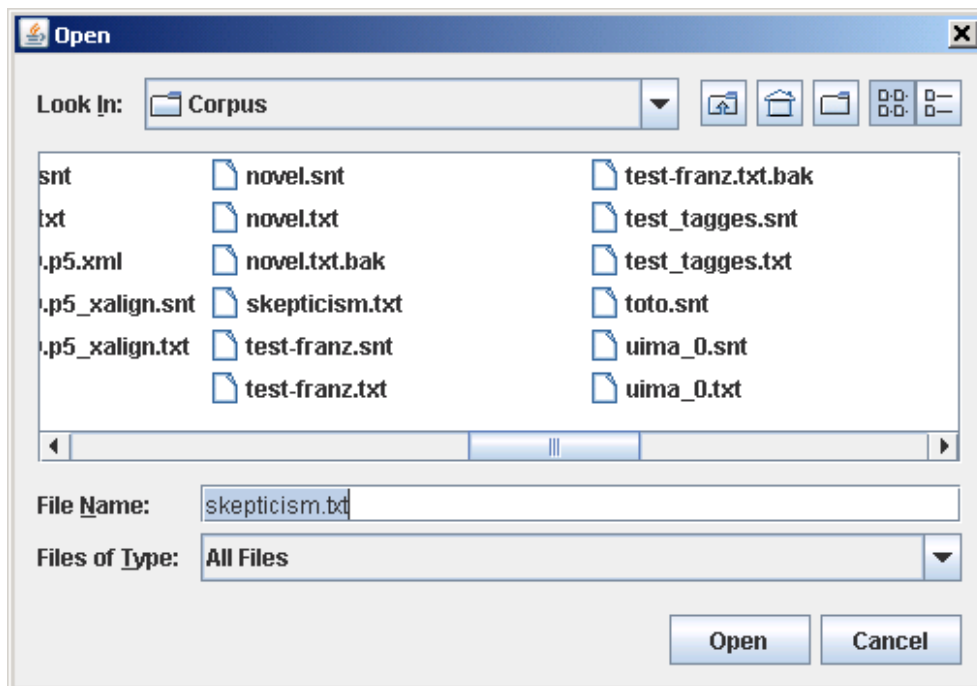


Figure 2.7: Opening a Unicode text

not to preprocess the text, it will nevertheless be normalized and tokenized, since these operations are necessary for all further Unitex operations. It is always possible to carry out the preprocessing later by clicking on "Preprocess Text..." in the "Text" menu.

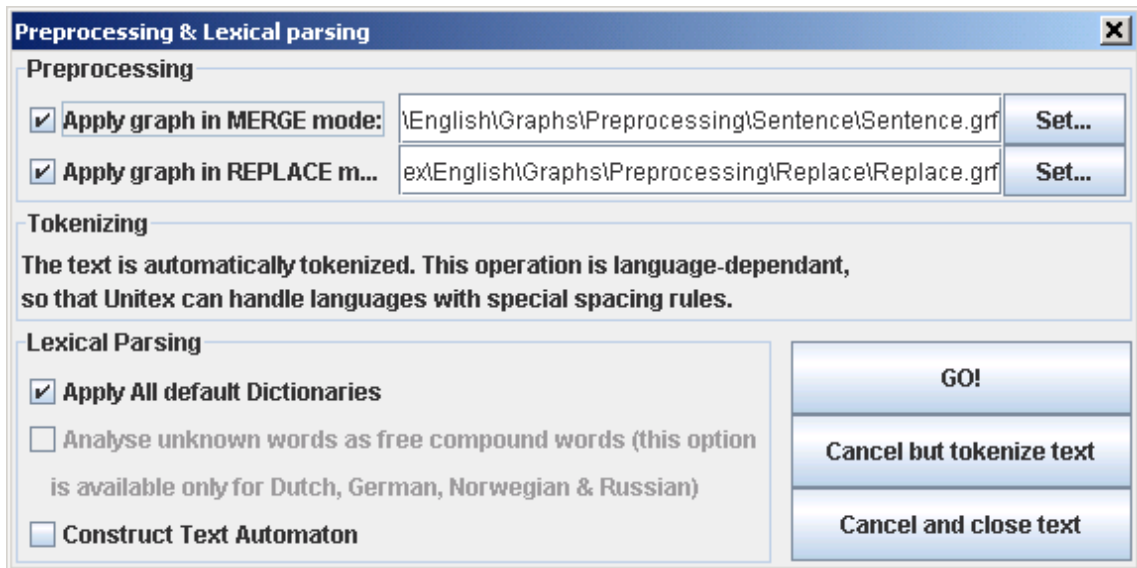


Figure 2.8: Preprocessing Window

If you choose to preprocess the text, Unitex proposes to parameterize it as in the window shown in figure 2.8. The option "Apply FST2 in MERGE mode" is used to split the text into sentences. The option "Apply FST2 in REPLACE mode" is used to make replacements in the text, especially for the normalization of non-ambiguous forms. With the option "Apply All default Dictionaries" you can apply dictionaries in the DELA format (Dictionnaires Electroniques du LADL). The option "Analyze unknown words as free compound words" is used in Norwegian for correctly analyzing compound words constructed via concatenation of simple forms. Finally, the option "Construct Text Automaton" is used to build the text automaton. This option is deactivated by default, because it consumes a large amount of memory and disk space if the text is too large. The construction of the text automaton is described in chapter 7.

NOTE: If you click on "Cancel but tokenize text", the program will carry out the normalization of separators and split the text into tokens. Click on "Cancel and close text" to cancel the operation.

2.5.1 Normalization of separators

The standard separators are the space, the tab and the newline characters. There can be several separators following each other, but since this isn't useful for linguistic analyses, separators are normalized according to the following rules:

- a sequence of separators that contains at least one newline is replaced by a single newline
- all other sequences of separators are replaced by a single space.

The distinction between space and newline is maintained at this point because the presence of newlines may have an effect on the process of splitting the text into sentences. The result of the normalization of a text named `my_text.txt` is a file in the same directory as the `.txt` file and is named `my_text.snt`.

NOTE: When the text is preprocessed using the graphical interface, a directory named `my_text_snt` is created immediately after normalization. This directory, called text directory, contains all the data associated with this text.

2.5.2 Splitting into sentences

Splitting texts into sentences is an important preprocessing step since this helps in determining the units for linguistic processing. The splitting is used by the text automaton construction program. In contrast to what one might think, detecting sentence boundaries is not a trivial problem. Consider the following text:

The family has urgently called Dr. Martin.

The full stop that follows *Dr* is followed by a word beginning with a capital letter. Thus it may be considered as the end of the sentence, which would be wrong. To avoid the kind of problems caused by the ambiguous use of punctuation, grammars are used to describe the different contexts for the end of a sentence. Figure 2.9 shows an example grammar for sentence splitting (for French sentences).

When a path of the grammar recognizes a sequence in the text and when this path produces the sentence delimiter symbol $\{S\}$, this symbol is inserted into the text.

The path shown at the top of figure 2.9 recognizes the sequence consisting of a question mark and a word beginning with a capital letter and inserts the symbol $\{S\}$ between the question mark and the following word. The following text:

What time is it? Eight o' clock.

will be converted to:

What time is it ?{S} Eight o' clock.

A grammar for end-of-sentence detection may use the following special symbols:

- $\langle E \rangle$: empty word, or epsilon. Recognizes the empty sequence;
- $\langle MOT \rangle$: recognizes any sequence of letters;

- `<MIN>` : recognizes any sequence of letters in lower case;
- `<MAJ>` : recognizes any sequence of letters in upper case;
- `<PRE>` : recognizes any sequence of letters that begins with an upper case letter;
- `<NB>` : recognizes any sequence of digits (1234 is recognized but not 1 234);
- `<PNC>` : recognizes the punctuation symbols ; , ! ? : and the inverted exclamation points and question marks in Spanish and some Asian punctuation letters;
- `<^>` : recognizes a newline;
- `#` : prohibits the presence of a space.

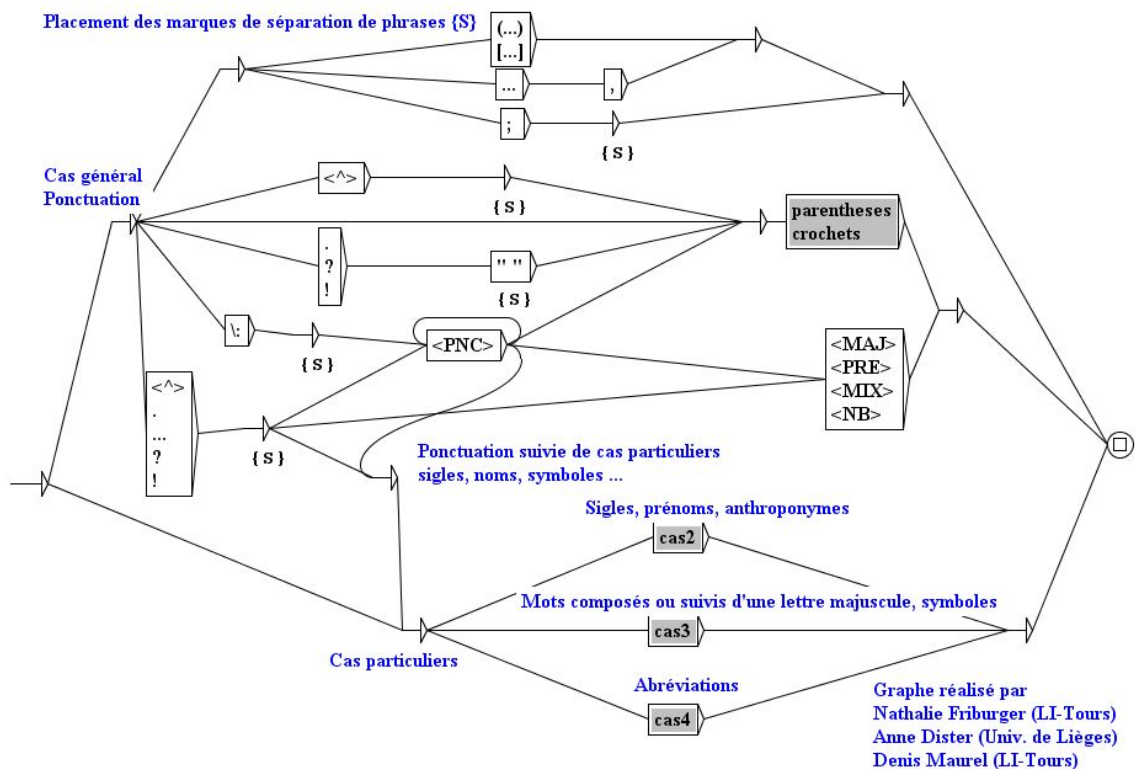


Figure 2.9: Sentence splitting grammar for French

By default, the space is optional between two boxes. If you want to prohibit the presence of the space you have to use the special character `#`. At the opposite, if you want to force the presence of the space, you must use the sequence `" "`. Lower and upper case letters are defined by an alphabet file (see chapter 13). For more details on grammars, see chapter 5. For more information about sentence boundary detection, see [21]. The grammar used here is named `Sentence.fst2` and can be found in the following directory:

```
/(user home directory)/(language)/Graphs/Preprocessing/Sentence
```

This grammar is applied to a text with the `Fst2Txt` program in MERGE mode. This has the effect that the output produced by the grammar, in this case the symbol `{S}`, is inserted into the text. This program takes a `.snt` file and modifies it.

2.5.3 Normalization of non-ambiguous forms

Certain forms present in texts can be normalized (for example, the English sequence `"I'm"` is equivalent to `"I am"`). You may want to replace these forms according to your own needs. However, you have to be careful that the forms normalized are unambiguous or that the removal of ambiguity has no undesirable consequences.

For instance, if you want to normalize `"O'clock"` to `"on the clock"`, it would be a bad idea to replace `"O"` by `"on the "`, because a sentence like:

John O'Connor said: "it's 8 O'clock"

would be replaced by the following incorrect sentence:

John on the Connor said: "it's 8 on the clock"

Thus, one needs to be very careful when using the normalization grammar. One needs to pay attention to spaces as well. For example, if one replaces `"'re"` by `"are"`, the sentence:

You're stronger than him.

will be replaced by:

Youare stronger than him.

To avoid this problem, one should explicitly insert a space, *i.e.* replace `"'re"` by `" are"`.

The accepted symbols for the normalization grammar are the same as the ones allowed for the sentence splitting grammar. The normalization grammar is called `Replace.fst2` and can be found in the following directory:

```
/(home directory)/(active language)/Graphs/Preprocessing/Replace
```

As in the case of sentence splitting, this grammar is applied using the `Fst2Txt` program, but in REPLACE mode, which means that input sequences recognized by the grammar are replaced by the output sequences that are produced. Figure 2.10 shows a grammar that normalizes verbal contractions in English.

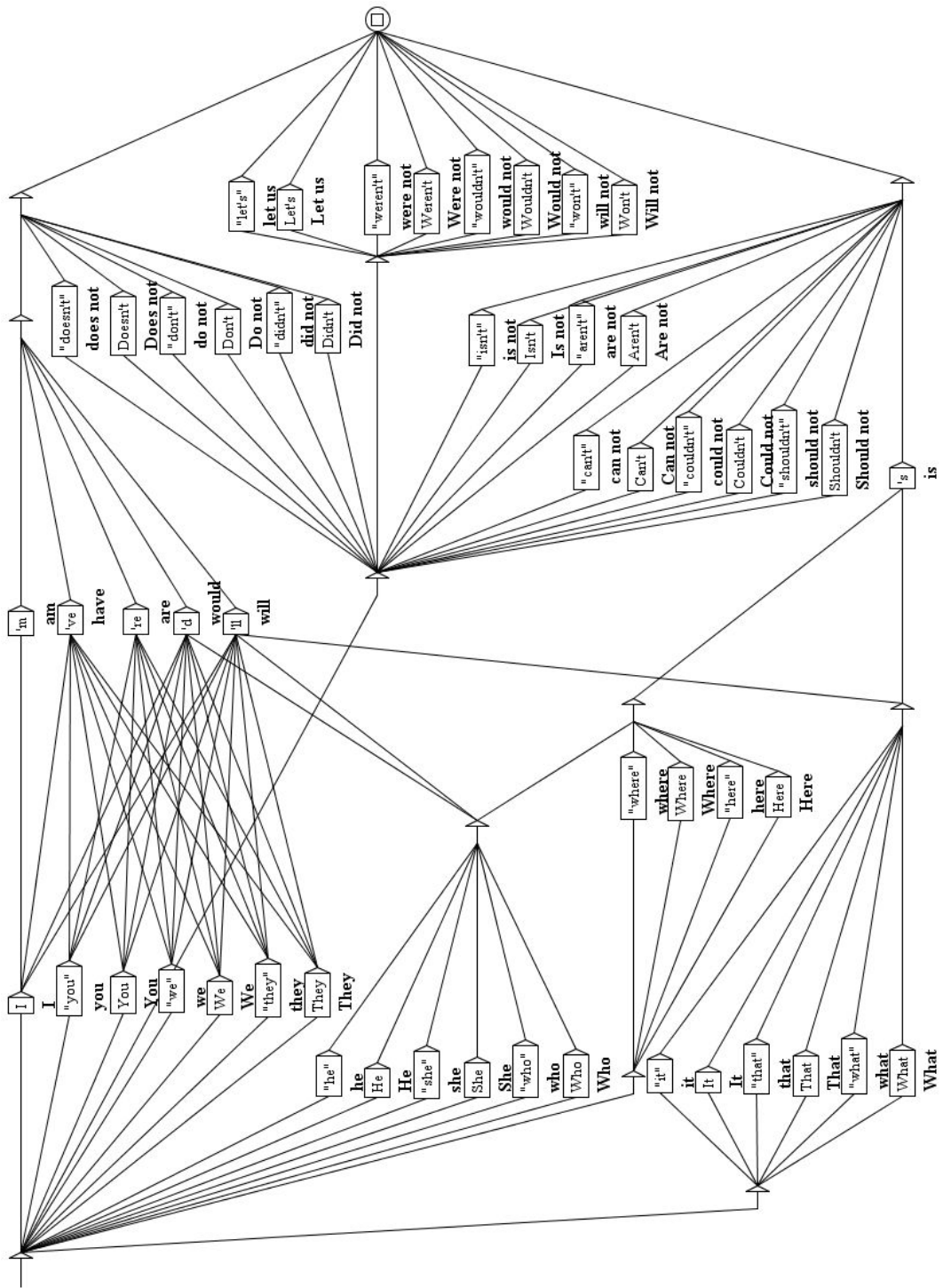


Figure 2.10: Normalization of English verbal contractions

2.5.4 Splitting a text into tokens

Some languages, in particular Asian languages, use separators that are different from the ones used in western languages. Spaces can be forbidden, optional, or mandatory. In order to better cope with these particularities, Unitex splits texts in a language dependent way. Thus, languages like English are treated as follows:

A token can be:

- the sentence delimiter `{S}`;
- the stop marker `{STOP}`. This token is a special one that can NEVER be matched in any way by a grammar. It can be used to bound elements in a corpus. For instance, if a corpus is made of news separated by `{STOP}`, it will be impossible that a grammar matches a sequence that overlaps the end of a news and the beginning of the following news;
- a lexical tag `{aujourd'hui, .ADV}`;
- a contiguous sequence of letters (the letters are defined in the language alphabet file);
- one (and only one) non-letter character, i.e. all characters not defined in the alphabet file of the current language; if it is a newline, it is replaced by a space.

For other languages, tokenization is done on a character by character basis, except for the sentence delimiter `{S}`, the `{STOP}` marker and lexical tags. This simple tokenization is fundamental for the use of Unitex, but limits the optimization of search operations for patterns.

Regardless of the tokenization mode, newlines in a text are replaced by spaces. Tokenization is done by the `Tokenize` program. This program creates several files that are saved in the text directory:

- `tokens.txt` contains the list of tokens in the order in which they are found in the text;
- `text.cod` contains an integer array; every integer corresponds to the index of a token in the file `tokens.txt`;
- `tok_by_freq.txt` contains the list of tokens sorted by frequency;
- `tok_by_alph.txt` contains the list of tokens in alphabetical order;
- `stats.n` contains some statistics about the text.

Tokenizing the text:

A cat is a cat.

returns the following list of tokens: *A SPACE cat is a .*

You will observe that tokenization is case sensitive (*A* and *a* are two distinct tokens), and that each token is listed only once. Numbering these tokens from 0 to 5, the text can be represented by a sequence of numbers (integers) as described in the following table:

Token number	0	1	2	1	3	1	4	1	2	5
Corresponding token	<i>A</i>		<i>cat</i>		<i>is</i>		<i>a</i>		<i>cat</i>	<i>.</i>

Table 2.1: Representation of the text *A cat is a cat.*

For more details, see chapter [13](#).

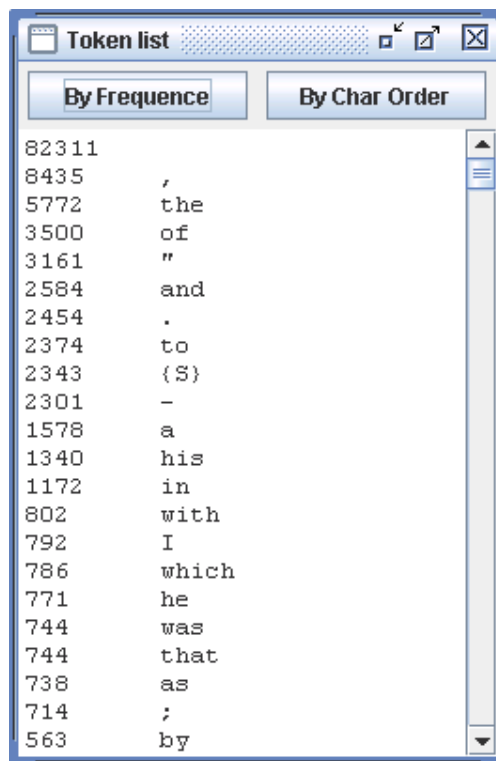


Figure 2.11: Tokens of an English text sorted by frequency

2.5.5 Applying dictionaries

Applying dictionaries consists of building the subset of dictionaries consisting only of forms that are present in the text. Thus, the result of applying an English dictionary to the text *Igor's father in law is ill* produces a dictionary of the following simple words:

```

father, .N+Hum:s
father, .V:W:P1s:P2s:P1p:P2p:P3p
ill, .A
ill, .ADV
ill, .N:s
in, .A
in, .N:s
in, .PART
in, .PREP
is, be.V:P3s
is, i.N:p
law, .N:s
law, .V:W:P1s:P2s:P1p:P2p:P3p
s, .N:s

```

as well as a dictionary of compound words consisting of a single entry:

```
father in law, .N+NPN+Hum+z1:s
```

Since the sequence *Igor* is neither a simple English word nor a part of a compound word, it is treated as an unknown word. The application of dictionaries is done through the program `Dico`. The three files produced (`d1f` for simple words, `d1c` for compound words and `err` for unknown words) are placed in the text directory. The `d1f` and `d1c` files are called text dictionaries.

As soon as the dictionary look-up is finished, Unitex displays the sorted lists of simple, compound and unknown words found in a new window. Figure 2.12 shows the result for an English text.

It is also possible to apply dictionaries without preprocessing the text. In order to do this, click on "Apply Lexical Resources..." in the "Text" menu. Unitex then opens a window (see figure 2.13) in which you can select the list of dictionaries to apply.

The list "User resources" lists all dictionaries present in the directory `(current language)/Dela` of the user. The dictionaries installed in the system are listed in the scroll list named "System resources". Use the `<Ctrl+click>` combination to select several dictionaries. System dictionaries will be applied prior to user dictionaries. Within the system or user list, you can fix the order of dictionaries using the up and down arrows, as shown on figure 2.13. The button "Set Default" allows you to define the current selection of dictionaries as the default. This default selection will then be used during preprocessing if you activate the option "Apply All default Dictionaries". If you right-click on a dictionary name, the associated documentation, if any, will be displayed in the lower frame of the window.

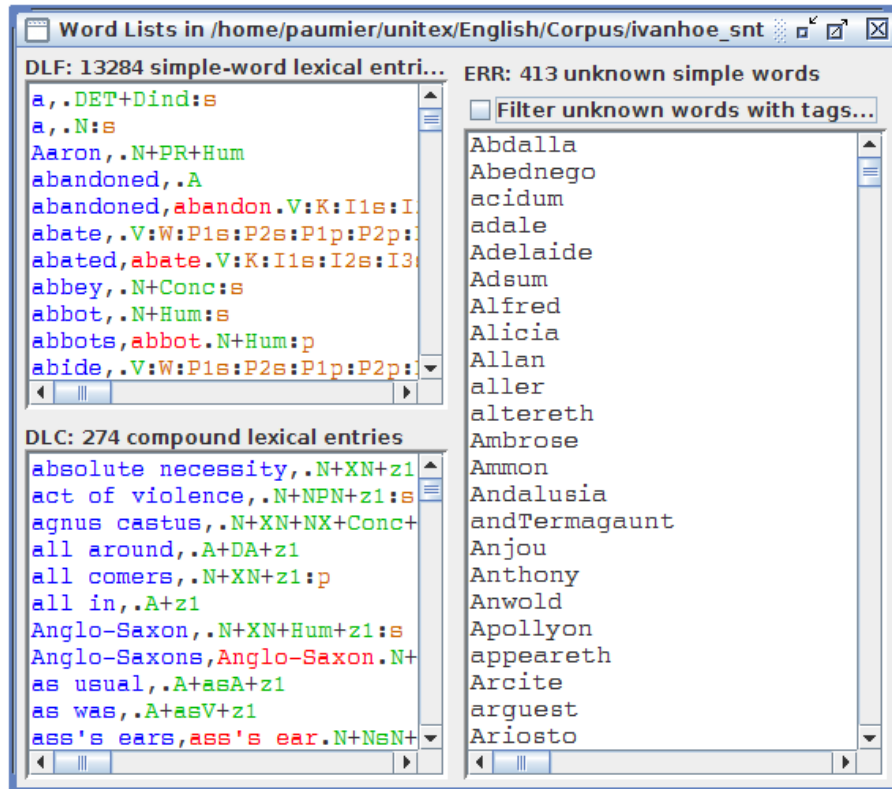


Figure 2.12: Result after applying dictionaries to an English text

2.5.6 Analysis of compound words in Dutch, German, Norwegian and Russian

In certain languages like Norwegian, German and others, it is possible to form new compound words by concatenating together other words. For example, the word *aftenblad* meaning *evening journal* is obtained by combining the words *aften* (*evening*) et *blad* (*journal*). The PolyLex program parses the list of unknown words after the application of dictionaries and tries to analyze each of these words as a compound word. If a word has at least one analysis as a compound word, it is removed from the list of unknown words and the lines produced for this word are appended to the simple word text dictionary.

2.6 Opening a tagged text

A tagged text is a text containing words with lexical tags enclosed in round brackets:

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

Such tags can be used to avoid ambiguities. In the previous example, it will be impossible to match *square bracket* as the combination of two simple words.

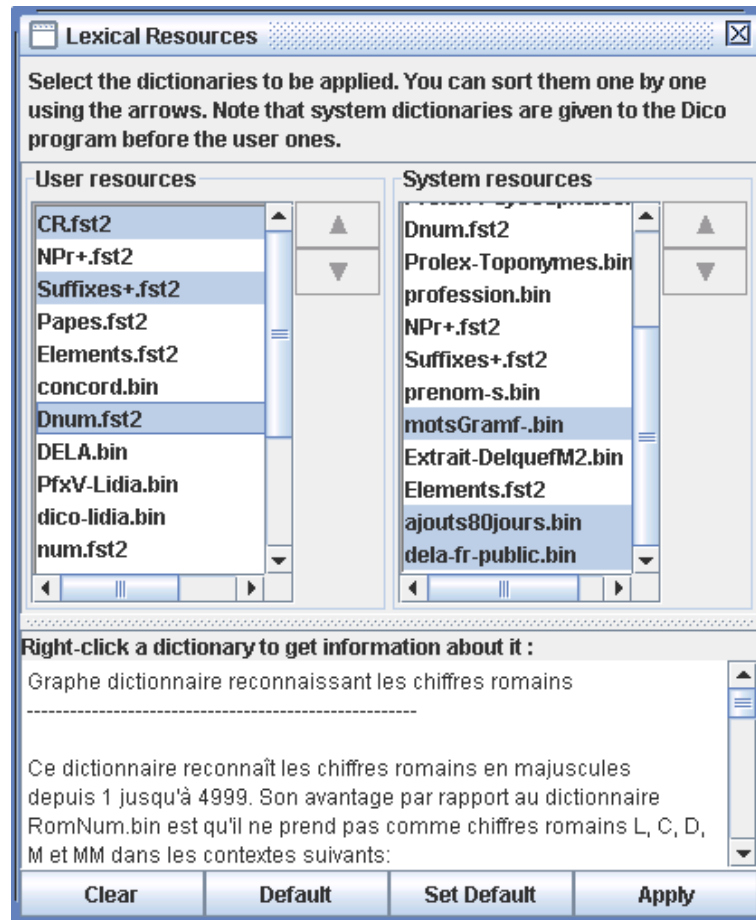


Figure 2.13: Parameterizing the application of dictionaries

However, the presence of these tags can alter the application of preprocessing graphs. To avoid complications, you can use the "Open Tagged Text..." command in the "Text" menu. With it, you can open a tagged text and skip the application of preprocessing graphs, as shown on Figure 2.14.

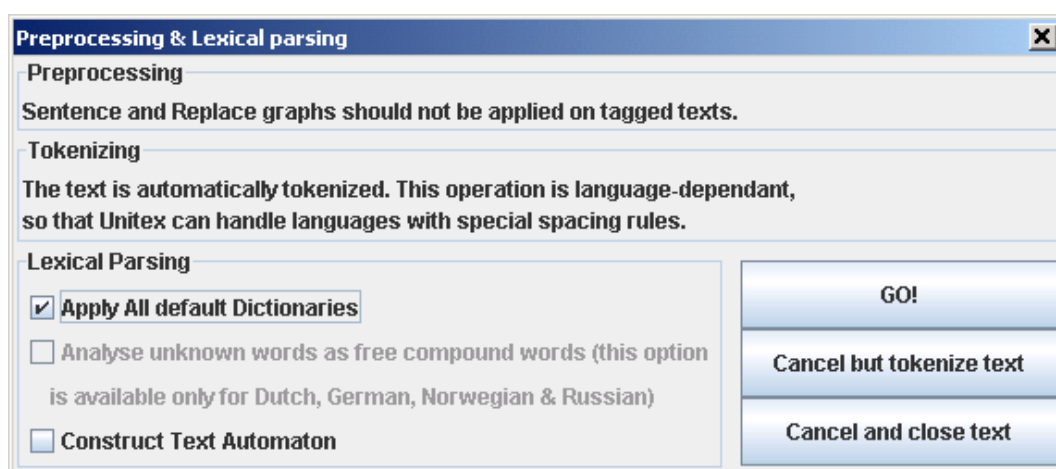


Figure 2.14: Preprocessing a tagged text

Chapter 3

Dictionaries

3.1 The DELA dictionaries

The electronic dictionaries distributed with Unitex use the DELA syntax (Dictionnaires Electroniques du LADL, LADL electronic dictionaries). This syntax describes the simple and compound lexical entries of a language with their grammatical, semantic and inflectional information. We distinguish two kinds of electronic dictionaries. The one that is used most often is the dictionary of inflected forms DELAF (DELA de formes Fléchies, DELA of inflected forms) or DELACF (DELA de formes Composées Fléchies, DELA of compound inflected forms) in the case of compound forms. The second type is a dictionary of non-inflected forms called DELAS (DELA de formes simples, simple forms DELA) or DELAC (DELA de formes composées, compound forms DELA).

Unitex programs make no distinction between simple and compound form dictionaries. We will use the terms DELAF and DELAS to distinguish the inflected and non-inflected dictionaries, no matter they contain simple word, compound words or both.

3.1.1 The DELAF format

Entry syntax

An entry of a DELAF is a line of text terminated by a newline that conforms to the following syntax:

```
apples,apple.N+conc:p/this is an example
```

The different elements of this line are:

- `apples` is the inflected form of the entry; it is mandatory;

- `apple` is the canonical form (lemma) of the entry. For nouns and adjectives (in French), it is usually the masculine singular form; for verbs, it is the infinitive. This information may be left out as in the following example:

```
apple, .N+Conc:s
```

This means that the canonical form is the same as the inflected form. The canonical form is separated from the inflected form by a comma.

- `N+Conc` is the sequence of grammatical and semantic information. In our example, `N` designates a noun, and `Conc` indicates that this noun designates a concrete object (see table 3.2).

Each entry must have at least one grammatical or semantic code, separated from the canonical form by a period. If there are more codes, these are separated by the `+` character.

- `:p` is an inflectional code which indicates that the noun is plural. Inflectional codes are used to describe gender, number, declination, and conjugation. This information is optional. An inflectional code is made up of one or more characters that represent one information each. Inflectional codes have to be separated by the `:` character, for instance in an entry like the following:

```
hang, .V:W:P1s:P2s:P1p:P2p:P3p
```

The `:` character is interpreted in OR semantics. Thus, `:W:P1s:P2s:P1p:P2p:P3p` means "infinitive", or "1st person singular present", or "2nd person singular present", etc. (see table 3.3) Since each character represents one information, you must not use the same character more than once. In this way, encoding the past participle using the code `:PP` would be exactly equivalent to using `:P` alone;

- `/this is an example` is a comment. Comments are optional and are introduced by the `/` character. These comments are left out when the dictionaries are compressed.

IMPORTANT REMARK: It is possible to use the full stop and the comma within a dictionary entry. In order to do this they have to be escaped using the `\` character:

```
1\,000,one thousand.NUMBER
United Nations,U\.N\..ACRONYM
```

WARNING: Each character is taken into account within a dictionary line. For example, if you insert spaces, they are considered to be a part of the information. In the following line:

hath,have.V:P3s /old form of 'has'

The space that precedes the / character will be considered to be part of a 4-character inflectional code.

It is possible to insert comments into a DELAF or DELAS dictionary by starting the line with a / character. Example:

```
/ 'English' designates a pool spin
English, .N+z3:s
```

Compound words with spaces or dashes

Certain compound words like *acorn-shell* can be written using spaces or dashes. In order to avoid duplicating the entries, it is possible to use the = character. At the time when the dictionary is compressed, the Compress program checks for each line if the inflected or canonical form contains a non-escaped = character. If this is the case, the program replaces this by two entries: one where the = character is replaced by a space, and one where it is replaced by a dash. Thus, the following entry:

```
acorn=shells,acorn=shell.N:p
```

is replaced by the following entries:

```
acorn shells,acorn shell.N:p
acorn-shells,acorn-shell.N:p
```

NOTE: If you want to keep an entry that includes the = character, escape it using \ as in the following example:

```
E\=mc2, .FORMULA
```

This replacement is done when the dictionary is compressed. In the compressed dictionary, the escaped = characters are replaced by simple =. As such, if a dictionary containing the following lines is compressed:

```
E\=mc2, .FORMULA
acorn=shell, .N:s
```

and if the dictionary is applied to the following text:

Formulas like E=mc2 have nothing to do with acorn-shells.

you will get the following lines in the dictionary of compound words of the text:

```
E=mc2, .FORMULA
acorn-shells, .N:p
```

Entry Factorization

Several entries containing the same inflected and canonical forms can be combined into a single one if they also share the same grammatical and semantic codes. Among other things this allows us to combine identical conjugations for a verb:

```
bottle, .V:W:P1s:P2s:P1p:P2p:P3p
```

If the grammatical and semantic information differ, one has to create distinct entries:

```
bottle, .N+Conc:s  
bottle, .V:W:P1s:P2s:P1p:P2p:P3p
```

Some entries that have the same grammatical and semantic entries can have different meanings, as it is the case for the French word *poêle* that describes a stove or a type of sheet in the masculine sense and a kitchen instrument in the feminine sense. You can thus distinguish the entries in this case:

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

NOTE: In practice, this distinction has the only consequence that the number of entries in the dictionary increases.

For the different programs that make up Unix these entries are equivalent to:

```
poêle, .N+z1:fs:ms
```

Whether this distinction is made is thus left to the maintainers of the dictionaries.

3.1.2 The DELAS Format

The DELAS format is very similar to the one used in the DELAF. The only difference is that there is only a canonical form followed by grammatical and/or semantic codes. The canonical form is separated from the different codes by a comma. There is an example:

```
horse, N4+An1
```

The first grammatical or semantic code will be interpreted by the inflection program as the name of the grammar used to inflect the entry. The entry of the example above indicates that the word *horse* has to be inflected using the grammar named N4. It is possible to add inflectional codes to the entries, but the nature of the inflection operation limits the usefulness of this possibility. For more details see below in section 3.4.

3.1.3 Dictionary Contents

The dictionaries provided with Unitex contain descriptions of simple and compound words. These descriptions indicate the grammatical category of each entry, optionally their inflectional codes, and various semantic information. The following tables give an overview of some of the different codes used in the Unitex dictionaries. These codes are the same for almost all languages, though some of them are special for certain languages (*i.e.* code for neuter nouns, etc.).

Code	Description	Examples
A	adjective	fabulous, broken-down
ADV	adverb	actually, years ago
CONJC	coordinating conjunction	but
CONJS	subordinating conjunction	because
DET	determiner	each
INTJ	interjection	eureka
N	noun	evidence, group theory
PREP	preposition	without
PRO	pronoun	you
V	verb	overeat, plug-and-play

Table 3.1: Frequent grammatical codes

Code	Description	Example
z1	general language	joke
z2	specialized language	floppy disk
z3	very specialized language	serialization
Abst	abstract	patricide
An1	animal	horse
An1Coll	collective animal	flock
Conc	concrete	chair
ConcColl	collective concrete	rubble
Hum	human	teacher
HumColl	collective human	parliament
t	transitive verb	kill
i	intransitive verb	agree

Table 3.2: Some semantic codes

NOTE: The descriptions of tense in table 3.3 correspond to French. Nonetheless, the majority of these definitions can be found in other languages (infinitive, present, past participle, etc.).

In spite of a common base in the majority of languages, the dictionaries contain encoding

particularities that are specific for each language. Thus, as the declination codes vary a lot between different languages, they are not described here. For a complete description of all codes used within a dictionary, we recommend that you contact the author of the dictionary directly.

Code	Description
m	masculine
f	feminin
n	neuter
s	singular
p	plural
1, 2, 3	1st, 2nd, 3rd person
P	present indicative
I	imperfect indicative
S	present subjunctive
T	imperfect subjunctive
Y	present imperative
C	present conditional
J	simple past indicative
W	infinitive
G	present participle
K	past participle
F	future indicative

Table 3.3: Common inflectional codes

However, these codes are not exclusive. A user can introduce his own codes and create his own dictionaries. For example, for educational purposes one could use a marker "faux-ami" (*false friend*) in a French dictionary:

```
blessier, .V+faux-ami/injure
casque, .N+faux-ami/helmet
journée, .N+faux-ami/day
```

It is equally possible to use dictionaries to add extra information. Thus, you can use the inflected form of an entry to describe an abbreviation and the canonical form to provide the complete form:

```
DNA, DeoxyriboNucleic Acid.ACRONYM
LADL, Laboratoire d'Automatique Documentaire et Linguistique.ACRONYM
UN, United Nations.ACRONYM
```

3.2 Checking dictionary format

When dictionaries become large, it becomes tiresome to check them by hand. Unitex contains the program `CheckDic` that automatically checks the format of DELAF and DELAS dictionaries.

This program verifies the syntax of the entries. For each malformed entry the program outputs the line number, the content of the line and an error message. Results are saved in the file `CHECK_DIC.TXT` which is displayed when the verification is finished. In addition to eventual error messages, the file also contains the list of all characters used in the inflectional and canonical forms, the list of grammatical and semantic codes, and the list of inflectional codes that appear in the dictionary. The character list makes it possible to verify that the characters used in the dictionary are consistent with those in the alphabet file of the language. Each character is followed by its value in hexadecimal notation.

The code lists can be used to check that there are no typing errors in the codes of the dictionary.

The `CheckDic` program works with non-compressed dictionaries, *i.e.* the files in text format. The general convention is to use the `.dic` extension for these dictionaries. In order to check the format of a dictionary, you first open it by choosing "Open..." in the "DELA" menu.

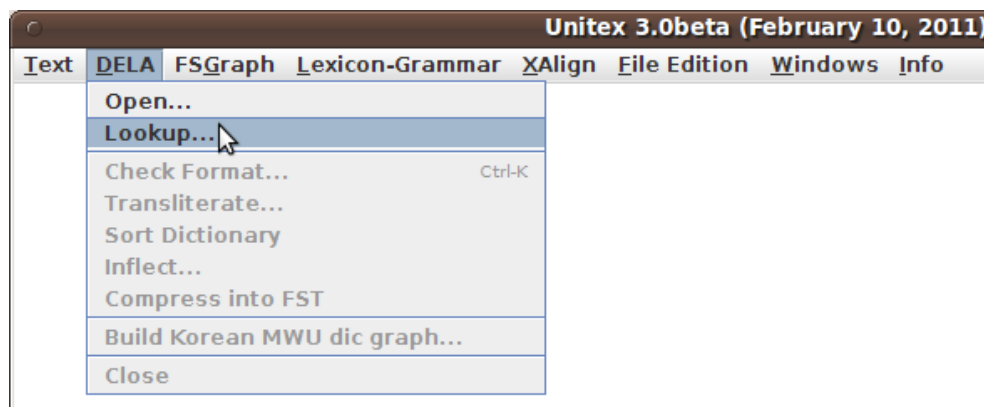


Figure 3.1: "DELA" Menu

Let's load the dictionary as in figure 3.2. Then, click on "Check Format..." in the "DELA" menu. A window like in figure 3.3 is opened. You must select the type of dictionary you want to check. After checking the dictionary in Figure 3.2, results are presented as shown in Figure 3.4.

The first error is caused by a missing period. The second, by the fact that no comma was found after the end of an inflected form. The third error indicates that the program didn't find any grammatical or semantic codes.

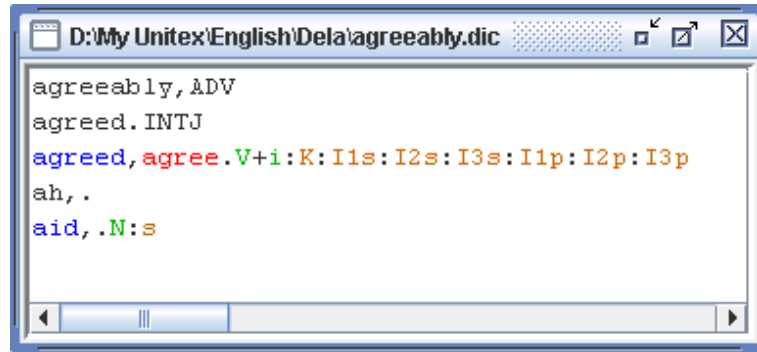


Figure 3.2: Dictionary example

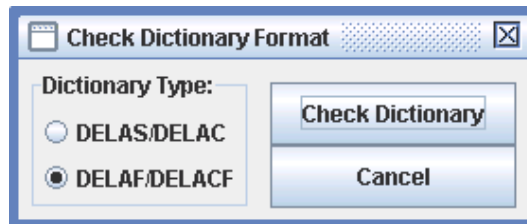


Figure 3.3: Checking a dictionary

3.3 Sorting

Unitex uses the dictionaries without having to worry about the order of the entries. When displaying them it is sometimes preferable to sort the dictionaries. The sorting depends on a number of criteria, first of all on the language of the text. Therefore the sorting of a Thai dictionary is done according to an order different from the alphabetical order. So different in fact that Unitex uses a sorting procedure developed specifically for Thai (see chapter 12).

For European languages the sorting is usually done according to the lexicographical order, although there are some variants. Certain languages like French treat some characters as equivalent. For example, the difference between the characters *e* and *é* is ignored if one wants to compare the words *manger* et *mangés* because the contexts *r* and *s* allow to decide the order. The difference is only taken into account when the contexts are identical, as they are when comparing *pêche* and *pèche*.

To allow for such effect, the `SortTxt` program uses a file which defines the equivalence of characters. This file is named `Alphabet_sort.txt` and can be found in the user directory for the current language. By default the first lines of this file for French look like this:

```

AÀÄäaàää
Bb
CÇçç
  
```

```

Check Results
Line 1: unexpected end of line
agreeably,ADV
Line 2: unexpected end of line
agreed.INTJ
Line 4: empty grammatical or semantic code
ah,.

-----
-----  Stats  -----
-----

File: D:\My Unitex\English\Dela\agreeably.dic
Type: DELAF
5 lines read
2 simple entries for 2 distinct lemmas
0 compound entry for 0 distinct lemma

-----
----  All chars used in forms  ----
-----

a (0061)
d (0064)
e (0065)
g (0067)
i (0069)
r (0072)

-----
----  3 grammatical/semantic codes used in dictionary  ----
-----

V
i
N

-----
----  8 inflectional codes used in dictionary  ----
-----

K
I1s
I2s
I3s
I1p
I2p
I3p
s

```

Figure 3.4: Results of checking

```
Dd
EÉÊËÊëéèêë
```

Characters in the same line are considered equivalent if the context permits. If two equivalent characters must be compared, they are sorted in the order they appear in from left to right. As can be seen from the extract above, there is no difference between lower and upper case. Accents and the cedille character are ignored as well.

To sort a dictionary, open it and then click on "Sort Dictionary" in the "DELA" menu. By default, the program always looks for the file `Alphabet_sort.txt`. If that file doesn't exist, the sorting is done according to the character indices in the Unicode encoding. By modifying that file, you can define your own sorting order.

NOTE: After applying the dictionaries to a text, the files `d1f`, `d1c` and `err` are automatically sorted using this program.

3.4 Automatic inflection

3.4.1 Inflection of simple words

As described in section 3.1.2, a line in a DELAS consists of a canonical form and a sequence of grammatical or semantic codes:

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

The first code is used to determine the grammatical code of the entry as well as the name of the grammar used to inflect the canonical form. There are two possible forms:

- `V32`: grammar name=`V32.fst2`, grammatical code=`V` (longest letter prefix)
- `N(NC_XXX)`: grammar name=`NC_XXX.fst2`, grammatical code=`N`

These inflectional grammars will automatically be compiled if needed. In the example above, all entries will be inflected by a grammar named `N4`.

In order to inflect a dictionary, click on "Inflect..." in the "DELA" menu. The window in figure 3.5 allows you to specify the directory in which inflectional grammars are found. By default, the subdirectory `Inflection` of the directory for the current language is used. You can also specify the kind of words your DELAS is supposed to contain. If an entry is found that does not correspond to your choice, an error message will be displayed.

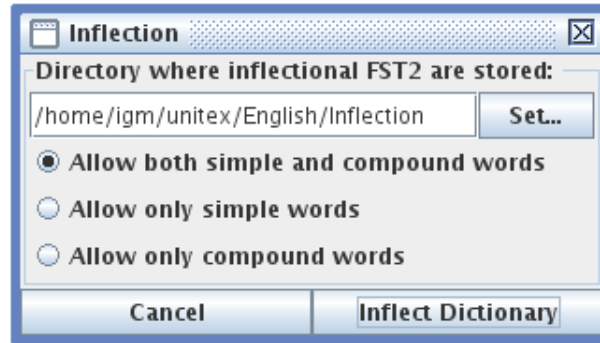


Figure 3.5: Configuration of automatic inflection

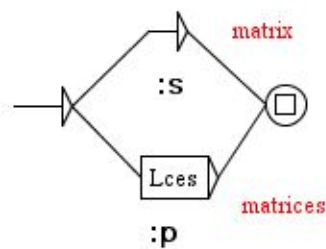


Figure 3.6: Inflectional grammar N4

Figure 3.6 shows an example of an inflectional grammar. The paths describe the suffixes to add or to remove to get to an inflected form from a canonical form, and the outputs (text in bold under the boxes) are the inflectional codes to add to a dictionary entry.

In our example, two paths are possible. The first does not modify the canonical form and adds the inflectional code **:s**. The second deletes a letter with the **L** operator, then adds the **ux** suffix and adds the inflectional code **:mp**. Here are the operators that can be used:

- **L** (left) removes a letter from the entry
- **R** (right) restores a letter to the entry. In French, many verbs of the first group are conjugated in the present singular of the third person form by removing the **r** of the infinitive and changing the 4th letter from the end to è: *peler* → *pèle*, *acheter* → *achète*, *gérer* → *gère*, etc. Instead of describing an inflectional suffix for each verb (**LLLLèle**, **LLLLète** and **LLLLère**), the **R** operator can be used to describe it in one way: **LLLLèRR**.
- **C** (copy) duplicates a letter in the entry and moves everything on its right by one position. In cases like *permitted* or *hopped*, we see a duplication of the final consonant of the verb. To avoid writing an inflectional graph for every possible final consonant, one can use the **C** operator to duplicate any final consonant.

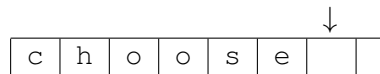
- **D** (delete) deletes a letter, shifting anything located on the right of this letter. For instance, if you want to inflect the Romanian word `europeean` into `europeeni`, you must use the sequence `LDRi`. `L` will move the cursor on the `a`, `D` will delete the `a`, shifting the `n` on the left, and then `Ri` will restore the `n` and add an `i`.
- **U** (unaccent) removes the accent of the current character, if any. For instance the sequence `LLUx` applied to the word `manges` produces the inflected form `mangex`, since `U` has turn the `e` into a `e`.
- **P** (uppercase) uppercases the initial letter of the stack. For instance, the sequence `Px` will turn `foo` into `Foox`.
- **W** (lowercase) lowercases the initial letter of the stack.
- `<R=?>` replaces the initial letter of the stack by the letter `?`.
- `<I=?>` inserts the letter `?` before the initial letter of the stack.
- `<X=n>` removes the first `n` letters of the stack.

There are also two operators dedicated to Korean:

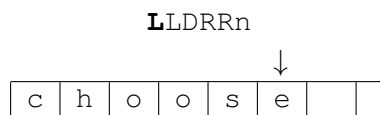
- **J** removes a Jamo letter. If the current character is a Hangul syllab character, this character is first replaced by the equivalent Jamo sequence, and then, the last Jamo letter is removed. If the current character is neither a Jamo nor a Hangul, and error is raised.
- **.** (latin dot) inserts a syllab bound. As a side effect, if the top of the stack contains Jamo letters, they are first recombined into a Hangul character.

In the example below, the inflection of `choose` is shown. The sequence `LLDRRn` describes the form `chosen`:

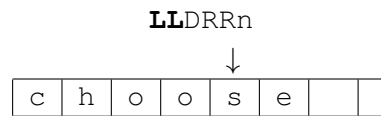
- **Step 0:** the canonical form is copied on the stack, and the cursor is set behind the last letter:



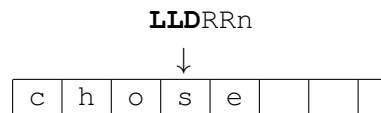
- **Step 1:** the cursor is moved one position to the left:



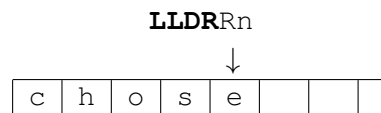
- Step 2: the cursor is moved one position to the left again:



- Step 3: one character is deleted; everything to the right of the cursor is shifted one position to the left:



- Step 4: the cursor is moved to the right:



- Step 5: and to the right again:



- Step 6: the character n is pushed on the stack:



When all operations have been fulfilled, the inflected form consists of all letters before the cursor (here *chosen*).

The inflection program explores all paths of the inflectional grammar and tries all possible forms. In order to avoid having to replace the names of inflectional grammars by the real grammatical codes in the dictionary used, the program replaces these names by the longest prefixes made of letters if you have selected the "Remove class numbers" button. Thus, N4

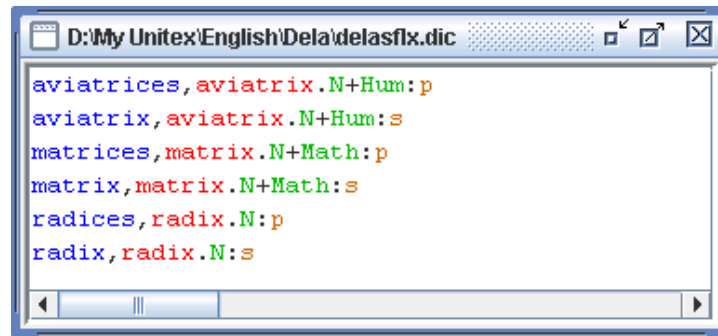


Figure 3.7: Result of automatic inflection

is replaced by N. By choosing the inflectional grammar names carefully, one can construct a ready-to-use dictionary.

Figure 3.7 shows the dictionary we get after the inflection of our DELAS example.

Semantic codes

In some languages, there are inflectional features that actually correspond to semantic ones, like for instance markers for the passive form. Such codes may not appear as inflectional ones, but rather as semantic ones. To do that and produce semantic codes, you have to insert a plus sign at the beginning of the output of a box. The box must only contain the semantic code preceded by a plus, as shown on Figure 3.8.

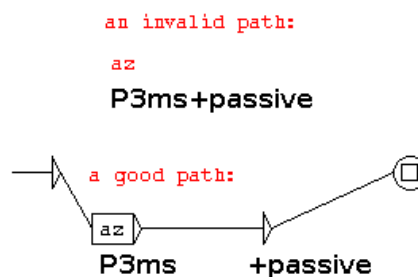


Figure 3.8: An inflection grammar with a semantic code

3.4.2 Inflection of compound words

See chapter 10.

3.4.3 Inflection of semitic languages

Semitic languages like Arabic or Hebrew are not inflected in the same way than other kinds of languages, since their morphology obey a different logic. In fact, in such languages, words are inflected according to *consonant skeletons*. A lemma is made of consonants, and the inflection process is supposed to enrich this skeleton with vowels.

First, let us see what a semitic entry is supposed to be:

ktb, \$V31-123

The \$ sign before the grammatical code indicates that this is a semitic entry, and the lemma (here `ktb`) is the consonant skeleton. Figure 3.9 shows the toy grammar `V31-123.grf` that illustrates how the semitic inflection process works.

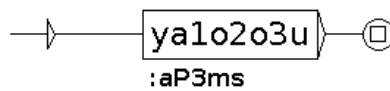


Figure 3.9: A toy semitic inflection grammar

Such a grammar obey the following rules:

1. All standard inflection operators can be used (L, R, etc).
2. A digit stands for a consonant of the skeleton (1 for the first, 2 for the second, etc). In our example, 1, 2 and 3 will respectively stand for k, t and b. If you want to access to a letter after the ninth one, you must protect its index with angles like `<10>`.

The DELAF output for this grammar is:

yakotobu, ktb.V:aP3ms

3.5 Compression

Unitex applies compressed dictionaries to the text. The compression reduces the size of the dictionaries and speeds up the lookup. This operation is done by the `Compress` program. This program takes a dictionary in text form as input (for example `my_dico.dic`) and produces two files:

- `my_dico.bin` contains the minimal automaton of the inflected forms of the dictionaries;
- `my_dico.inf` contains the codes extracted from the original dictionary.

The minimal automaton in the `my_dico.bin` file is a representation of inflected forms in which all common prefixes and suffixes are factorized. For example, the minimal automaton of the words `me`, `te`, `se`, `ma`, `ta` et `sa` can be represented by the graph shown in Figure 3.10.

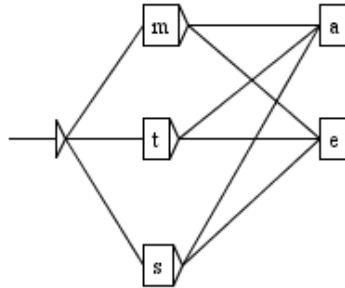


Figure 3.10: Representation of a minimal automaton

To compress a dictionary, open it and click on "Compress into FST" in the "DELA" menu. The compression is independent from the language and from the content of the dictionary. The messages produced by the program are displayed in a window that is not closed automatically. You can see the size of the resulting `.bin` file, the number of lines read and the number of inflectional codes created. Figure 3.11 shows the result of the compression of a dictionary of simple words.

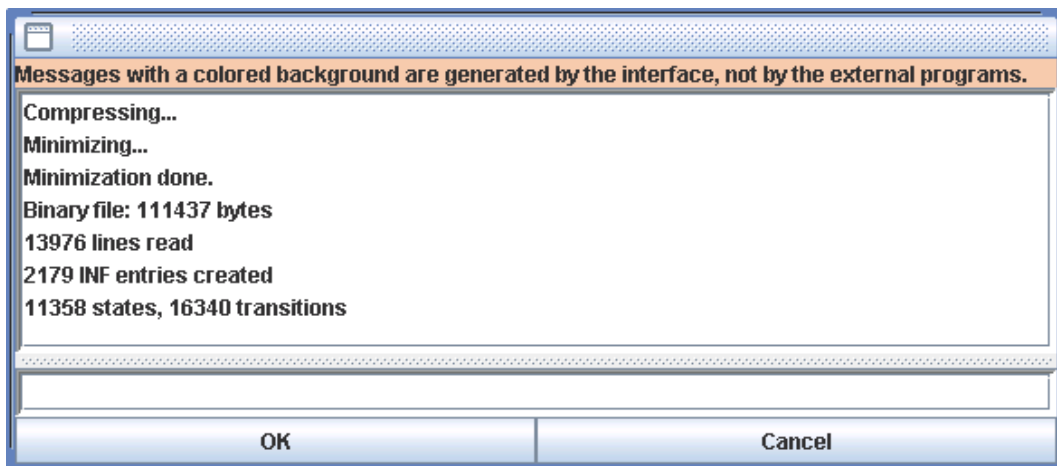


Figure 3.11: Results of a compression

The resulting files are compressed to about 95% for dictionaries containing simple words and 50% for those with compound words.

NOTE: for semitic languages, a special compression algorithm is used to reduce the size of the output `.bin` and `.inf` files. The fact that a language is considered as a semitic one can be configured in the global preferences.

3.6 Applying dictionaries

Dictionaries can be applied (1) after pre-processing or (2) by explicitly clicking on "Apply Lexical Resources" in the "Text" menu (see section 2.5.5).

Unitex can manipulate compressed dictionaries (`.bin`) and dictionary graphs (`.fst2`). We will now describe the rules for applying dictionaries in detail. Dictionary graphs will be described in section 3.6.3.

3.6.1 Priorities

The priority rule says that if a word in a text is found in a dictionary, this word will not be taken into account by dictionaries with lower priority.

This allows for eliminating a part of ambiguity when applying dictionaries. For example, the French word *par* has a nominal interpretation in the golf domain. If you don't want to use this meaning, it is sufficient to create a filter dictionary containing only the entry `par, .PREP` and to apply this with highest priority. This way, even if simple word dictionaries contain different entries, they will be ignored given the priority rule.

There are three priority levels. The dictionaries whose names without extension end with `-` have the highest priority; those that end with `+` have the lowest one. All other dictionaries are applied with medium priority. The order in which dictionaries with the same priority are applied does not matter. On the command line, the command:

```
Dico ex.snt alph.txt ctr+.bin cities-.bin rivers.bin regions-.bin
```

will apply the dictionaries in the following order (`ex.snt` is the text to which the dictionaries are applied, and `alph.txt` is the alphabet file used):

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

3.6.2 Application rules for dictionaries

Besides the priority rule, the application of dictionaries respects upper case letters and spaces. The upper case rule is as follows:

- if there is an upper case letter in the dictionary, then an upper case letter has to be in the text;
- if a lower case letter is in the dictionary, there can be either an upper or lower case letter in the text.

Thus, the entry `peter, .N:fs` will match the words `peter`, `Peter` et `PETER`, while `Peter, .N+firstName` only recognizes `Peter` and `PETER`. Lower and upper case letters are defined in the alphabet file passed to the `Dico` program as a parameter.

Respecting white space is a very simple rule: For each sequence in the text to be recognized by a dictionary entry, it has to have exactly the same number of spaces. For example, if the dictionary contains `aujourd'hui, .ADV`, the sequence `Aujourd' hui` will not be recognized because of the space that follows the apostrophe.

3.6.3 Dictionary graphs

The `Dico` program can also apply dictionary graphs. Dictionary graphs conform to the following rule: if applied by `Locate` in MERGE mode, they must produce output sequences that are valid DELAF lines.

Figure 3.12 shows a graph that recognizes chemical elements. We can observe a first advantage of graphs over usual dictionaries: we can force case with double quotes. Thus, this graph will correctly match `Fe` but not `FE`, while this restriction cannot be specified in a normal DELAF.

Another advantage of dictionary graphs is that they can use results given by previous dictionaries. Thus, it is possible to apply the standard dictionary, and then tag as proper names all the unknown words that begin with an uppercase letter, thanks to the graph `NP r+` shown in figure 3.13. The `+` in the graph name gives to it a low priority, so that it will be applied after the standard dictionary. This graph works with words that are still unknown after the application of the standard dictionary. Square brackets stand for a context definition. For more information about contexts, see section 6.3.

Since dictionary graphs are applied using the engine of `Locate`, they have exactly the same properties than syntactic graphs. So, you can use morphological filters and/or morphological mode. For instance, the graph shown on Figure 3.14 use morphological filters to recognize roman numerals. Note that it also uses contexts in order to avoid recognizing uppercase letters in some contexts.

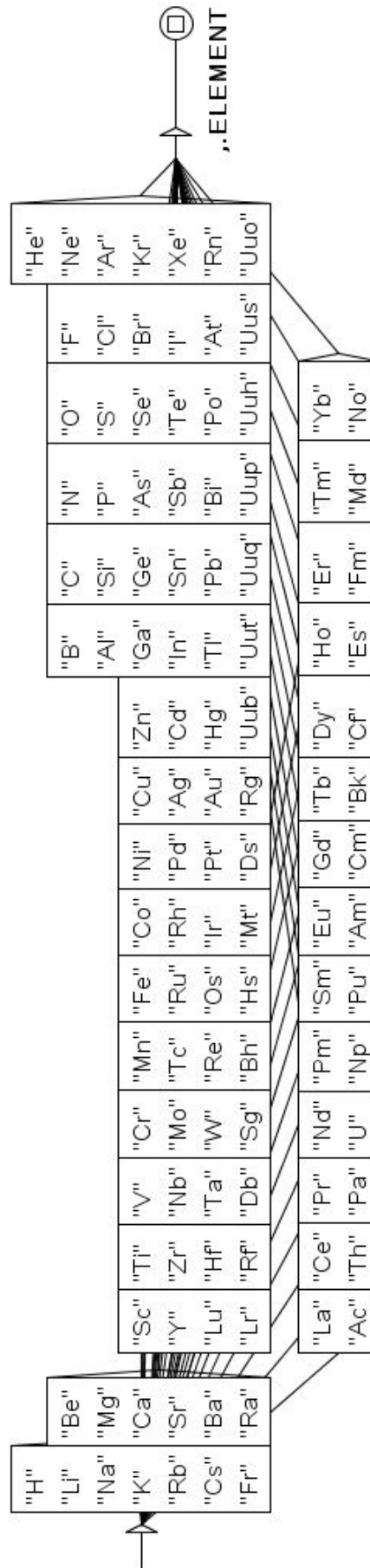


Figure 3.12: Dictionary graph of chemical elements

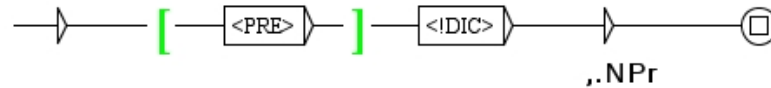


Figure 3.13: Dictionary graph that tags unknown words beginning with an uppercase letter as proper names

By default, dictionary graphs are applied in MERGE mode. If you want to apply them in REPLACE mode, you must suffix graph names with `-r`. This can be combined with the `+` and `-` priority marks:

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

Exporting produced entries as a morphological dictionary

Dictionary entries produced by dictionary graphs are, of course, taken into account by the `Locate` program. However, you can not refer to them in the morphological mode since they do not belong to a morphological dictionary. If you want to do so, you just have to add `b` at the end of the graph name. If you add `z` instead, then the morphological dictionary will be compressed immediately, thus being usable by the next dictionary graph to be applied.

Naming conventions

The whole naming scheme for dictionary graph is as follows:

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

where:

- X is in `[rRmM]`: `r` means REPLACE mode; `M` means MERGE mode (default);
- Y is in `[bBzZ]`: option that rules the production of a morphological dictionary (see previous section);
- Z is in `[aAllsS]`: `a` means that the graph will be applied in "All matches" mode; `l` means "Longest matches" mode (default); `s` means "Shortest matches" mode.

3.6.4 Morphological dictionary graphs

In addition to dictionary graphs that produce new entries in the text dictionaries, you can design morphological dictionary graphs. The output of such graphs will be used as special input for the construction of the text automaton. We call them "morphological dictionary graphs", because their main utility is to introduce new morphological analysis in the text automaton, using the morphological mode (see section 6.4). This functionality will be helpful for agglutinative languages like Korean.

The rule is simple: any output of a dictionary graph that begins with a slash will be added to the file `tags.ind`, located in the text directory. This file is used by the `Txt2Fst2` program

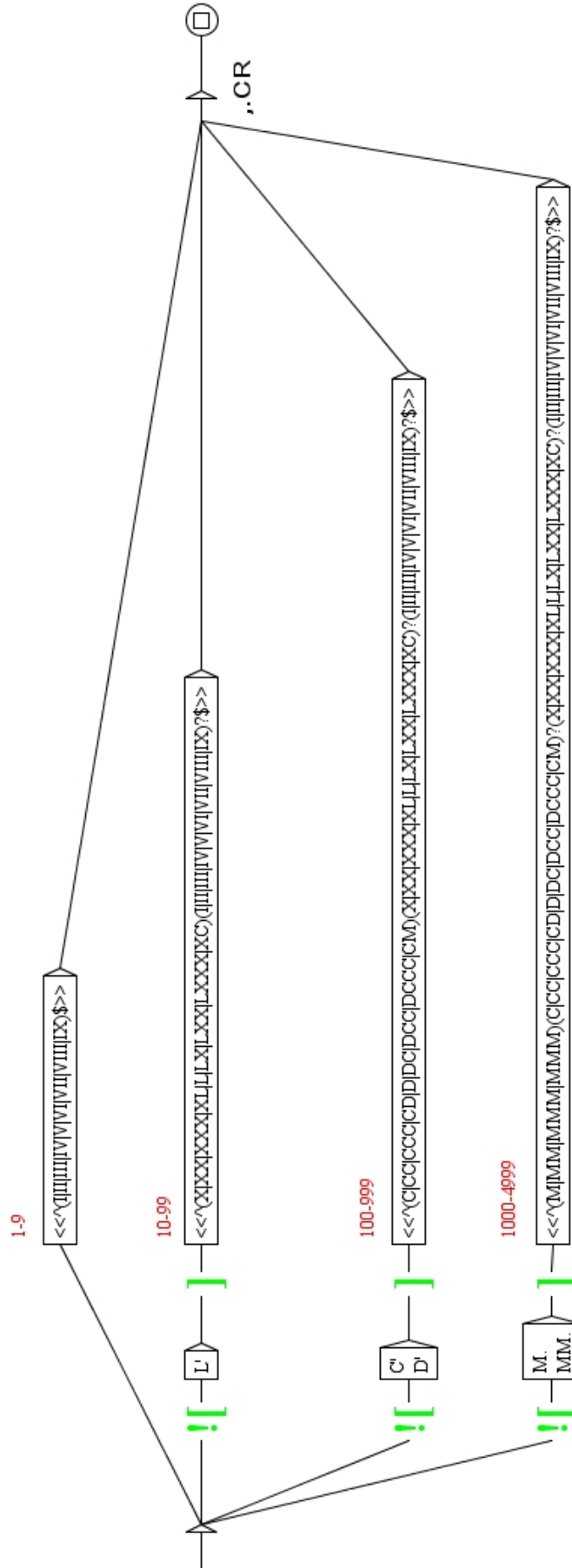


Figure 3.14: Dictionary graph of roman numerals

in order to add interpretations into the text automaton. Let us consider the grammar shown on Figure 3.15 that matches words made of the prefix `un` followed by an adjective. If we apply this grammar as a dictionary graph, we obtain new paths in the text automaton, as shown on Figure 3.16. Note that when two tags correspond to analysis within the same token, the link between them is displayed with a dashed line.

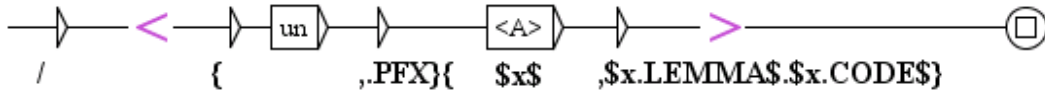


Figure 3.15: Example of morphological dictionary graph

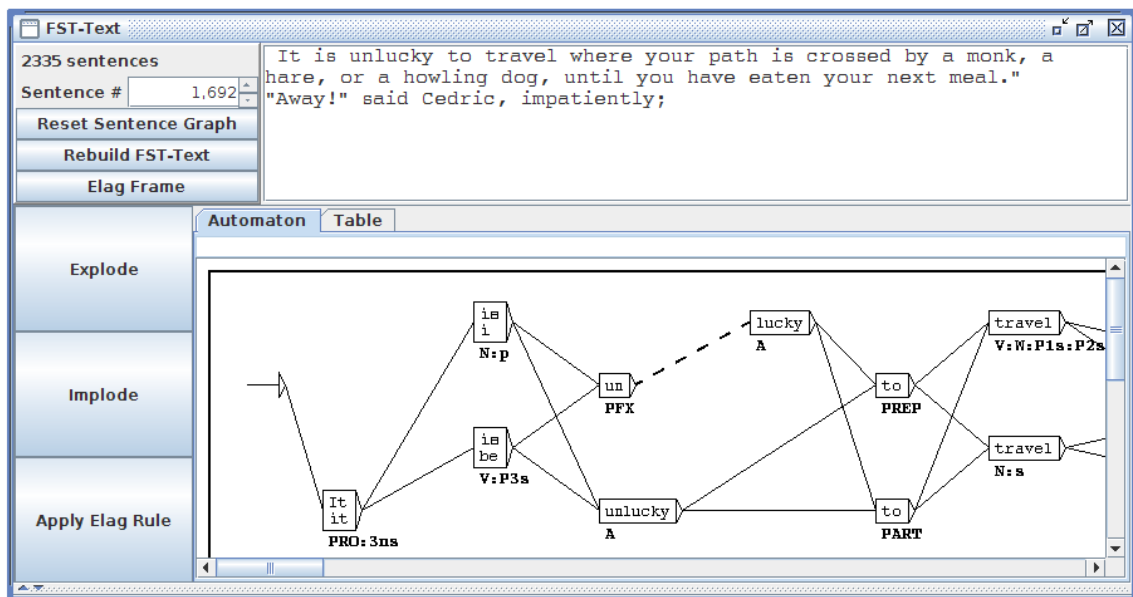


Figure 3.16: Path added by a morphological dictionary graph

3.7 Bibliography

Table 3.4 gives some references for electronic dictionaries with simple and compound words. For more details, see the references page on the Unitex website:

<http://www-igm.univ-mlv.fr/~unitex>

Language	Simple words	Compound words
English	[56], [70]	[15], [84]
French	[19], [20], [61]	[20], [37], [86], [45]
Modern Greek	[2], [17], [58]	[59], [60]
Italian	[28], [29]	[90]
Spanish	[8]	[7]
Portuguese	[25], [82], [79], [71]	[78], [79]

Table 3.4: Some bibliographical references for electronic dictionaries

Chapter 4

Searching with regular expressions

This chapter describes how to search a text for simple patterns by using regular expressions.

4.1 Definition

The goal of this chapter is not to give an introduction on formal languages but to show how to use regular expressions in Unitex in order to search for simple patterns. Readers who are interested in a more formal presentation can consult the many works that discuss regular expression patterns.

A regular expression can be:

- a token (`book`) or a lexical mask (`<smoke.V>`);
- the concatenation of two regular expressions (`he smokes`);
- the union of two regular expressions (`Pierre+Paul`);
- the Kleene star of a regular expression (`bye*`).

4.2 Tokens

In a regular expression, a token is defined as in [2.5.4](#) (page [41](#)). Note that the symbols dot, plus, star, less than, opening and closing parentheses and double quotes have a special meaning. It is therefore necessary to precede them with an escape character `\` if you want to search for them. Here are some examples of valid tokens:

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

By default, Unix is set up to let lower case patterns also find upper-case matches. It is possible to enforce case-sensitive matching using quotation marks. Thus, "peter" recognizes only the form peter and not Peter or PETER.

NOTE: in order to make a space obligatory, it needs to be enclosed in quotation marks.

4.3 Lexical masks

A lexical mask is a search query that matches tokens or sequences of tokens.

4.3.1 Special symbols

There are two kinds of lexical masks. The first category contains all symbols that have been introduced in section 2.5.2 except for the symbol <PNC>, which matches punctuation signs, and <^>, which matches a line feed. Since all line feeds have been replaced by spaces this symbol cannot longer be useful when searching for lexical masks. These symbols, also called *meta-symbols*, are the following:

- <E> : the empty word or epsilon. Matches the empty string;
- <TOKEN> : matches any token, except the space; used by default for morphological filters
- <MOT> : matches any token that consists of letters;
- <MIN> : matches any lower-case token;
- <MAJ> : matches any upper-case token;
- <PRE> : matches any token that consists of letters and starts with a capital letter;
- <DIC> : matches any word that is present in the dictionaries of the text;
- <SDIC> : matches any simple word in the text dictionaries;
- <CDIC> : matches any composed word in the dictionaries of the text;
- <TDIC> : matches any tagged token like {XXX, XXX.XXX};
- <NB> : matches any contiguous sequence of digit (1234 is matched but not 1 234);
- # : prohibits the presence of space.

NOTE: as described in section 2.5.4, NO meta can be used to match the {STOP} marker, not even <TOKEN>.

4.3.2 References to information in the dictionaries

The second kind of lexical masks refers to the information in the text dictionaries. The four possible forms are:

- `<be>`: matches all the entries that have `be` as canonical form. Note that this pattern is ambiguous if `be` is also a grammatical or semantic code;
- `<be.>`: matches all the entries that have `be` as canonical form. This pattern is not ambiguous as the previous one;
- `<be.V>`: matches all entries having `be` as canonical form and the grammatical code `V`;
- `<V>`: matches all entries having the grammatical code `V`. This pattern is as ambiguous as the first one. To remove the ambiguity, you can use either `<.V>` or `<+V>`;
- `{am,be.V}` or `<am,be.V>`: matches all the entries having `am` as inflected form, `be` as canonical form and the grammatical code `V`. This kind of lexical mask is only of interest if applied to the text automaton where all the ambiguity of the words is explicit. While executing a search on the text, that lexical mask matches the same as the simple token `am`.

4.3.3 Grammatical and semantic constraints

The references to dictionary information (`be`, `V`) in these examples are basic. It is possible to express more complex lexical masks by using several grammatical or semantic codes separated by the character `+`. An entry of the dictionary is then only found if it has all the codes that are present in the mask. The mask `<N+z1>` thus recognizes the entries:

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

but not:

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

It is possible to exclude codes by preceding them with the character `~` instead of `+`. In order to be recognized, an entry has to contain all the codes required by the lexical mask and none of the prohibited ones. The mask `<A~z3>` thus recognizes all the adjectives that do not have the code `z3` (cf. table 3.2). If you want to refer to a code containing the character `~` you have to escape this character by preceding it with a `\`.

CHANGE NOTE: before version 2.1, the negation operator was the minus. If you want to preserve backward compatibility without modifying your graphs, you have to call `Locate` by hand with the `-g minus` option.

The order in which the codes appear in the mask is not important. The three following patterns are equivalent:

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

NOTE: it is not possible to use a lexical mask that only has prohibited codes. <~N> and <~A~z1> are thus incorrect masks. However, you can express such constraints using contexts (see section 6.3).

4.3.4 Inflectional constraints

It is also possible to specify constraints about the inflectional codes. These constraints have to be preceded by at least one grammatical or semantic code. They are represented as inflectional codes present in the dictionaries. Here are some examples of lexical masks using inflectional constraints:

- <A:m> recognizes a masculine adjective;
- <A:mp:f> recognizes a masculine plural or a feminine adjective;
- <V:2:3> recognizes a verb in the 2nd or 3rd person; that excludes all tenses that have neither a 2nd or 3rd person (infinitive, past participle and present participle) as well as the tenses that are conjugated in the first person.

In order to let a dictionary entry E be recognized by mask M , it is necessary that at least one inflectional code of E contains all the characters of an inflectional code of M . Consider the following example:

$E = \text{pretext}, .V:W:P1s:P2s:P1p:P2p:P3p$
 $M = \langle V:P3s:P3 \rangle$

No inflectional code of E contains the characters P , 3 and s at the same time. However, the code $P3p$ of E does contain both characters P and 3 . The code $P3$ is included in at least one code of E , mask M thus recognizes entry E . The order of the characters inside an inflectional code is without importance.

4.3.5 Negation of a lexical mask

It is possible to negate a lexical mask by placing the character ! immediately after the character <. Negation is possible with the masks <MOT>, <MIN>, <MAJ>, <PRE>, <DIC> as well as with the masks that carry grammatical, semantic or inflectional codes (i.e. <!V~z3:P3>). The masks # and " " are the negation of each other. The mask <!MOT> recognizes all tokens that do not consist of letters except for the sentence separator {S} and the {STOP} marker. Negation has no effect on <NB>, <SDIC>, <CDIC>, <TDIC> and <TOKEN>.

The negation is interpreted in a special way in the lexical masks <!DIC>, <!MIN>, <!MAJ> and <!PRE>. Instead of recognizing all forms that are not recognized by the mask without negation, these masks find only forms that are sequences of letters. Thus, the mask <!DIC> allows you to find all unknown words in a text. These unknown forms are mostly proper names, neologisms and spelling errors.

The negation of a dictionary mask like <V:G> will match any word, except for those that are matched by this mask. For instance, <!V:G> will not match the word *being*, even if there are homonymic non-verbal entries in the dictionaries:

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

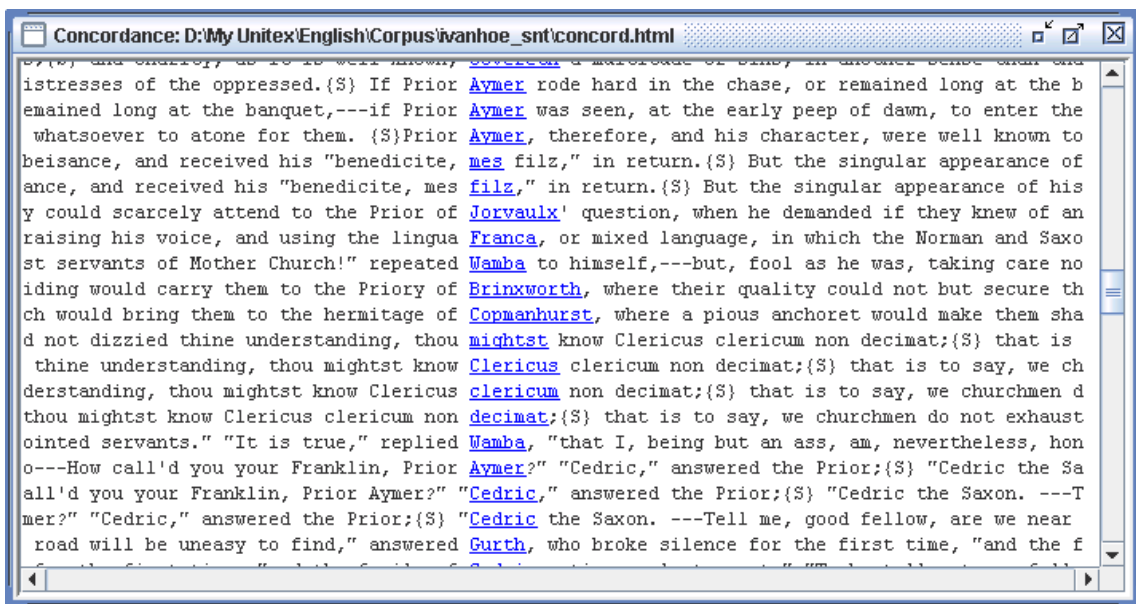


Figure 4.1: Result of the search for <!DIC>

Here are some examples of lexical masks with the different types of constraints:

- <A~Hum:fs> : a non-human adjective in the feminine singular;
- <lire.V:P:F> : the verb *lire* in the present or future tense;
- <suis,suivre.V> : the word *suis* as inflected form of the verb *suivre* (as opposed to the form of the verb *être*);
- <facteur.N~Hum> : all nominal entries that have *facteur* as canonical form and that do not have the semantic code Hum;

- `<!ADV>` : all words that are not adverbs;
- `<!MOT>` : all tokens that are not made of letters (cf. figure 4.2). This mask does not recognize the sentence separator `{S}` and the special tag `{STOP}`.

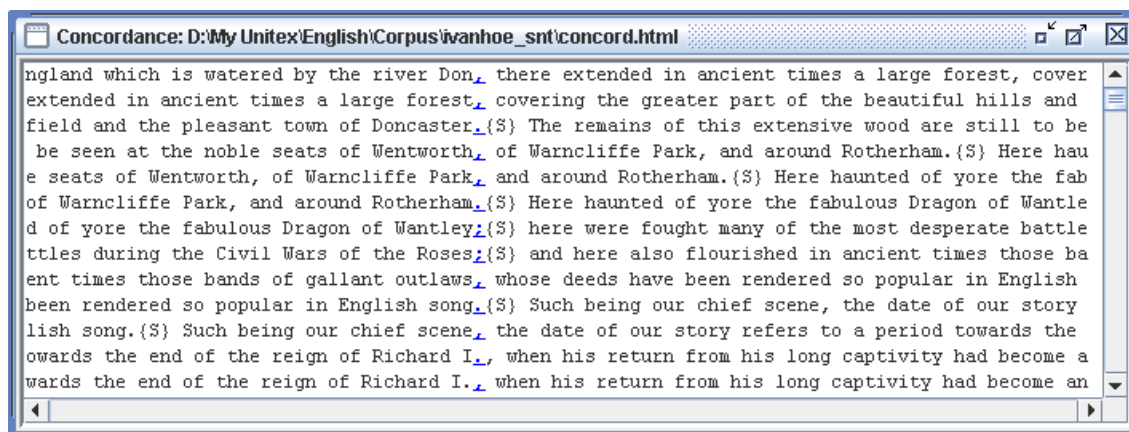


Figure 4.2: Result of a search for the pattern `<!MOT>`

4.4 Concatenation

There are three ways to concatenate regular expressions. The first consists in using the concatenation operator which is represented by the dot. Thus, the expression:

```
<DET>.<N>
```

recognizes a determiner followed by a noun. The space can also be used for concatenation, as well as the empty string. The following expressions:

```
the <A> cat
the<A>cat
```

recognizes the token *the*, followed by an adjective and the token *cat*. The parenthesis are used as delimiters of a regular expression. All of the following expressions are equivalent:

```
the <A> cat
(the <A>)cat
the.<A>cat
(the).<A> cat
(the.<A>)) (cat)
```

4.5 Union

The union of regular expressions is expressed by typing the character + between them. The expression

```
(I+you+he+she+it+we+they) <V>
```

recognizes a pronoun followed by a verb. If an element in an expression is optional, it is sufficient to use the union of this element and the empty word epsilon. Examples:

```
the (little+<E>) cat recognizes the sequences the cat and the little cat
```

```
(<E>+Anglo-).(French+Indian) recognizes French, Indian, Anglo-French and Anglo-Indian
```

4.6 Kleene star

The Kleene star, represented by the character *, allows you to recognize zero, one or several occurrences of an expression. The star must be placed on the right hand side of the element in question. The expression:

```
this is very* cold
```

recognizes *this is cold, this is very cold, this is very very cold*, etc. The star has a higher priority than the other operators. You have to use brackets in order to apply the star to a complex expression. The expression:

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

recognizes a zero followed by a comma and by a possibly empty sequence of digits.

WARNING: It is prohibited to search for the empty word with a regular expression. If you try to search for `(0+1+2+3+4+5+6+7+8+9) *`, the program will raise an error as shown in figure 4.3.

4.7 Morphological filters

It is possible to apply morphological filters to the lexemes found. For that, it is necessary to immediately follow the lexeme found by a filter in double angle brackets:

```
lexical mask<<morphological pattern>>
```

The morphological filters are expressed as regular expressions in POSIX format (see [63] for the detailed syntax). Here are some examples of elementary filters:

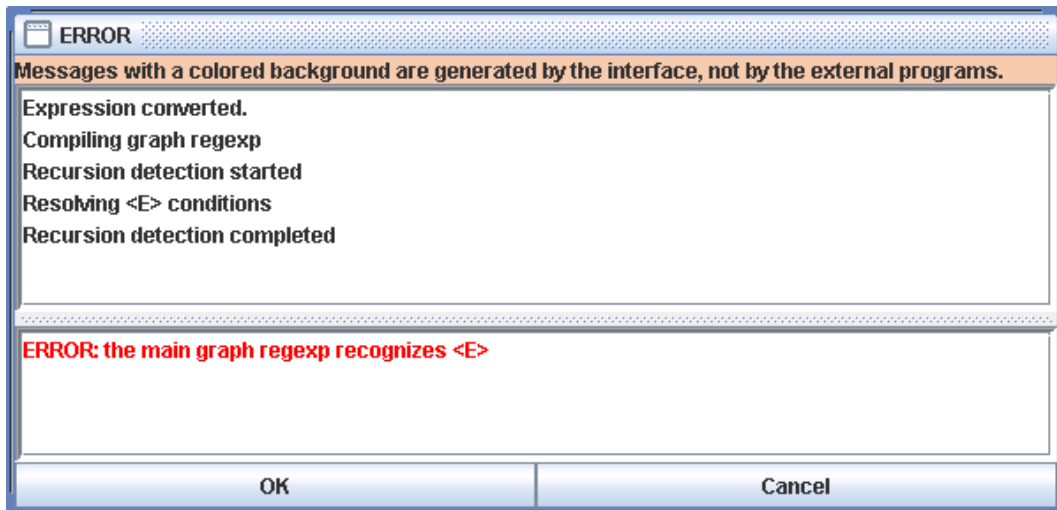


Figure 4.3: Error message when searching for the empty string

- <<ss>>: contains *ss*
- <<^a>>: begins with *a*
- <<ez\$>>: ends with *ez*
- <<a.s>>: contains *a* followed by any character, followed by *s*
- <<a.*s>>: contains *a* followed by a sequence of any character, followed by *s*
- <<ss|tt>>: contains *ss* or *tt*
- <<[aeiouy]>>: contains a non accentuated vowel
- <<[aeiouy]{3,5}>>: contains a sequence of non-accentuated vowels whose length is between 3 and 5
- <<es?>>: contains *e* followed by an optional *s*
- <<ss[^e]?>>: contains *ss* followed by an optional character which is not *e*

It is possible to combine these elementary filters to form more complex filters:

- <<[ai]ble\$>>: ends with *able* or *ible*
- <<^(anti|pro)-?>>: begins with *anti* or *pro*, followed by an optional dash
- <<^([rst][aeiouy]){2,}\$>>: a word formed by 2 or more sequences beginning with *r*, *s* or *t* followed by a non-accentuated vowel

- `<<^[^l]|l[^e]>>`: does not begin with `l` unless the second letter is an `e`, in other words, any word except the ones starting with `le`. Such constraints are better described using contexts (see section 6.3).

By default, a morphological filter alone is regarded as applying it to the lexical mask `<TOKEN>`, that means any token except space and `{STOP}`. On the other hand, when a filter follows a lexical mask immediately, it applies to what was recognized by the lexical mask. Here are some examples of such combinations:

- `<V:K><<i$>>`: Past participle ending with `i`
- `<CDIC><<->>`: A compound word containing a dash
- `<CDIC><< . * >>`: a compound word containing at least two spaces
- `<A:fs><<^pro>>`: a feminine singular adjective beginning with `pro`
- `<DET><<^[^u]|(u[^n])|(un.+)>>`: a (French) determiner different from `un`
- `<!DIC><<es$>>`: a word which is not in the dictionary and which ends with `es`
- `<V:S:T><<uiss>>`: a verb in the past or present subjunctive, and containing `uiss`

NOTE: By default, morphological filters are subject to the same variations of case as lexical masks. Thus, the filter `<<^b>>` will recognize all the words starting with `b`, but also those which start with `B`. To force the matcher to respect case, add `_f_` immediately after the filter, e.g.: `<<^b>>_f_`.

4.8 Search

4.8.1 Search configuration

In order to search for an expression, first open a text (cf. chapter 2). Then click on "Locate Pattern..." in the "Text" menu. The window of figure 4.4 appears.

The "Locate pattern in the form of" box allows you to select regular expression or grammar. Click on "Regular expression".

The "Index" box allows you to select the recognition mode:

- "Shortest matches" : prefers shortest matches in case of nested sequences. For instance, if your grammar can recognize the sequences *a very hot chili* and *very hot*, the first one will be discarded;
- "Longest matches" : prefers longest matches (*a very hot chili* in our example). This is the default;
- "All matches" : outputs all recognized sequences.

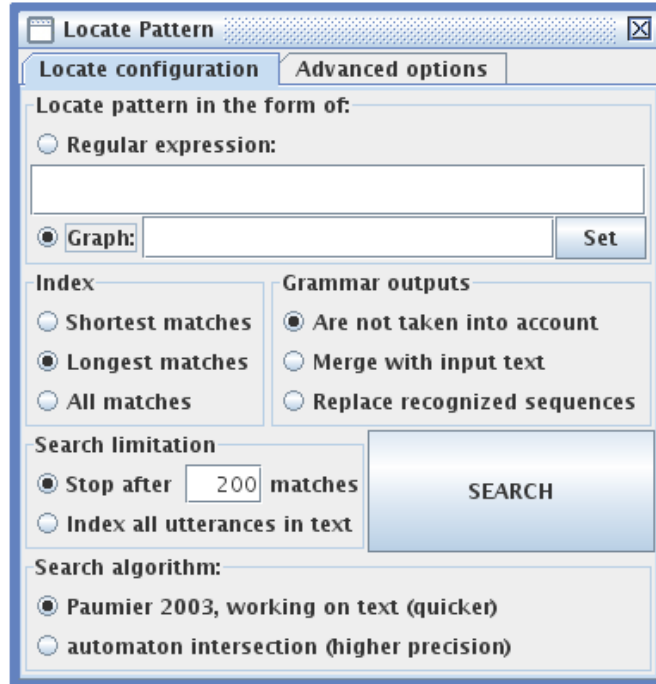


Figure 4.4: “Locate pattern” window

The “Search limitation” box is used to limit the number of results to a certain number of occurrences. By default, the search is limited to the first 200 occurrences.

The options of the “Grammar outputs” box do not concern regular expressions. They are described in section 6.10. The same for options of tab “Advanced options” (see section 6.10.2).

In the “Search algorithm” frame, you can specify whether you want to perform the locate operation on the text using the `Locate` program or on the text automaton with `LocateTfst`. By default, search is done with the `Locate` program, as `Unitex` always did until now. If you want to use `LocateTfst`, please read dedicated section 7.7.

Enter an expression and click on “Search” in order to start the search. `Unitex` will transform the expression into a grammar in the `.grf` format. This grammar will then be compiled into a grammar of the `.fst2` format that will be used for the search.

4.8.2 Presentation of the results

When the search is finished, the window of figure 4.5 appears showing the number of matched occurrences, the number of recognized tokens and the ratio between this number and the total number of tokens in the text.

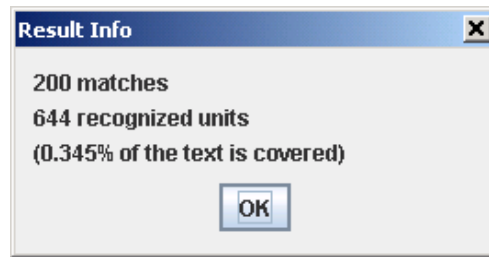


Figure 4.5: Search results

After having clicked on "OK" you will see window 4.6 appear, which allows you to configure the presentation of the matched occurrences. You can also open this window by clicking on "Located Sequences..." in the "Text" menu. The list of occurrences is called a *concordance*.

The "Modify text" box offers the possibility to replace the matched occurrences with the generated outputs. This possibility will be examined in chapter 6.

The "Extract units" box allows you to create a text file with all the sentences that do or do not contain matched units. With the button "Set File", you can select the output file. Then click on "Extract matching units" or "Extract unmatching units" depending on whether you are interested in sentences with or without matching units.

In the "Show matching sequences in context" box, you can select the length in characters of the left and right contexts of the occurrences that will be presented in the concordance. If an occurrence has less characters than its right context, the line will be completed with the necessary number of characters. If an occurrence has a length greater than that of the right context, it will be displayed completely.

NOTE: in Thai, the size of the contexts is measured in displayable characters and not in real characters. This makes it possible to keep the line alignment in the concordance despite the presence of diacritics that combine with other letters instead of being displayed as normal characters.

You can choose the sort order in the list "Sort According to". The mode "Text Order" displays the occurrences in the order of their appearance in the text. The other six modes allow you to sort in columns. The three zones of a line are the left context, the occurrence and the right context. The occurrences and the right contexts are sorted from left to right. The left contexts are sorted from right to left. The default mode is "Center, Left Col.". The concordance is generated in the form of an HTML file.

If a concordance reaches several thousands of occurrences, it is advisable to display it in a web browser (Firefox [11], Netscape [12], Internet Explorer, etc.) instead. Check "Use a web browser to view the concordance" (cf. figure 4.6). This option is activated by default if the number of occurrences is greater than 2000. You can configure which web browser to use by

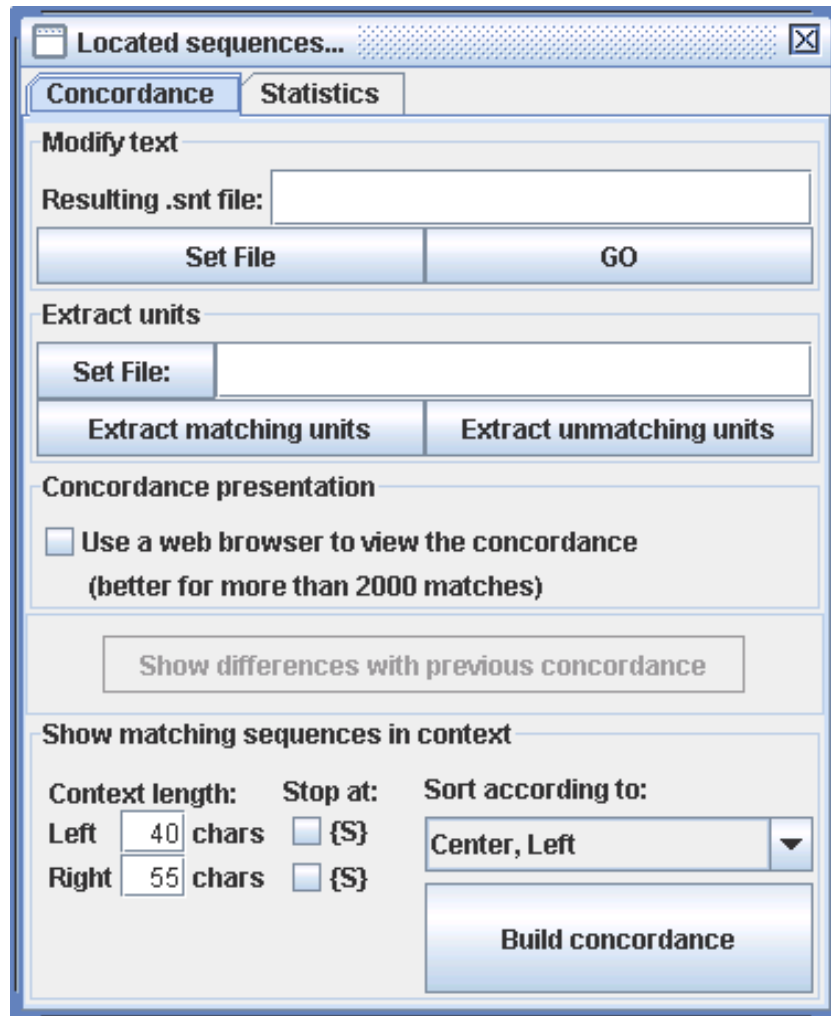


Figure 4.6: Result display configuration

clicking on "Preferences..." in the menu "Info". Click on the tab "Language & Presentation" and select the program to use in the field "Html Viewer" (cf. figure 4.7).

If you choose to open the concordance in Unitex, you will see a window as shown on Figure 4.8. Utterances react as hyperlinks. If you click on an occurrence, the text frame is opened and the corresponding sequence is highlighted. Moreover, if the text automaton is available and if this window is not iconified, the sentence automaton that contains the occurrence will be shown.

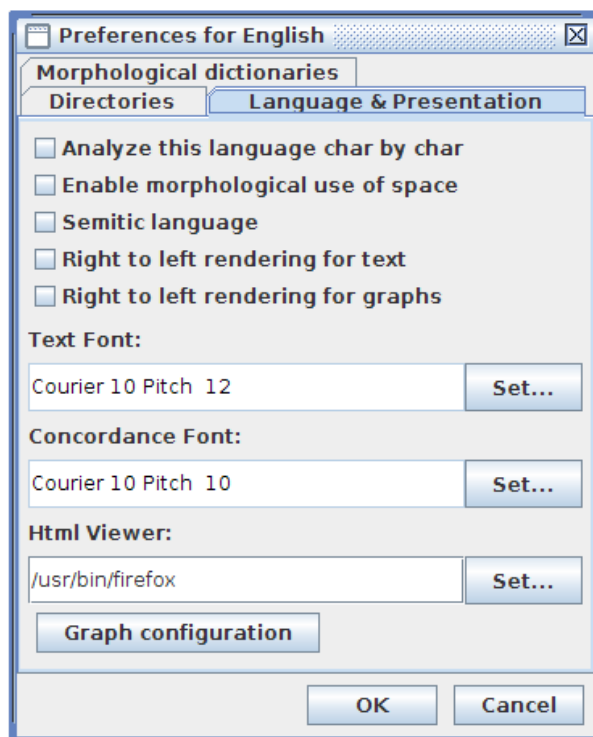


Figure 4.7: Selection of a web browser for displaying concordances

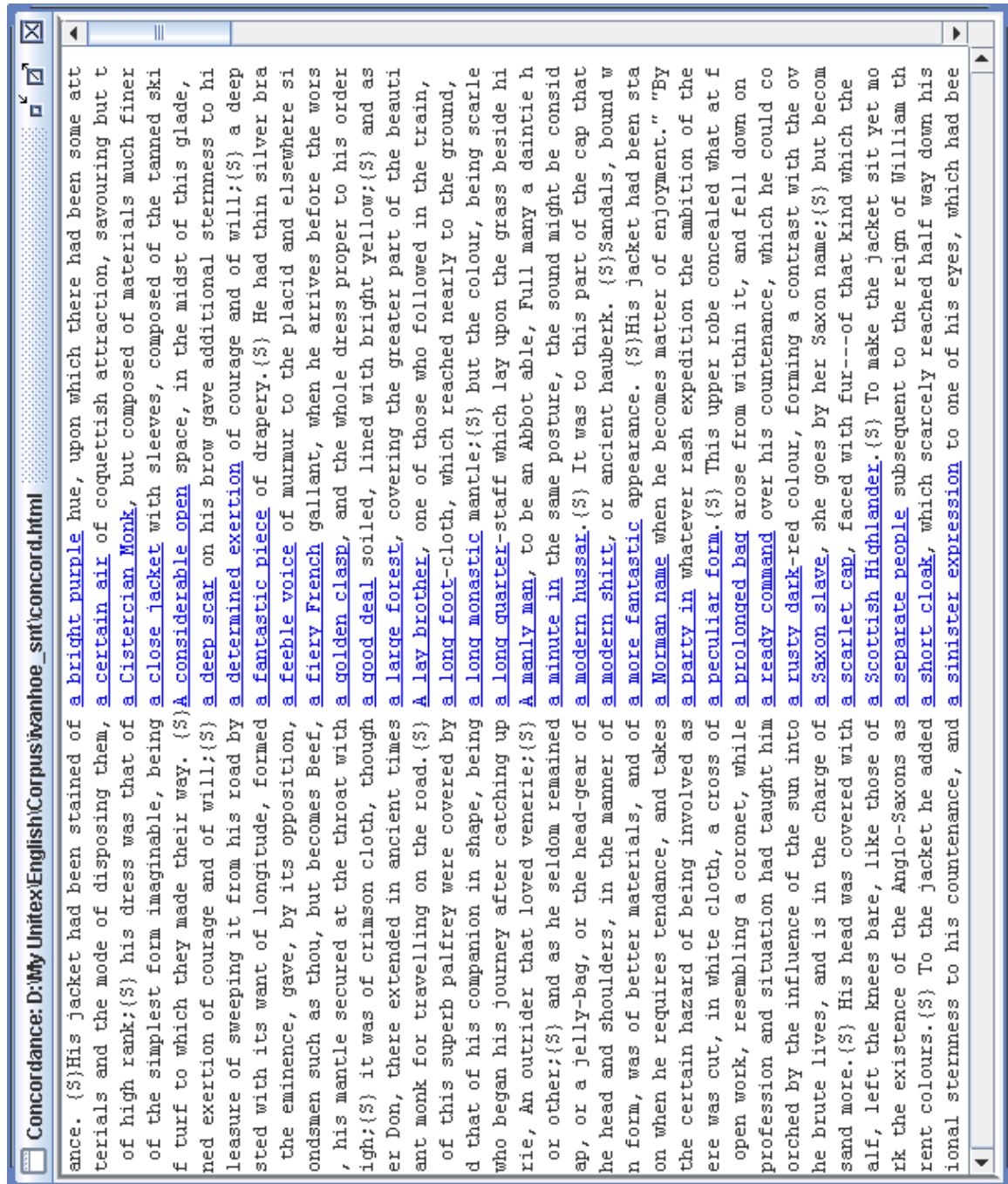


Figure 4.8: Example concordance

4.8.3 Statistics

If you select the “Statistics” tab in the “Located sequences..” frame, you will see the panel shown on figure 4.9. This panel allows you to get some statistics from the previously indexed sequences.

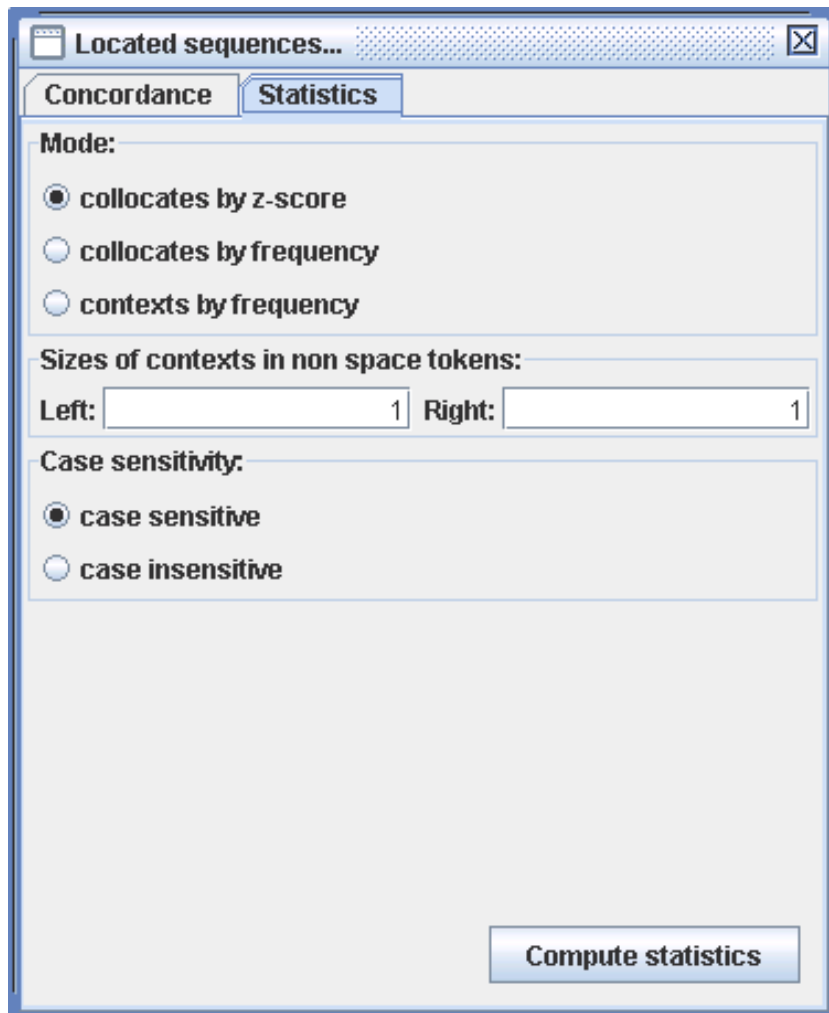


Figure 4.9: Statistics panel

In the “Mode” panel, you can select the kind of statistics you want:

- collocates by z-score: the previous one, plus some additional information (number of occurrences of the collocate in the match context and in the whole corpus, z-score of the collocate)
- collocates by frequency: shows the tokens that cooccur in the match context

- contexts by frequency: shows matches with left and right contexts (see below). “count” is the number of occurrences of a given match+context

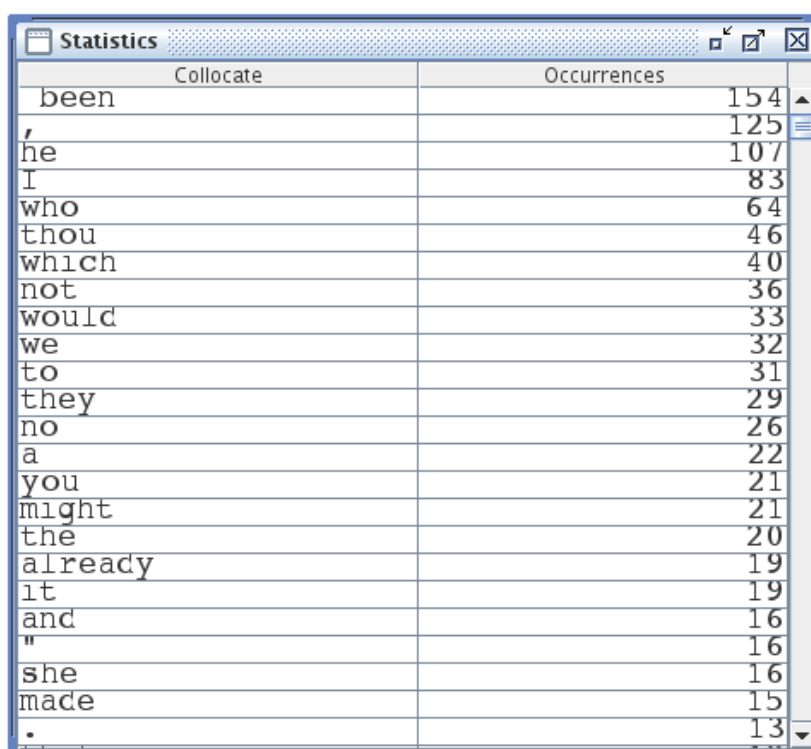
In the second panel, you can set the length of left and right contexts to be used, in non space tokens. NOTE: this notion of context has nothing to do with contexts in grammars.

In the last panel, you can allow or not case variations. If you allow case variations, the and THE will be considered as a same token, and the count will be the sum of the counts of the and THE.

The following figures show the statistics computed in each mode for the query <have> on `ivanhoe.snt`.

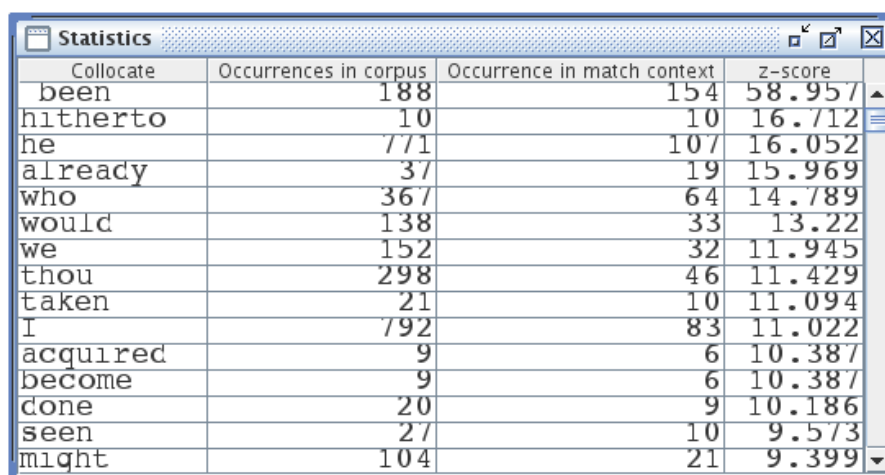
Left context	Match	Right context	Occurrences
would	have	been	10
which	had	been	10
,	had	been	7
might	have	been	6
,	had	not	5
he	had	been	5
I	have	been	5
to	have	been	5
it	had	been	5
we	have	already	4
he	had	hitherto	4
who	had	been	4
,	had	he	4
I	have	a	4
would	have	thought	4
,	had	they	3
,	having	been	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: left+match+right count



Collocate	Occurrences
been	154
,	125
he	107
I	83
who	64
thou	46
which	40
not	36
would	33
we	32
to	31
they	29
no	26
a	22
you	21
might	21
the	20
already	19
it	19
and	16
"	16
she	16
made	15
.	13

Figure 4.11: collocate count



Collocate	Occurrences in corpus	Occurrence in match context	z-score
been	188	154	58.957
hitherto	10	10	16.712
he	771	107	16.052
already	37	19	15.969
who	367	64	14.789
would	138	33	13.22
we	152	32	11.945
thou	298	46	11.429
taken	21	10	11.094
I	792	83	11.022
acquired	9	6	10.387
become	9	6	10.387
done	20	9	10.186
seen	27	10	9.573
might	104	21	9.399

Figure 4.12: collocate, count and other information

Chapter 5

Local grammars

Local grammars are a powerful tool to represent the majority of linguistic phenomena. The first section presents the formalism in which these grammars are represented. Then we will see how to construct and present grammars using Unitex.

5.1 The local grammar formalism

5.1.1 Algebraic grammars

Unitex grammars are variants of algebraic grammars, also known as context-free grammars. An algebraic grammar consists of rewriting rules. Below you see a grammar that matches any number of a characters:

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

The symbols to the left of the rules are called *non-terminal symbols* since they can be replaced. Symbols that cannot be replaced by other rules are called *terminal symbols*. The items at the right side are sequences of non-terminal and terminal symbols. The epsilon symbol ε designates the empty word. In the grammar above, S is a non-terminal symbol and a a terminal (symbol). S can be rewritten as either an a followed by a S or as the empty word. The operation of rewriting by applying a rule is called *derivation*. We say that a grammar generates a word if there exists a sequence of derivations that produces that word. The non-terminal that is the starting point of the first derivation is called an *axiom*.

The grammar above also generates the word aa , since we can derive this word according to the axiom S by applying the following derivations:

Derivation 1: rewriting the axiom to aS

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

Derivation 2: rewriting S at the right side of aS

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

Derivation 3: rewriting S to ε

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

We call the set of words generated by a grammar the *language generated by the grammar*. The languages generated by algebraic grammars are called *algebraic languages* or *context-free languages*.

5.1.2 Extended algebraic grammars

Extended algebraic grammars are algebraic grammars where the members on the right side of the rule are not just sequences of symbols but regular expressions. Thus, the grammar that generates a sequence of an arbitrary number of a 's can be written as a grammar consisting of one rule:

$$S \rightarrow a^*$$

These grammars, also called *recursive transition networks (RTN)* or *syntax diagrams*, are suited for a user-friendly graphical representation. Indeed, the right member of a rule can be represented as a graph whose name is the left member of the rule.

However, Unitex grammars are not exactly extended algebraic grammars, since they contain the notion of *transduction*. This notion, which is derived from the field of finite state automata, enables a grammar to produce some output. With an eye towards clarity, we will use the terms grammar or graph. When a grammar produces outputs, we will use the term *transducer*, as an extension of the definition of a transducer in the area of finite state automata.

5.2 Editing graphs

5.2.1 Creating a graph

In order to create a graph, click on "New" in the "FSGraph" menu. You will then see the window coming up as in figure 5.2. The symbol in arrow form is the *initial state* of the graph. The round symbol with a square is the *final state* of the graph. The grammar only recognizes expressions that are described along the paths between initial and final states.

In order to create a box, click inside the window while pressing the Ctrl key. A blue rectangle will appear that symbolizes the empty box that was created (see figure 5.3). After creating the box, it is automatically selected.

You see the contents of that box in the text field at the top of the window. The newly created box contains the $\langle \varepsilon \rangle$ symbol that represents the empty word epsilon. Replace this symbol by the text `I+you+he+she+it+we+they` and press the Enter key. You see that the box now contains seven lines (see figure 5.4). The + character serves as a separator. The box is displayed in the form of red text lines since it is not connected to another one at the moment. We often use this type of boxes to insert comments into a graph.

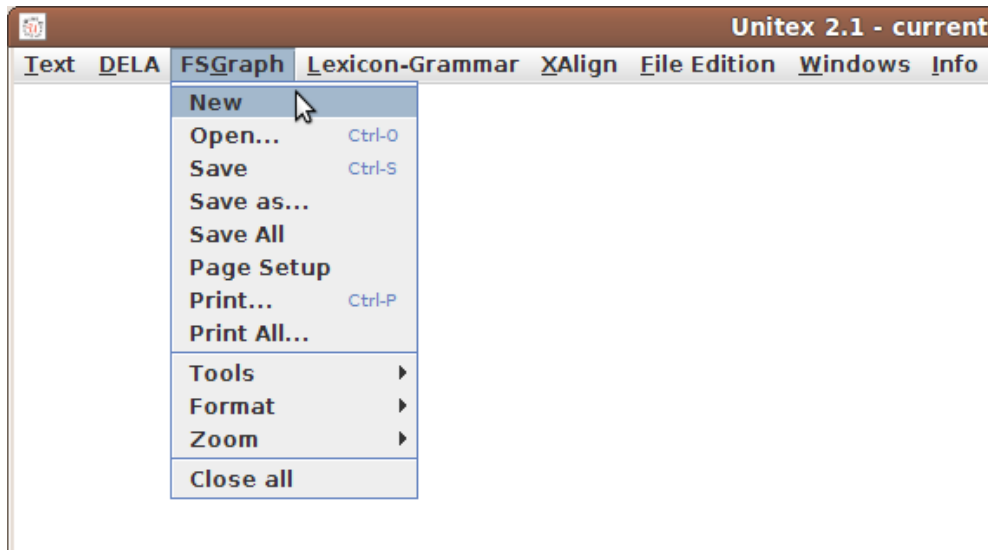


Figure 5.1: FSGraph menu

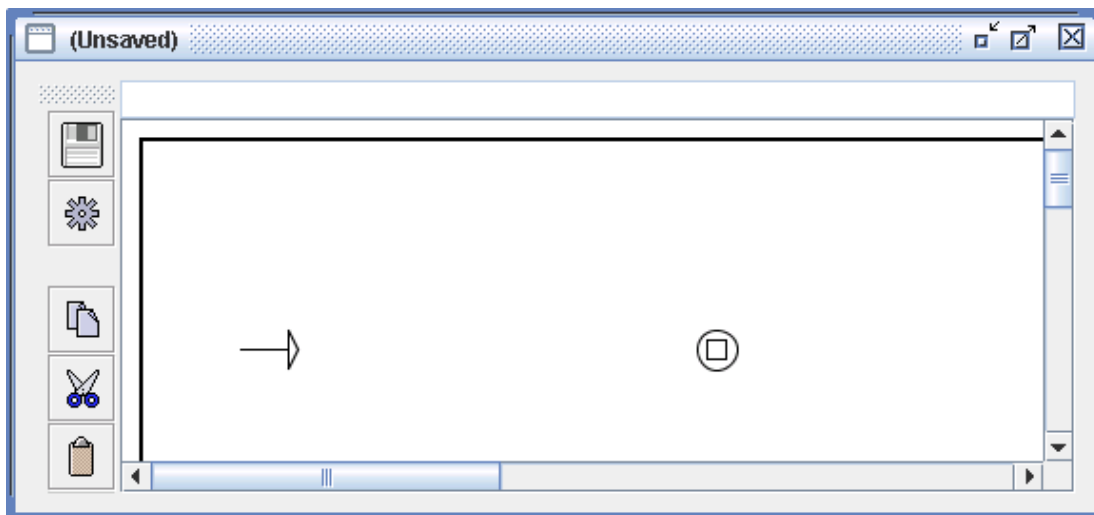


Figure 5.2: Empty graph

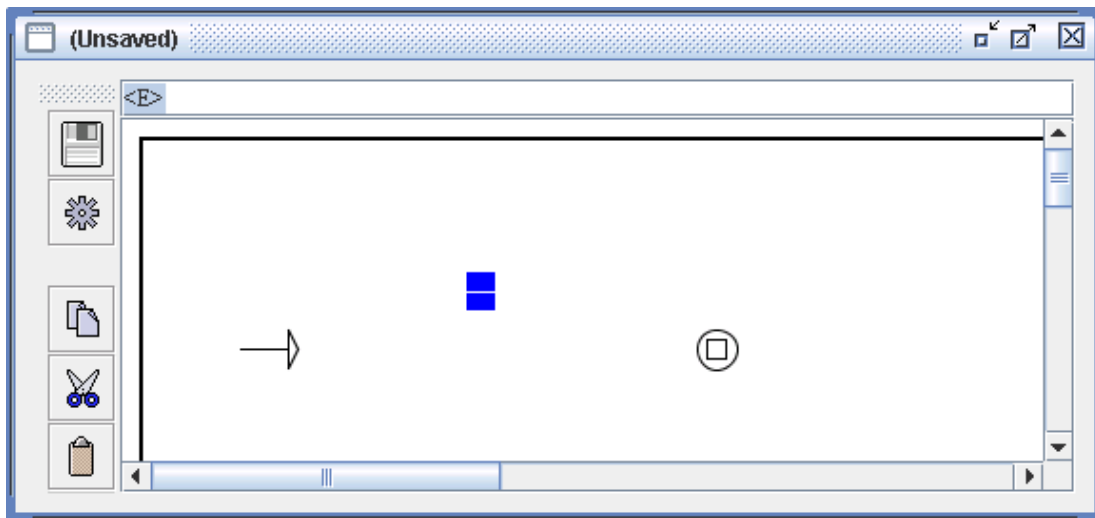


Figure 5.3: Creating a box

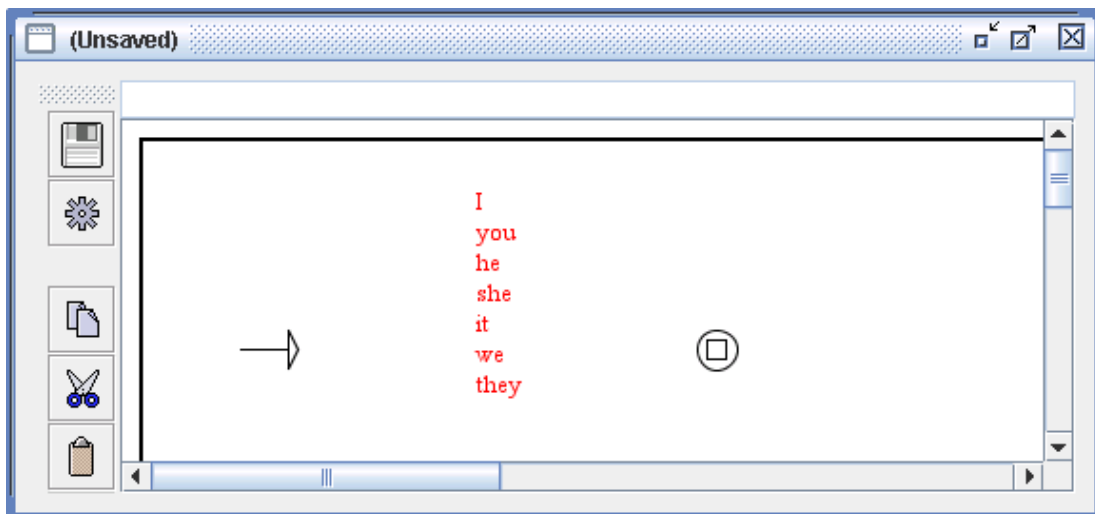


Figure 5.4: Box containing I+you+he+she+it+we+they

To connect a box to another one, first click on the source box, then click on the target box. If there already exists a transition between two boxes, it is deleted. It is also possible to do that by clicking first on the target box and then on the source box while pressing Shift. In our example, after connecting the box to the initial and final states of the graph, we get a graph as in figure 5.5:

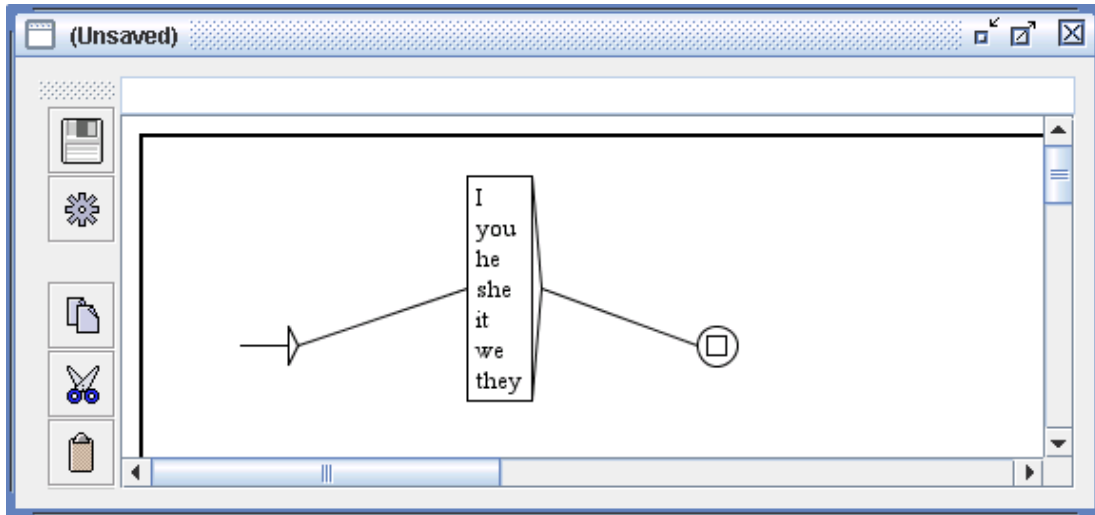


Figure 5.5: Graph that recognizes English pronouns

NOTE: If you double-click a box, you connect this box to itself (see figure 5.6). To undo this double-click on the same box a second time, or use the "Undo" button.

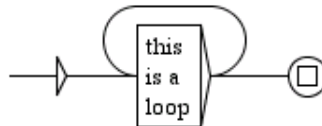


Figure 5.6: Box connected to itself

Click on "Save as..." in the "FSGraph" menu to save the graph. By default, Unitex proposes to save the graph in the sub-directory `Graphs` in your personal folder. You can see if the graph was modified after the last saving by checking if the title contains the text `(Unsaved)`.

5.2.2 Sub-Graphs

In order to call a sub-graph, its name is inserted into a box and preceded by the `:` character. If you enter the text:

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

into a box, you get a box similar to the one in figure 5.7:

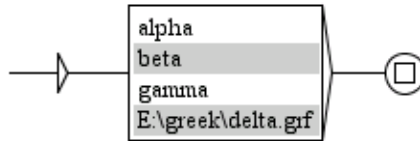


Figure 5.7: Graph that calls sub-graphs `beta` and `delta`

You can indicate the full name of the graph (`E:\greek\delta.grf`) or simply the base name without the path (`beta`); in this case, the sub-graph is expected to be in the same directory as the graph that references it. References to absolute path names should as a rule be avoided, since such calls are not portable. If you use such an absolute path name, the graph compiler will emit a warning (see figure 5.8).

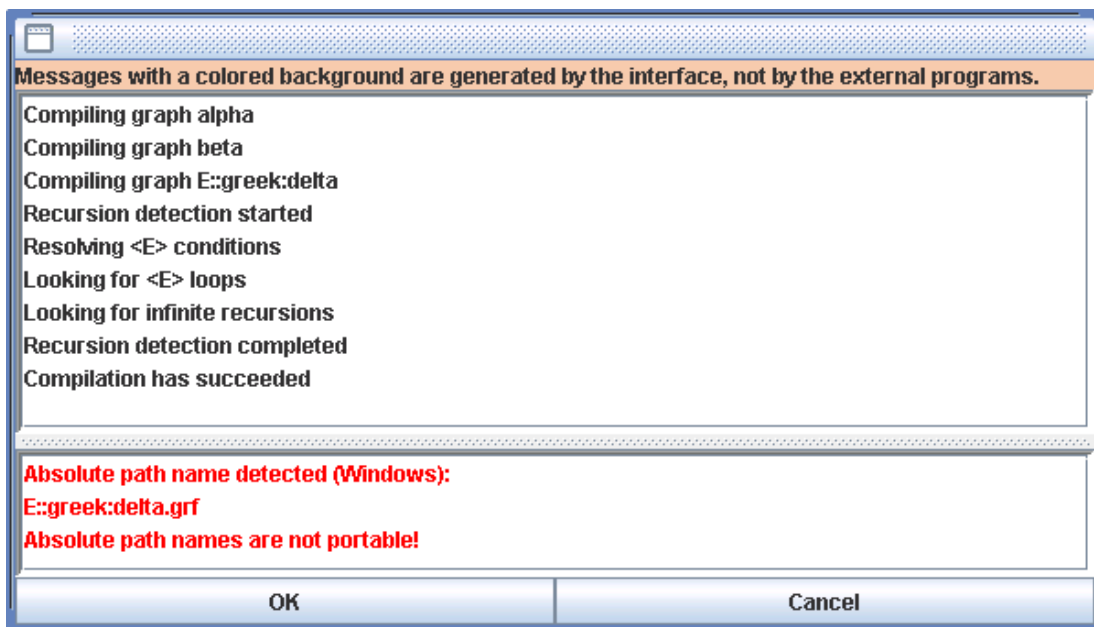


Figure 5.8: Warning about a non portable graph name

For portability you should not use `\` or `/` as separator in graph path names. Use instead `:` which is understood as a system-independent separator. In figure 5.8 `\` and `/` are internally converted by the graph compiler to `:` (`E::greek:delta.grf`).

Graph repository

When you need to call a grammar `X` inside a grammar `Y`, a simple method is to copy all

the graphs of X into the directory that contains the graphs of Y . This method raises two problems:

- the number of graphs in the directory grows quickly;
- two graphs cannot share the same name.

To avoid that, you can store the grammar X in a special directory, called the *graph repository*. This directory is a kind of library where you can store graphs, and then call them using `::` instead of `:.` To use this mechanism, you first need to set the path to the graph repository. Go into the "Info>Preferences...>Directories" menu, and select your directory in the "Graph repository" frame (see Figure 5.9). There is one graph repository per language, so feel free to share or not the same directory for all the languages you work with.

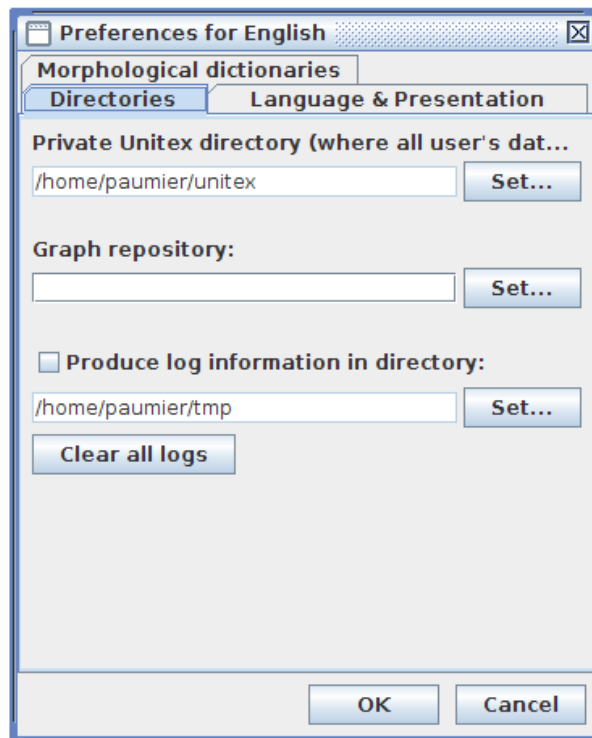


Figure 5.9: Setting the path to the graph repository

Let us assume that we have a repository tree as on Figure 5.10. If we want to call the graph named `DET` that is located in sub-directory `Johnson`, we must use the call `::Det:Johnson:DET` (see Figure 5.11¹).



Figure 5.10: Graph repository example

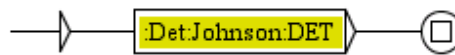


Figure 5.11: Call to a graph located in the repository

TRICK: If you want to avoid long path names like `::Det:Johnson:DET`, you can create a graph named `DET` and put it the repository root (here `D:\repository\DET.grf`). In this graph, just put a call to `::Det:Johnson:DET`. Then, you can just call `::DET` in your own graphs. This has two advantages: 1) you do not have long path names; 2) you can modify the graphs in your repository with no constraint on your own graphs, because the only graph that will have to be modified is the one located at the repository root.

Calls to sub-graphs are represented in the boxes by grey lines, or brown lines in the case of graphs located in the repository. On Windows, you can open a sub-graph by clicking on the grey line while pressing the `Alt` key. On Linux, the combination `<Alt+Click>` is intercepted by the system.² In order to open a sub-graph, click on its name by pressing the left and the right mouse button simultaneously.

5.2.3 Manipulating boxes

You can select several boxes using the mouse. In order to do so, click and drag the mouse without releasing the button. When you release the button, all boxes touched by the selection rectangle will be selected and are displayed in white on blue ground, as shown on Figure 5.12.

When boxes are selected, you can move them by clicking and dragging the cursor without releasing the button. In order to cancel the selection, click on an empty area of the graph. If you click on a box, all boxes of the selection will be connected to it.

¹To avoid confusion, graph calls that refer to the repository are displayed in brown instead of grey.

²If you are working on KDE, you can deactivate `<Alt+Click>` in `kcontrol`.

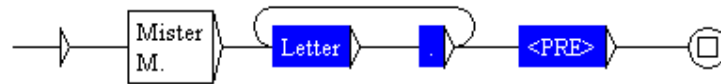


Figure 5.12: Selecting several boxes

You can perform a copy-paste with several boxes. Select them and press <Ctrl+C> or click on "Copy" in the "Edit" menu. The selection is now in the Unitex clipboard. You can then paste this selection by pressing <Ctrl+V> or by selecting "Paste" in the "Edit" menu.

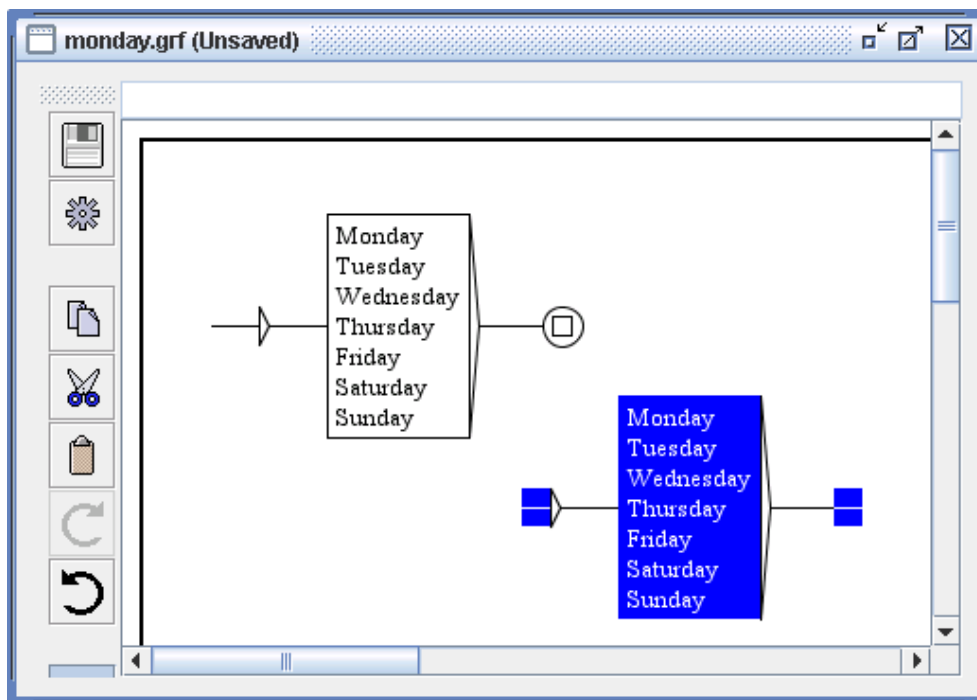


Figure 5.13: Copy-Paste of a multiple selection

NOTE: You can paste a multiple selection into a different graph than the one where you copied it from.

In order to delete boxes, select them, delete the text that they contain (*i.e.* the text presented in the text field above the window) and press the Enter key. The initial and final states cannot be deleted.

5.2.4 Transducers

A transducer is a graph in which outputs can be associated with boxes. To insert an output, use the special character /. All characters to the right of it will be part of the output. Thus,

the text `one+two+three/number` results in a box like in figure 5.14.

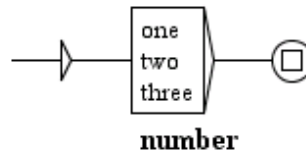


Figure 5.14: Example of a transducer

The output associated with a box is represented in bold text below it.

5.2.5 Using Variables

It is possible to select parts of a text sequence recognized by a grammar using variables. To associate a variable `var1` with parts of a grammar, use the special symbols `$var1 (` (and `$var1)`) to define the beginning and the end of the part to store. Create two boxes containing one `$var1 (` (and the second `$var1)`). These boxes must not contain anything but the variable name preceded by `$` and followed by a parenthesis. Then link these boxes to the zone of the grammar to store. In the graph in figure 5.15 you see a sequence of digits before `dollar` or `dollars`. This sequence will be stored in a variable named `var1`.

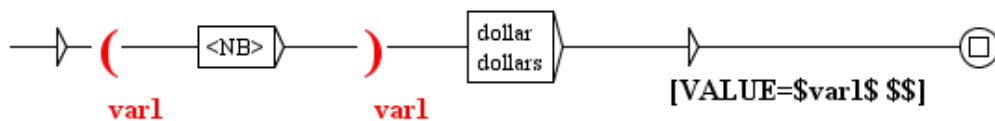


Figure 5.15: Using the variable `var1`

Variable names may contain latin letters (without accents), upper or lower case, numbers, or the `_` (underscore) character. Unitex distinguishes between uppercase and lowercase characters.

When a variable is defined, you can use it in transducer outputs by surrounding its name with `$`. The grammar in figure 5.16 recognizes a date formed by a month and a year, and produces the same date as an output, but in the order year-month.

If you want to use the character `$` in the output of a box, you have to double it, as shown on figure 5.15.

The default behavior of `Locate` and `LocateTfst` is to consider variables that have not been defined as being empty. You can modify this behavior as shown in section 6.10.2. Moreover, it is possible to test whether a variable has been defined or not, as shown in section 6.7.5.

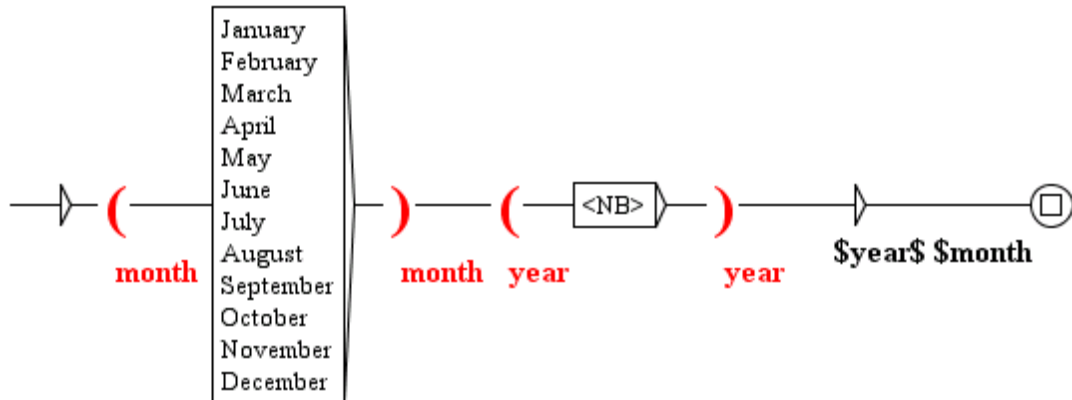


Figure 5.16: Inverting month and year in a date

5.2.6 Copying lists

It can be practical to perform a copy-paste operation on a list of words or expressions from a text editor to a box in a graph. In order to avoid having to copy every term manually, Unitex provides a mean to copy lists. To use this, select the list in your text editor and copy it using `<Ctrl+C>` or the copy function integrated in your editor. Then create a box in your graph, and press `<Ctrl+V>` or use the "Paste" command in the "Edit" menu to paste it into the box. A window as in Figure 5.17 opens:

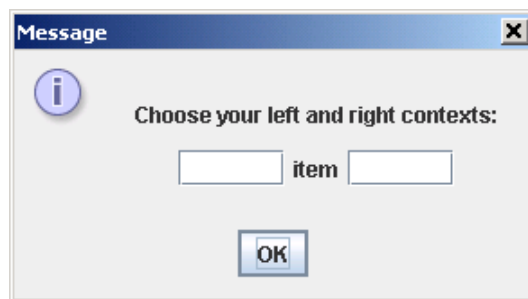


Figure 5.17: Selecting a context for copying a list

This window allows you to define the left and right contexts that will automatically be used for each term of the list. By default, these contexts are empty. If you use the contexts `<` and `.v>` with the following list:

eat
sleep
drink
play

read

you will get the box in figure 5.18:

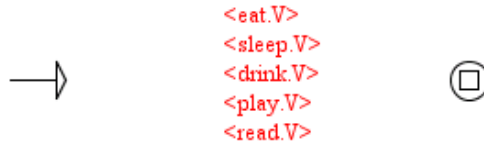


Figure 5.18: Box resulting from copying a list and applying contexts

5.2.7 Special Symbols

The Uunitex graph editor interprets the following symbol in a special manner:

" + : / < > # \

Table 5.1 summarizes the meaning of these symbols for Uunitex, as well as the ways to recognize these characters in texts.

Character	Meaning	Escape
"	quotation marks mark sequences that must not be interpreted by Uunitex, and whose case must be taken verbatim	\"
+	+ separates different lines within the boxes	"+"
:	: introduces a call to a subgraph	": " or \:
/	/ indicates the start of a transduction within a box	\/
<	< indicates the start of a pattern or a meta	"<" or \<
>	> indicates the end of a pattern or a meta	">" or \>
#	# prohibits the presence of a space	"#"
\	\ escapes most of the special characters	\\

Table 5.1: Encoding of special characters in the graph editor

5.2.8 Toolbar Commands

The toolbar on the left of a graph contains shortcuts for certain commands and allows you to manipulate boxes of a graph by using some "tools". This toolbar may be moved by clicking on the "rough" zone. It may also be dissociated from the graph and appear in an separate



Figure 5.19: Toolbar

window (see figure 5.19). In this case, closing this window puts the toolbar back at its initial position. Each graph has its own toolbar.

The first two icons are shortcuts for saving and compiling the graph. The following five correspond to the Copy, Cut, Paste, Redo and Undo operations. The last icon showing a key is a shortcut to open the window with the graph display options.

The other six icons correspond to edit commands for boxes. The first one, a white arrow, corresponds to the boxes' normal edit mode. The 5 others correspond to specific tools. In order to use a tool, click on the corresponding icon: The mouse cursor changes its form and mouse clicks are then interpreted in a particular fashion. What follows is a description of these tools, from left to right:

- creating boxes: creates a box at the empty place where the mouse was clicked;
- deleting boxes: deletes the box that you click on;
- connect boxes to another box: using this utility you select one or more boxes and connect it or them to another one. In contrast to the normal mode, the connections are inserted to the box where the mouse button was released on;
- connect boxes to another box in the opposite direction: this utility performs the same operation as the one described above, but connects the boxes to the one clicked on in opposite direction;
- open a sub-graph: opens a sub-graph when you click on a grey line within a box.

5.3 Display options

5.3.1 Sorting the lines of a box

You can sort the content of a box by selecting it and clicking on "Sort Node Label" in the "Tools" submenu of the "FSGraph" menu. This sort operation does not use the `SortTxt` program. It uses a basic sort mechanism that sorts the lines of the box according to the order of the characters in the Unicode encoding.

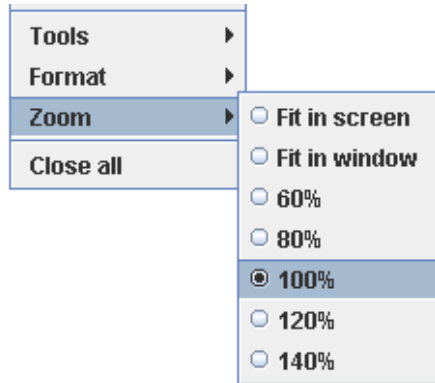


Figure 5.20: Zoom sub-menu

5.3.2 Zoom

The "Zoom" submenu allows you to choose the zoom scale that is applied to display the graph.

The "Fit in screen" option stretches or shrinks the graph in order to fit it into the screen. The "Fit in window" option adjusts the graph so that it is displayed entirely in the window.

5.3.3 Antialiasing

Antialiasing is a shading effect that avoids pixelization effects. You can activate this effect by clicking on "Antialiasing..." in the "Format" sub-menu. Figure 5.21 shows one graph displayed normally (the graph on top) and with antialiasing (the graph at the bottom).

This effect slows Unitex down. We recommend not to use it if your machine is not powerful enough.

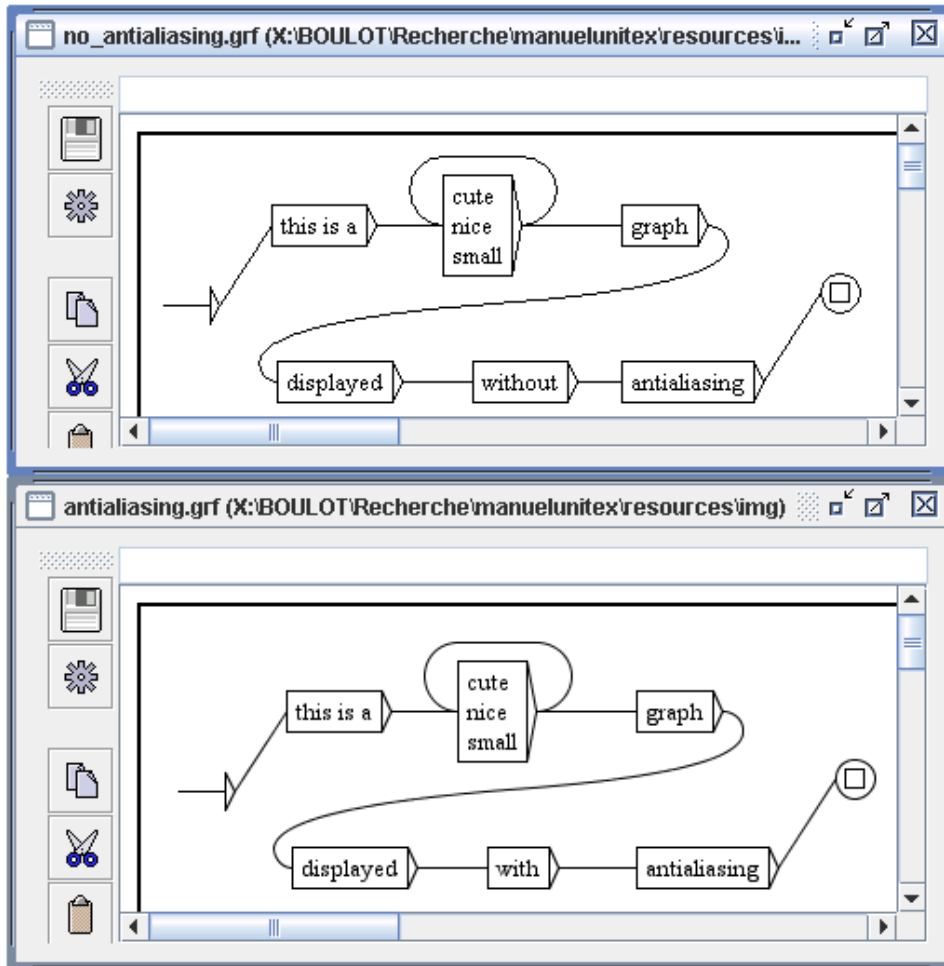


Figure 5.21: Antialiasing example

5.3.4 Box alignment

In order to get nice-looking graphs, it is useful to align the boxes, both horizontally and vertically. To do this, select the boxes to align and click on "Alignment..." in the "Format" sub-menu of the "FSGraph" menu or press <Ctrl+M>. You will then see the window in Figure 5.22.

The possibilities for horizontal alignment are:

- Top: boxes are aligned with the top-most box;
- Center: boxes are centered on the same axis;
- Bottom: boxes are aligned with the bottom-most box.

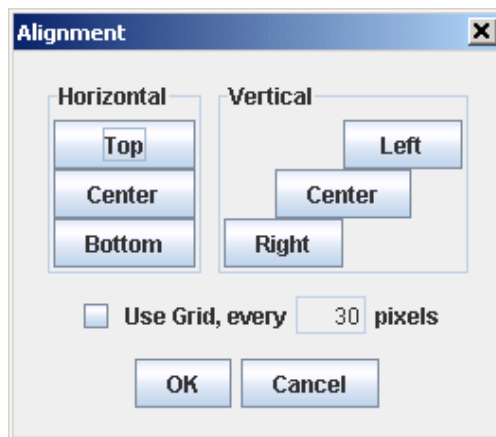


Figure 5.22: Alignment window

The possibilities for vertical alignment are:

- Left: boxes are aligned with the left-most box;
- Center: boxes are centered on the same axis;
- Right: boxes are aligned with the right-most box.

Figure 5.23 shows an example of alignment. The group of boxes to the right is (quite) a copy of the ones to the left that was aligned.

The option "Use Grid" in the alignment window shows a grid as the background of the graph. This allows you to approximately align the boxes.

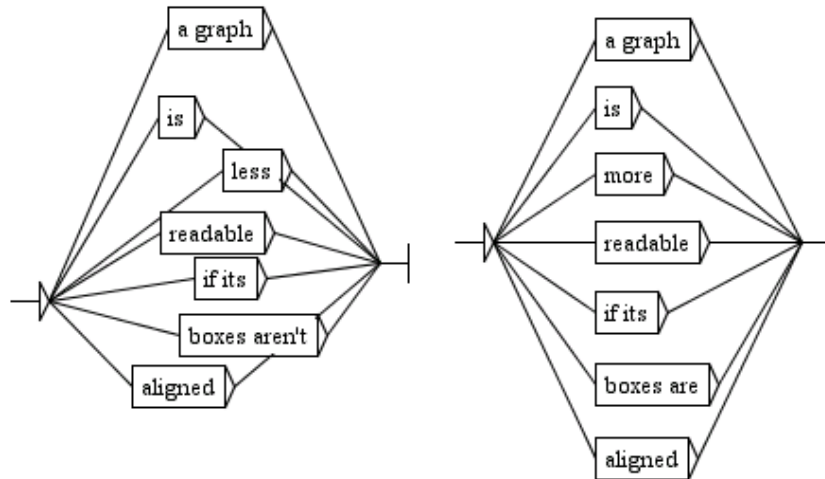


Figure 5.23: Example of box alignment

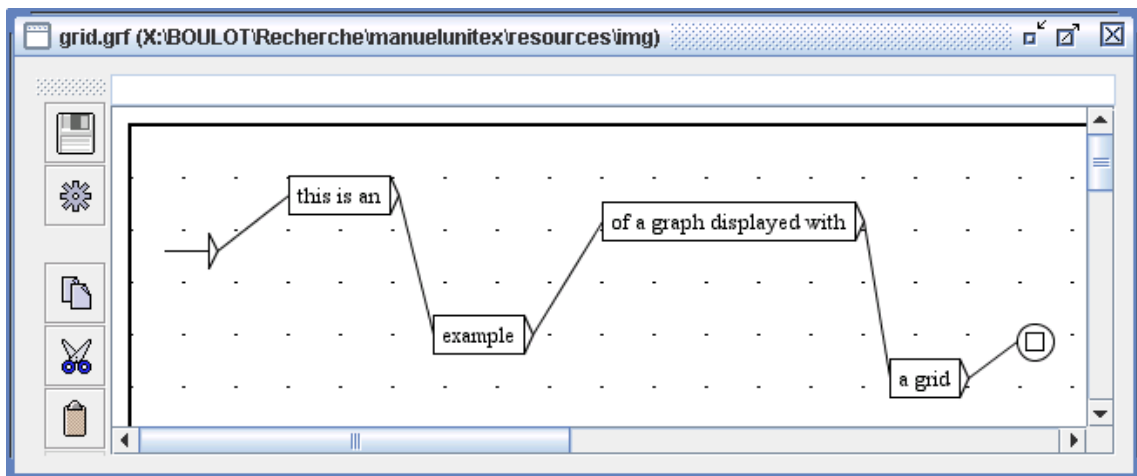


Figure 5.24: Example of using the grid

5.3.5 Display options, fonts and colors

You can configure the display style of a graph by pressing <Ctrl+R> or by clicking on "Presentation..." in the "Format" sub-menu of the "FSGraph" menu, which opens the window as in figure 5.25.

The font parameters are:

- Input: font used within the boxes and in the text area where the contents of the boxes is edited;
- Output: font used for the attached transducer outputs.

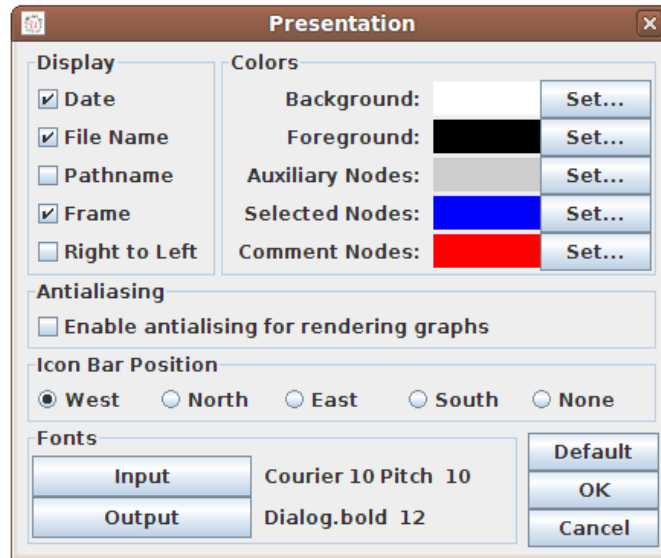


Figure 5.25: Configuring the display options of a graph

The color parameters are:

- Background: the background color;
- Foreground: the color used for the text and for the box display;
- Auxiliary Nodes: the color used for calls to sub-graphs;
- Selected Nodes: the color used for selected boxes;
- Comment Nodes: the color used for boxes that are not connected to others.

The other parameters are:

- Date: display of the current date in the lower left corner of the graph;
- File Name: display of the graph name in the lower left corner of the graph;
- Pathname: display of the graph name along with its complete path in the lower left corner of the graph. This option only has an effect if the option "File Name" is selected;
- Frame: draw a frame around the graph;
- Right to Left: invert the reading direction of the graph (see an example in figure 5.26).

You can reset the parameters to the default ones by clicking on "Default". If you click on "OK", only the current graph will be modified. In order to modify the preferences for a language as a default, click on "Preferences..." in the "Info" menu and click on the "Graph configuration" button in the "Language & Presentation" tab.

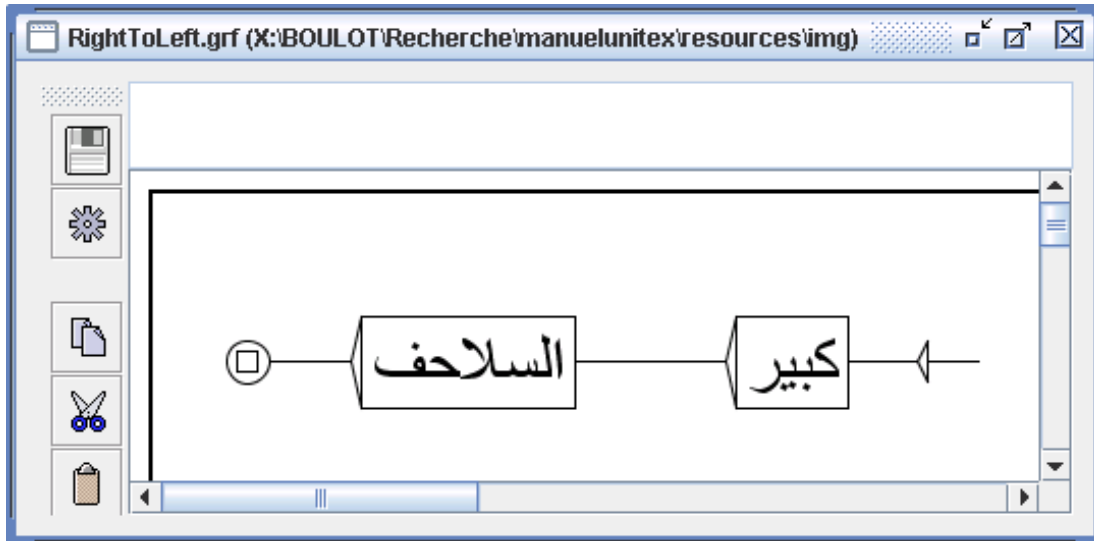


Figure 5.26: Graph with reading direction set to right to left

5.4 Exporting graphs

5.4.1 Inserting a graph into a document

In order to include a graph into a document, you have to convert it to an image. To do this, save your graph as a PNG image. Click on "Save as..." in the "FSGraph" menu, and select the PNG file format. You will get an image ready to be inserted into a document, or to be edited with an image editor. You should activate antialiasing for the graph that interests you (this is not obligatory but results in a better image quality).

Another solution consists of making a screenshot:

On Windows:

Press "Print Screen" on your keyboard. This key should be next to the F12 key. Start the `Paint` program in the Windows "Utilities" menu. Press `<Ctrl+V>`. `Paint` will tell you that the image in the clipboard is too large and asks if you want to enlarge the image. Click on "Yes". You can now edit the screen image. Select the area that interests you. To do so, switch to the select mode by clicking on the dashed rectangle symbol in the upper left corner of the window. You can now select the area of the image using the mouse. When you have selected the zone, press `<Ctrl+C>`. Your selection is now in the clipboard, you can now just go to your document and press `<Ctrl+V>` to paste your image.

On Linux:

Take a screen capture (for example using the program `xv`). Edit your image at once using a graphic editor (for example `TheGimp`), and paste your image in your document in the same way as in Windows.

Vector graphics

If you prefer vector graphics, you can save your graph under the SVG file format, which is editable with softwares like the Open Source one Inkscape ([24]). With this software, you can obtain PostScript exports ready to use in pretty L^AT_EX documents.

5.4.2 Printing a Graph

You can print a graph by clicking on "Print..." in the "FSGraph" menu or by pressing <Ctrl+P>.

WARNING: You should make sure that the page orientation parameter (portrait or landscape) corresponds to the orientation of your graph.

You can setup the printing preferences by clicking on "Page Setup" in the "FSGraph" menu. You can also print all open graphs by clicking on "Print All...".

Chapter 6

Advanced use of graphs

6.1 Types of graphs

Unitex can handle several types of graphs that correspond to the following uses: automatic inflection of dictionaries, preprocessing of texts, normalization of text automata, dictionary graphs, search for patterns, disambiguation and automatic graph generation. These different types of graphs are not interpreted in the same way by Unitex. Certain operations, like transduction, are allowed for some types and forbidden for others. In addition, special symbols are not the same depending on the type of graph. This section presents each type of graph and shows their peculiarities.

6.1.1 Inflection transducers

An inflection transducer describes the morphological variation that is associated with a word class by assigning inflectional codes to each variant. The paths of such a transducer describe the modifications that have to be applied to the canonical forms and the corresponding outputs contain the inflectional information that will be produced.

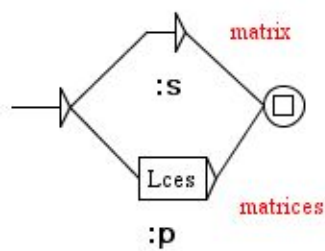


Figure 6.1: Example of an inflectional grammar

The paths may contain operators and letters. The possible operators are represented by the characters L, R, C, D, U,P and W. All letters that are not operators are characters. The only

allowed special symbol is the empty word `<E>`. It is not possible to refer to information in dictionaries in an inflection transducer, but it is possible to reference subgraphs.

Transducer outputs are concatenated in order to produce a string of characters. This string is then appended to the produced dictionary entry. Outputs with variables do not make sense in an inflection transducer.

Case of letters is respected: lowercase letters stay lowercase, the same for uppercase letters. Besides, the connection of two boxes is exactly equivalent to the concatenation of their contents together with the concatenation of their outputs. (cf. figure 6.2).

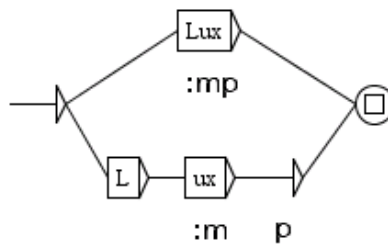


Figure 6.2: Two equivalent paths in an inflection grammar

Inflection transducers may be compiled before being used by the inflection program. If not, the inflection program will compile them on the fly.

For more details, see section 3.4.

6.1.2 Preprocessing graphs

Preprocessing graphs are meant to be applied to texts before they are tokenized into lexical units. These graphs can be used for inserting or replacing sequences in the texts. The two customary uses of these graphs are normalization of non-ambiguous forms and sentence boundary recognition.

The interpretation of these graphs in Unitex is very close to that of syntactic graphs used by the search for patterns. The differences are the following:

- you can use the special symbol `<^>` that recognizes a newline;
- if you work in character by character mode, you can use the special symbol `<L>` that recognizes one letter, as defined in the alphabet file;
- it is impossible to refer to information in dictionaries;
- it is impossible to use morphological filters;
- it is impossible to use morphological mode;

- it is impossible to use contexts.

The figures 2.9 (page 38) and 2.10 (page 40) show examples of preprocessing graphs.

6.1.3 Graphs for normalizing the text automaton

Graphs for normalizing the text automaton allow you to normalize ambiguous forms. They can describe several labels for the same form. These labels are then inserted into the text automaton thus making the ambiguity explicit. Figure 6.3 shows an extract of the normalization graph used by default for French.

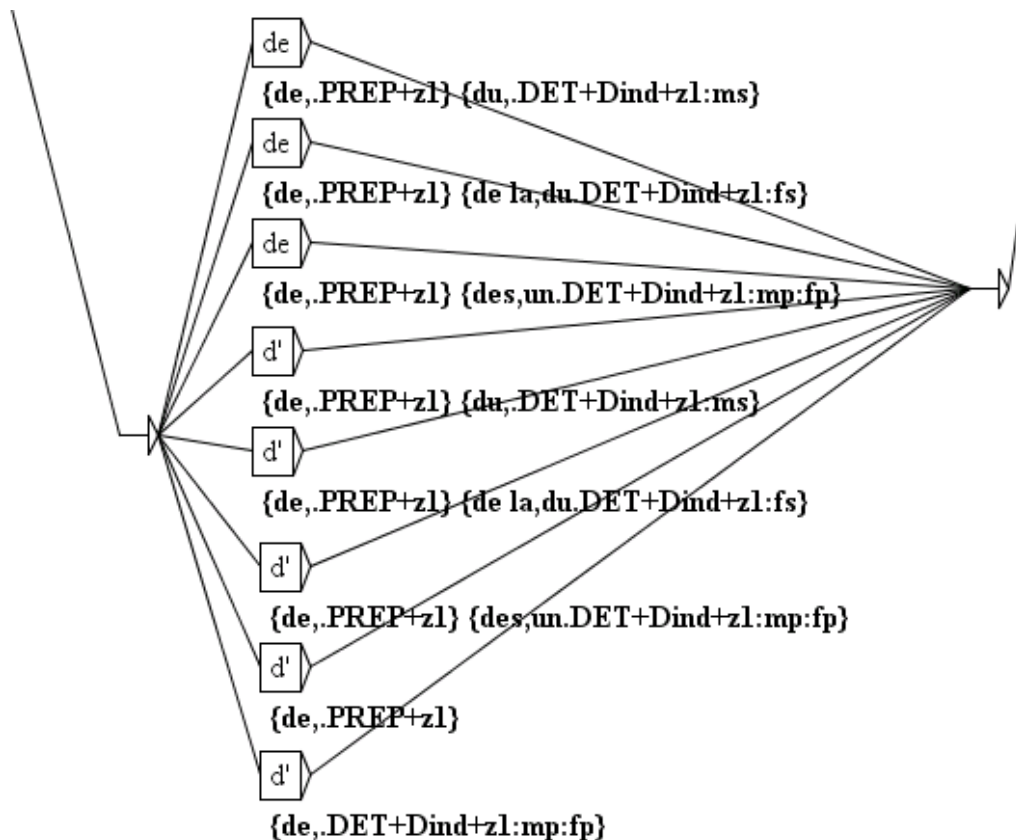


Figure 6.3: Extract of the normalization graph used for French

The paths describe the forms that have to be normalized. Lower case and upper case variants are taken into account according to the following principle: uppercase letters in the graph only recognize uppercase letters in the text automaton; lowercase letters can recognize both lowercase and uppercase letters.

The transducer outputs represent the sequences of labels that will be inserted into the text automaton. These labels can be dictionary entries or strings of characters. The labels that

represent dictionary entries have to respect the DELAF format and must be enclosed by the { and } symbols. Outputs with variables do not make sense in this kind of graph. You cannot use morphological filters, morphological mode or contexts.

It is possible to reference subgraphs. It is not possible to reference information in dictionaries in order to describe the forms to normalize. The only special symbol that is recognized in this type of graph is the empty word <E>. The graphs for normalizing ambiguous forms need to be compiled before using them.

6.1.4 Syntactic graphs

Syntactic graphs, often called local grammars, allow you to describe syntactic patterns that can then be searched in the texts. Of all kinds of graphs these have the greatest expressive power because they allow you to refer to information in dictionaries.

Lower case/upper case variants may be used according to the principle described above. It is still possible to enforce respect of case by enclosing an expression in double quotes. The use of double quotes also allows you to enforce the respect of spaces. In fact, Unitex by default assumes that a space is possible between two boxes. In order to enforce the presence of a space you have to enclose it in double quotes. For prohibiting the presence of a space you have to use the special symbol #.

Syntactic graphs can reference subgraphs (cf. section 5.2.2). They also have outputs including outputs with variables. The produced sequences are interpreted as strings of characters that will be inserted in the concordances or in the text if you want to modify it (cf. section 6.10.4).

Syntactic graphs can use contexts (see section 6.3).

Syntactic graphs can use morphological filters (see section 4.7).

Syntactic graphs can use morphological mode (see section 6.4).

The special symbols that are supported by the syntactic graphs are the same as those that are usable in regular expressions (cf. section 4.3.1).

It is not obligatory to compile syntactic graphs before using them for pattern matching. If a graph is not compiled the system will compile it automatically.

6.1.5 ELAG grammars

ELAG grammars for disambiguation between lexical symbols in text automata are described in section 7.3.1, page 157.

6.1.6 Parameterized graphs

Parameterized graphs are meta-graphs that allow you to generate a family of graphs using a lexicon-grammar table. It is possible to construct parameterized graphs for all possible kinds of graphs. The construction and use of parameterized graphs are explained in chapter 8.

6.2 Compilation of a grammar

6.2.1 Compilation of a graph

Compilation is the operation that converts the `.grf` format to a format that can be manipulated more easily by Unitex programs. In order to compile a graph, you must open it and then click on "Compile FST2" in the "Tools" submenu of the menu "FSGraph". Unitex then launches the `Grf2Fst2` program. You can keep track of its execution in a window (cf. Figure 6.4).

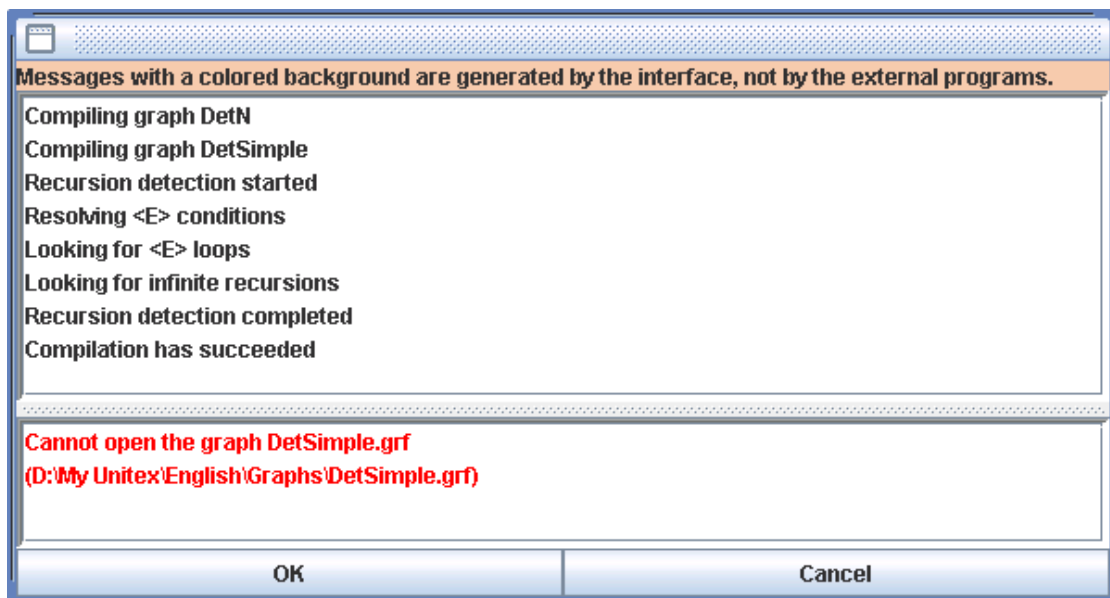


Figure 6.4: Compilation window

If the graph references subgraphs, those are automatically compiled. The result is a `.fst2` file that contains all the graphs that make up a grammar. The grammar is then ready to be used by Unitex programs.

6.2.2 Approximation with a finite state transducer

The FST2 format conserves the architecture in subgraphs of the grammars, which is what makes them different from strict finite state transducers. The `Flatten` program allows

you to turn a FST2 grammar into a finite state transducer whenever this is possible, and to construct an approximation if not. This function thus permits to obtain objects that are easier to manipulate and to which all classical algorithms on automata can be applied.

In order to compile and thus transform a grammar, select the command "Compile & Flatten FST2" in the "Tools" submenu of the "FSGraph" menu. The window of Figure 6.5 allows you to configure the approximation process.

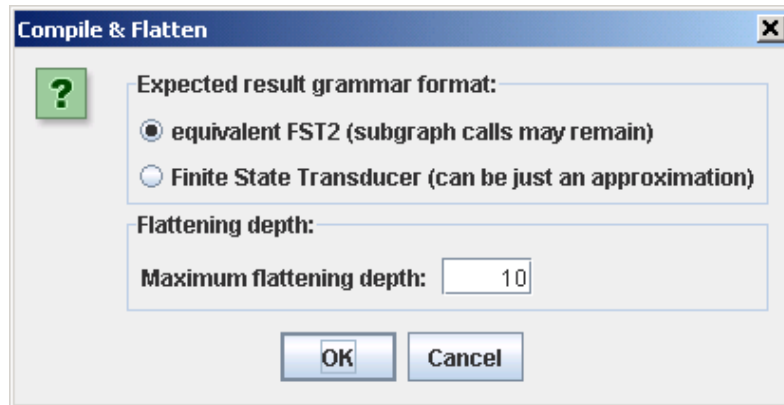


Figure 6.5: Configuration of approximation of a grammar

The box "Flattening depth" lets you specify the level of embedding of subgraphs. This value represents the maximum depth up to which the callings of subgraphs will be replaced by the subgraphs themselves.

The "Expected result grammar format" box allows you to determine the behavior of the program beyond the selected limit. If you select the "Finite State Transducer" option, the calls to subgraphs will be replaced by $\langle \epsilon \rangle$ beyond the maximum depth. This option guarantees that we obtain a finite state transducer, however possibly not equivalent to the original grammar. On the contrary, the "equivalent FST2" option indicates that the program should allow for subgraph calls beyond the limited depth. This option guarantees the strict equivalence of the result with the original grammar but does not necessarily produce a finite state transducer. This option can be used for optimizing certain grammars.

A message indicates at the end of the approximation process if the result is a finite state transducer or an FST2 grammar and in the case of a transducer if it is equivalent to the original grammar (cf. Figure 6.6).

6.2.3 Constraints on grammars

With the exception of inflection grammars, a grammar can never have an empty path. This means that the paths of a main graph must not recognize the empty word but this does not prevent a subgraph of that grammar from recognizing epsilon.

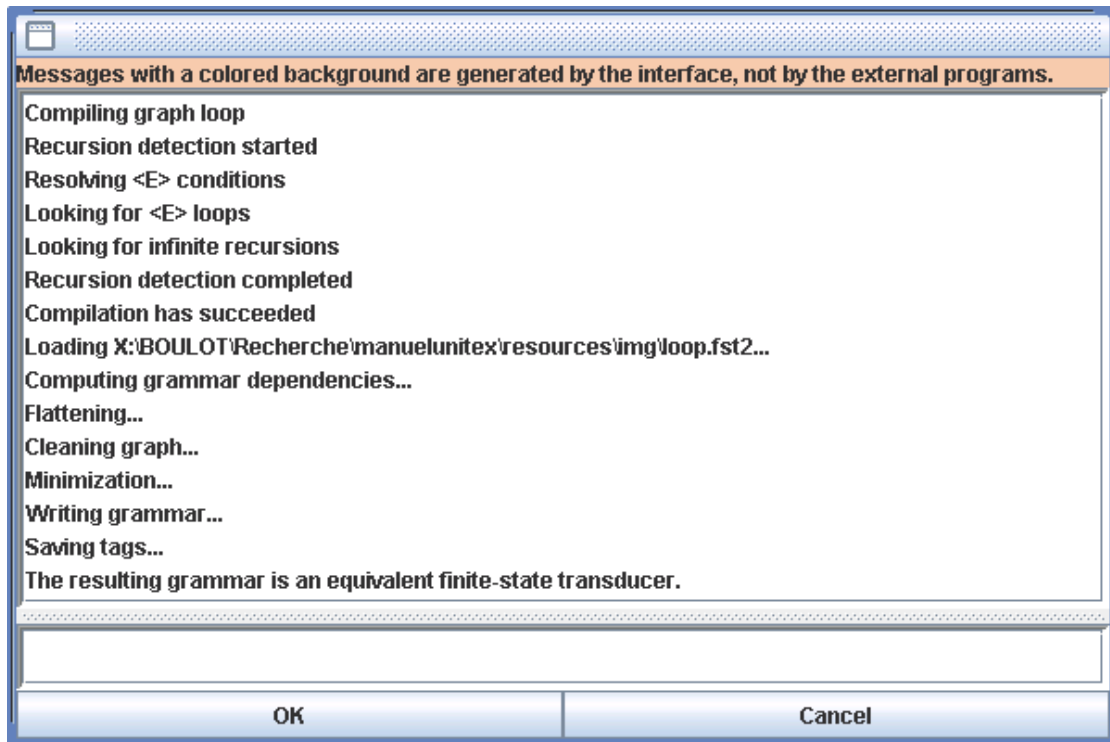


Figure 6.6: Resultat of the approximation of a grammar

It is not possible to associate a transducer output with a call to a subgraph. Such outputs are ignored by Unitex. It is therefore necessary to use an empty box that is situated to the left of the call to the subgraph in order to specify the output (cf. Figure 6.7).

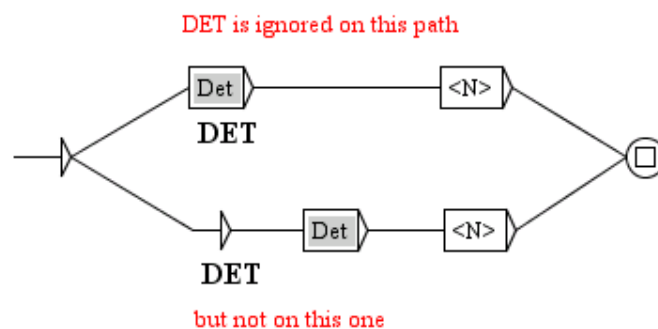


Figure 6.7: How to associate an output with a call to a subgraph

The grammars must not contain void loops because the Unitex programs cannot terminate the exploration of such a grammar. A void loop is a configuration that causes the `Locate`

program to enter an infinite loop. Void loops can originate from transitions that are labeled by the empty word or from recursive calls to subgraphs.

Void loops due to transitions with the empty word can have two origins of which the first is illustrated by the Figure 6.8. This type of loops is due to the fact that a transition with the empty word cannot be eliminated automatically by Unitex because it is associated with an output. Thus, the transition with the empty word of Figure 6.8 will not be suppressed and will cause a void loop.

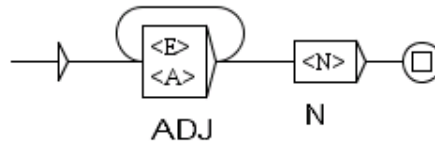


Figure 6.8: Void loop due to a transition by the empty word with a transduction

The second category of loop by epsilon concerns the call to subgraphs that can recognize the empty word. This case is illustrated in Figure 6.9: if the subgraph `Adj` recognizes epsilon, there is a void loop that Unitex cannot detect.

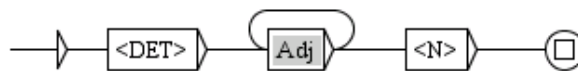


Figure 6.9: Void loop due to a call to a subgraph that recognizes epsilon

The third possibility of void loops is related to recursive calls to subgraphs. Look at the graphs `Det` and `DetCompose` in figure 6.10. Each of these graphs can call the other *without reading any text*. The fact that none of these two graphs has labels between the initial state and the call to the subgraph is crucial. In fact, if there were at least one label different from epsilon between the beginning of the graph `Det` and the call to `DetCompose`, this would mean that the Unitex programs exploring the graph `Det` would have to read the pattern described by that label in the text before calling `DetCompose` recursively. In this case the programs would loop infinitely only if they recognized the pattern an infinite number of times in the text, which is impossible.

6.2.4 Error detection

In order to keep the programs from blocking or crashing, Unitex automatically detects errors during graph compilation. The graph compiler checks that the main graph does not recognize the empty word and searches for all possible forms of void loops. When an error

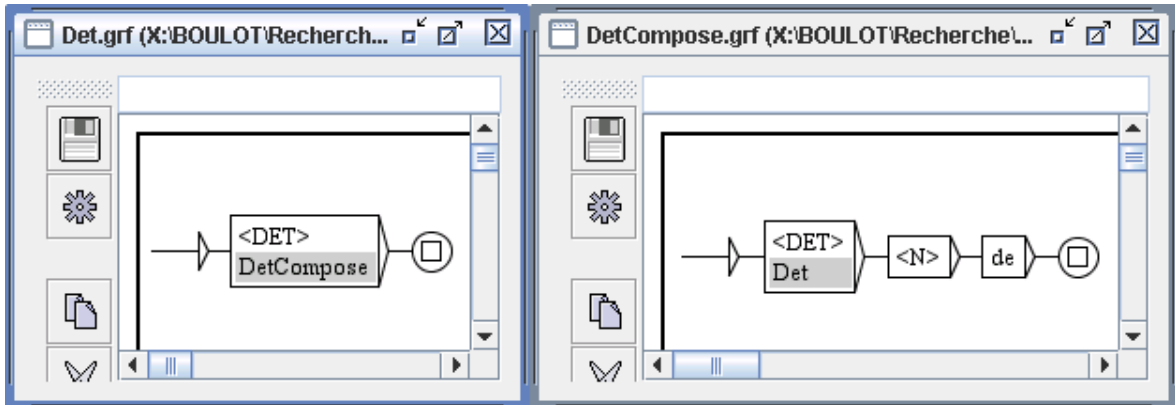
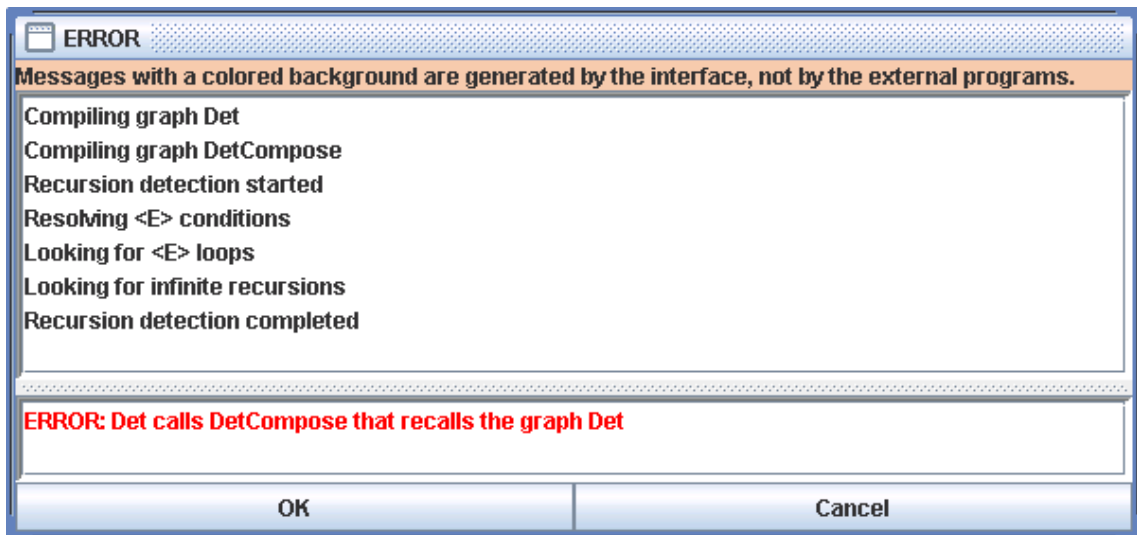


Figure 6.10: Void loop caused by two graphs calling each other

Figure 6.11: Error message when trying to compile `Det`

is encountered, an error message is displayed in the compilation window. Figure 6.11 shows the message that appears if one tries to compile the graph `Det` of Figure 6.10.

When you start a pattern search with a `.grf` graph, if Unitex detects an error at the graph compilation, the locate operation is automatically interrupted.

6.3 Contexts

Unitex graphs as we described them up to there are equivalent to algebraic grammars. These are also known as context-free grammars, because if you want to match a sequence A , the context of A is irrelevant. Thus, you cannot use a context-free graph for matching occurrences

of president not followed by of the republic.

However, you can draw graphs with positive or negative contexts. In that case, graphs are no more equivalent to algebraic grammars, but to context-sensitive grammars that do not have the same theoretical properties.

6.3.1 Right contexts

To define a right context, you must bound a zone of the graph with boxes containing $\$ [$ and $\$]$, which indicate the start and the end of the right context. These bounds appear in the graph as green square brackets. Both bounds of a right context must be located in the same graph.

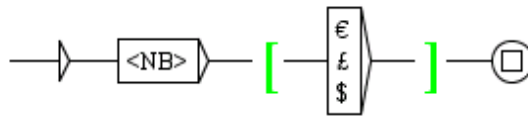


Figure 6.12: Using a right context

Figure 6.12 shows a simple right context. The graph matches numbers followed by a currency symbol, but this symbol will not appear in matched sequences, *i.e.* in the concordance.

Right contexts are interpreted as follows. During the application of a grammar on a text, let us assume that a right context start is found. Let *pos* be the current position in the text at this time. Now, the `Locate` program tries to match the expression described inside the right context. If it fails, then there will be no match. If it matches the whole right context (that is to say if `Locate` reaches the right context end), then the program will rewind at the position *pos* and go on exploring the grammar after the right context end.

You can also define negative right contexts, using $\$! [$ to indicate the right context start. Figure 6.13 shows a graph that matches numbers that are not followed by `th`. The difference with positive right contexts is that when `Locate` tries to match the expression described inside the context, reaching the context stop will be considered as a failure, because it would have matched a forbidden sequence. At the opposite, if the context stop cannot be reached, then `Locate` will rewind at the position *pos* and go on exploring the grammar after the context end.

Right contexts can appear anywhere in the graph, including the beginning of the graph. Figure 6.14 shows a graph that matches an adjective in the right context of something that is not a past participle. In other words, this graph matches adjectives that are not ambiguous with past participles.

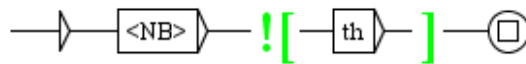


Figure 6.13: Using a negative right context



Figure 6.14: Matching an adjective that is not ambiguous with a past participle

This mechanism allows you to formulate complex patterns. For instance, the graph of figure 6.15 matches a sequence of two simple nouns that is not ambiguous with a compound word. In fact, the pattern `<CDIC><<^[^]+ [^]+>>` matches a compound word with exactly one space, and the pattern `<N><<^[^]+>>` matches a noun without space, that is to say a simple noun. Thus, in the sentence *Black cats should like the town hall*, this graph will match *Black cats*, but not *town hall*, which is a compound word.

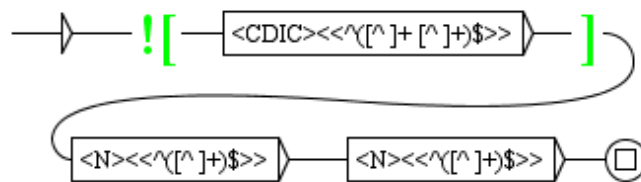


Figure 6.15: Advanced use of right contexts

You can use nested contexts. For instance, the graph shown in figure 6.16 matches a number that is not followed by a dot, except for a dot followed by a number. Thus, in the sequence *5.0+7.=12*, this graph will match *5*, *0* and *12*.

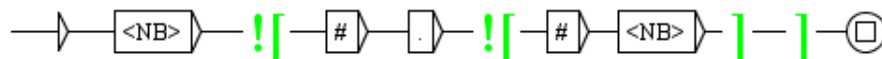


Figure 6.16: Nested contexts

If a right context contains boxes with transducer outputs, the outputs are ignored. However, it is possible to use a variable that was defined inside a right context (cf. figure 6.17). If you apply this graph in MERGE mode to the text *the cat is white*, you will obtain:

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

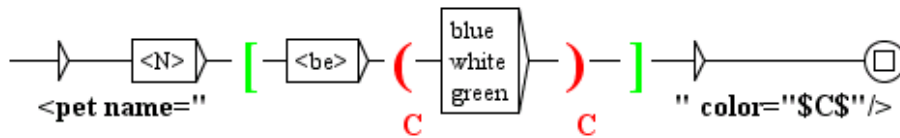


Figure 6.17: Variable defined inside a right context

6.3.2 Left contexts

It is also possible to look for an expression X only if it occurs after an expression Y . Of course, it was already possible to do that with a grammar like the one shown on Figure 6.18. However, with such a grammar, the context part on the left will be included in the match, as shown on Figure 6.19.

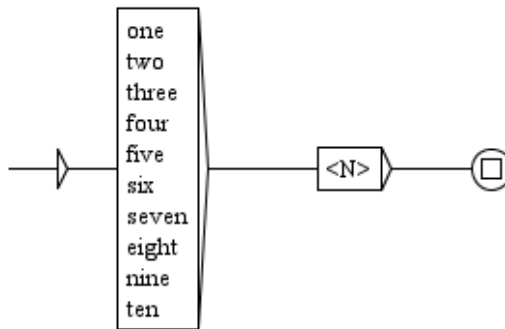


Figure 6.18: Matching a noun that occurs after a numerical determiner

To avoid that, you can use the special symbol $\$*$ to indicate the end of the left context of the expression you want to match. This symbol will be represented by a green star in the graph, as shown on Figure 6.20. The effect of such a context is to use this part of the grammar for computing matches, but to ignore it in the results, as shown on Figure 6.21.

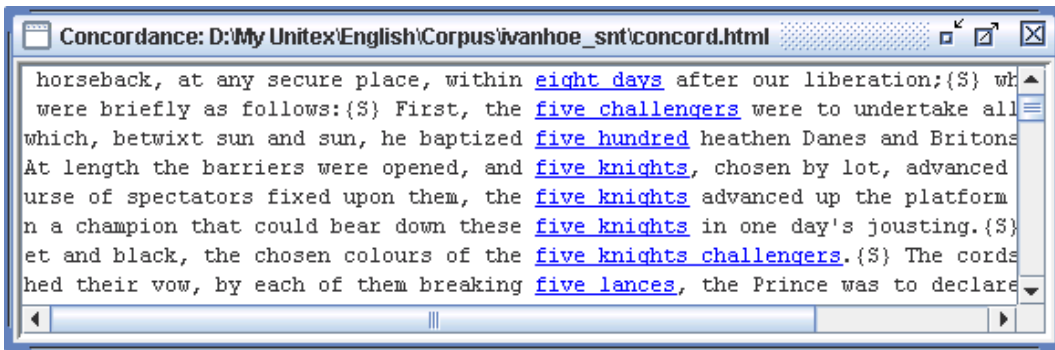


Figure 6.19: Results of the application of the grammar shown on Figure 6.18

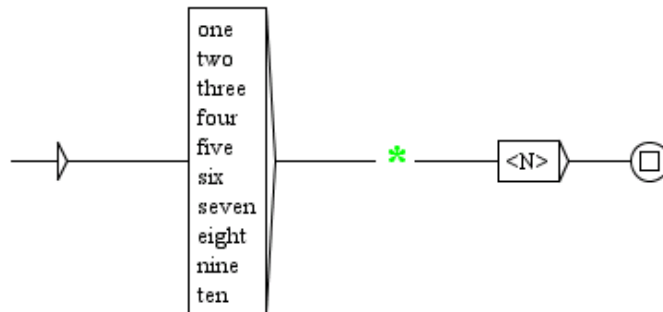


Figure 6.20: Matching a noun after a left context

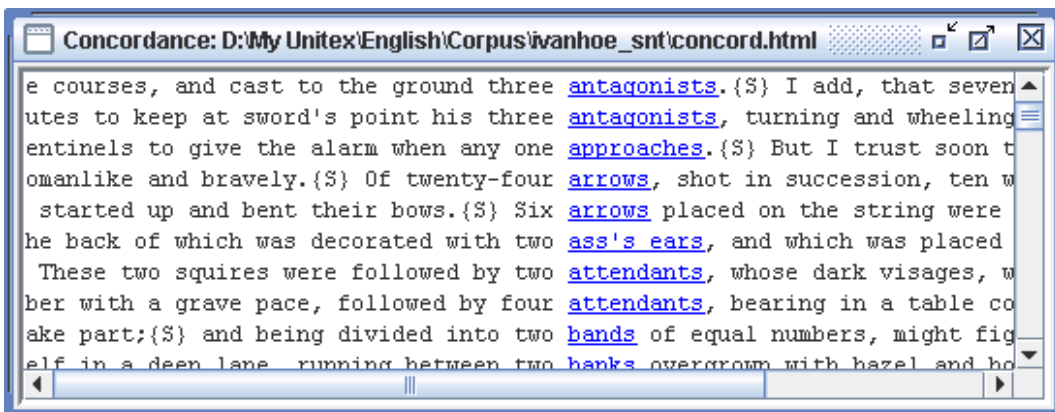


Figure 6.21: Results of the application of the grammar shown on Figure 6.20

All the outputs produced in the left context are ignored, as you can see in the concordance of Figure 6.23, showing the results obtained with the grammar of Figure 6.22.

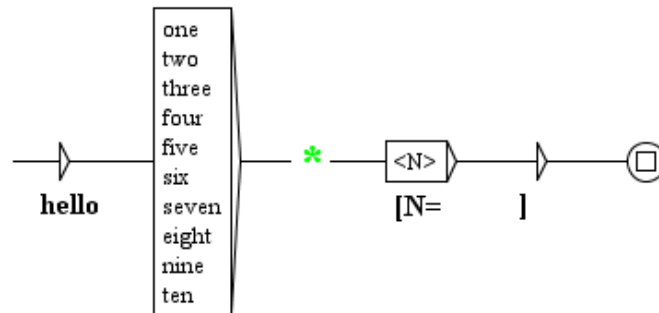


Figure 6.22: Ignored output in a left context

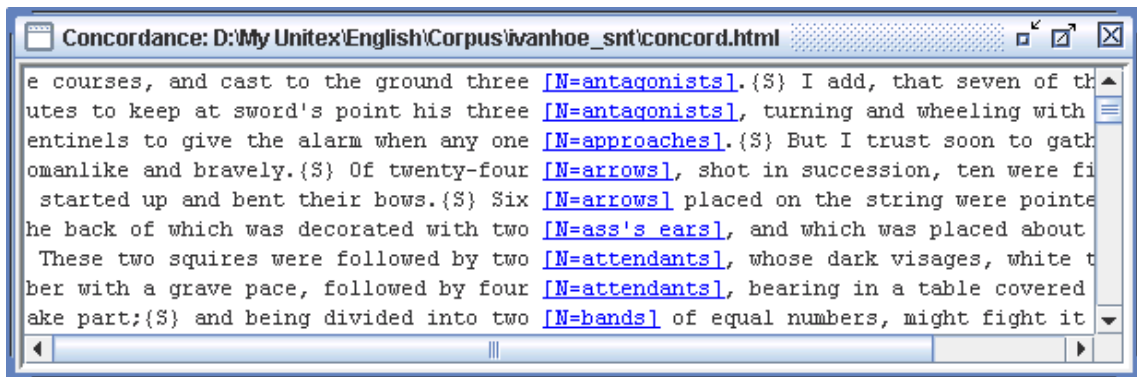


Figure 6.23: Results of the application of the grammar shown on Figure 6.22

However, you can catch things with variables (see section 6.7.5) and use them outside the left context, as shown on grammar of Figure 6.24.

So, with left and right contexts, you can make a distinction between the pattern used to match something, and the thing you want to extract in your results. For instance, the grammar shown on Figure 6.26 looks for expressions like *the animal's*, but only extract nouns, as you can see on Figure 6.27.

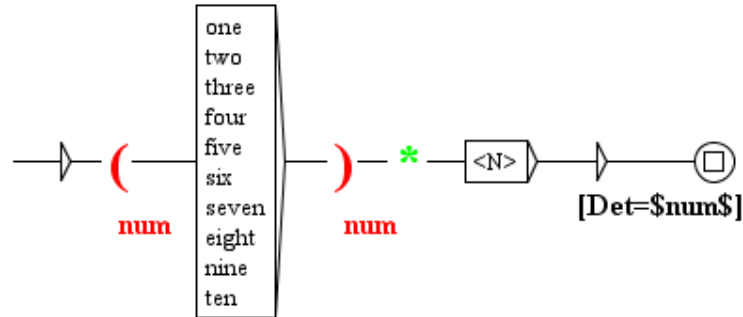


Figure 6.24: Using a variable in a left context

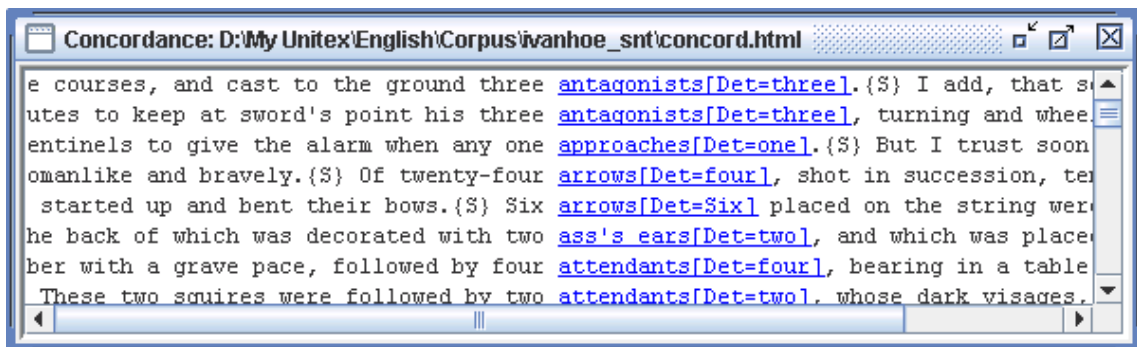


Figure 6.25: Results of the application of the grammar shown on Figure 6.24

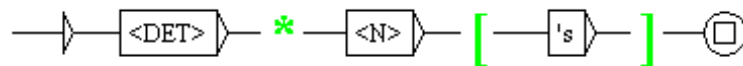


Figure 6.26: A grammar with both left and right contexts

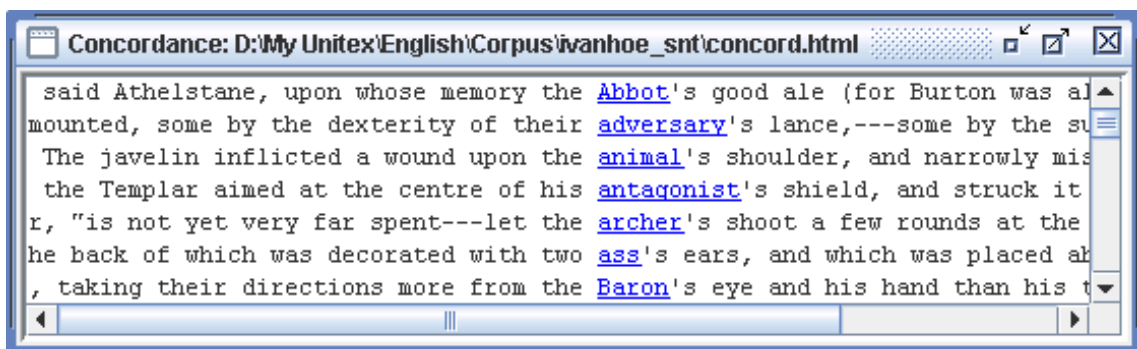
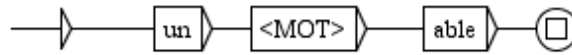


Figure 6.27: Results of the application of the grammar shown on Figure 6.26

6.4 The morphological mode

6.4.1 Why ?

As Unitex works on a tokenized version of the text, it is not possible to perform queries that need to enter inside tokens, except with morphological filters (see section 4.7), as shown on Figure 6.28.



This does not work. We should use the following morphological filter:
`<<^un.*able$>>`

Figure 6.28: Matching morphological things

However, even morphological filters cannot allow any query, since they cannot refer to dictionaries. Thus, it is impossible to formulate this way a query like “a word made of the prefix *un* followed by an adjective suffixed with *able*”.

To overcome this difficulty, we introduced a morphological mode in the `Locate` program. It consists of bounding a part of your grammar with the special symbols `$<` and `$>`. Within this zone, things are matched letter by letter, as shown on Figure 6.29.

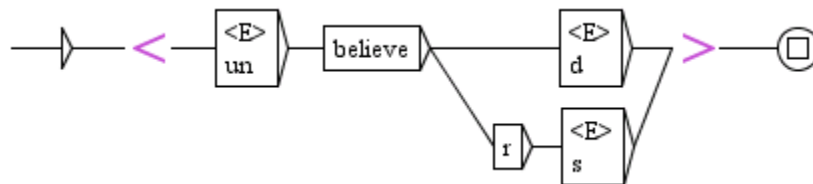


Figure 6.29: Example of morphological zone in a grammar

6.4.2 The rules

In this mode, the content of the graph is not interpreted as it is in the normal way.

1. There is no implicit space between boxes. So, if you want to match a space, you have to make it explicit with " " (a space between double quotes).
2. You can still use subgraphs, but the end of the morphological zone must occur in the same graph as its beginning.
3. You can use morphological filters on `<DIC>` and patterns referring to dictionaries, like `<be>`, `<N:ms>`, etc.

4. You can use morphological filters alone or on <TOKEN>, but note that your filters will only apply to the current character. As a consequence, filters like <<[1-9][0-9]>> that try to match more than one character will never match anything. In fact, in morphological mode, morphological filters should only be used to express negations like <<[^aeiouy]>> (any character that is not a vowel).
5. Left and right contexts are forbidden.
6. You can use outputs.
7. <MOT> will match any letter, as defined in the alphabet file.
8. <MIN> will match any lowercase letter, as defined in the alphabet file.
9. <MAJ> will match any uppercase letter, as defined in the alphabet file.
10. <DIC> will match any word present in the morphological dictionaries (see below).
11. You can use patterns that refer to the morphological dictionaries, like <have>, <V:K>, etc.
12. The meta #, <PRE>, <NB>, <SDIC> and <CDIC> are forbidden.
13. If you reach the end of the morphological zone and if you are not at the end of a token, the match will fail. For instance, if the text contains *enabled*, you can not only match *enable*.

6.4.3 Morphological dictionaries

In morphological mode, you can perform queries using dictionaries. For instance, you can ask for every word made of the prefix *un* followed by an adjective with the grammar shown on Figure 6.30.

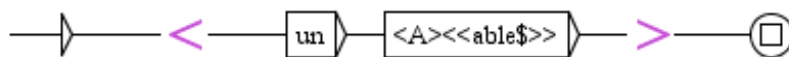


Figure 6.30: Matching words made of 'un'+adjective ending with 'able'

However, if we want to match with this grammar the word *unaware*, we must know that *aware* is an adjective. But, *aware* may not be present in the text, so that we cannot rely on the text dictionaries. This is the reason why we must define a list of dictionaries to lookup in in morphological mode. To do that, go in "Info>Preferences>Morphological dictionaries", as shown on Figure 6.31. You can select as many dictionaries as you want, but they MUST be *.bin* ones. Once done, you can apply your grammar and get results.

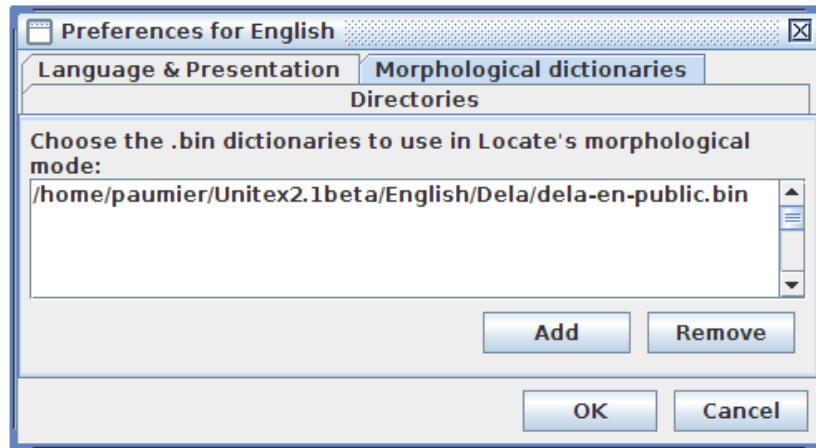


Figure 6.31: Configuration of morphological dictionaries

6.4.4 Dictionary entry variables

You can associate variables to patterns that refer to the morphological dictionaries, except `<DIC>`. To do that, you must set the output of the box with `xxx` where `xxx` is a valid variable name. That defines a special variable named `xxx` that represents the dictionary entry that has matched with your pattern. Now you can get the inflected form, lemma and codes of the entry with `$xxx.INFLECTED$`, `$xxx.LEMMA$` and `$xxx.CODE$`, as shown on Figure 6.32. You can also use the following patterns:

- `$xxx.CODE.GRAM$`: provides only the first grammatical code, supposed to be the POS category
- `$xxx.CODE.SEM$`: provides all remaining grammatical codes, if any, separated with +
- `$xxx.CODE.FLEX$`: provides all inflectional codes, if any, separated with :

Moreover, such variables can be used even after the end of the morphological mode, as shown on Figure 6.34. They can also be tested as explained in section 6.7.5.



Figure 6.32: Using a morphological variable

Dictionary variables in LocateTfst

In grammars to be applied with `LocateTfst`, you have an extra feature. Even if you are not in morphological mode, you can capture the information associated to a tag contained

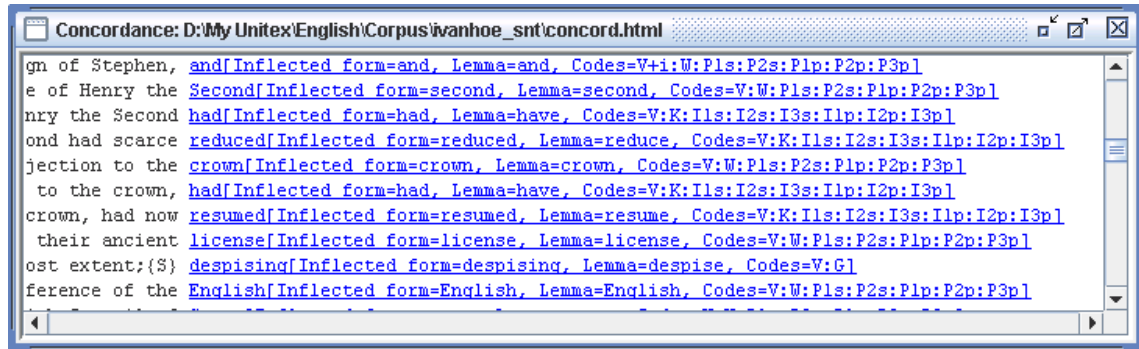


Figure 6.33: Results of grammar of Figure 6.32 applied in MERGE mode



Figure 6.34: Using a morphological variable in normal mode

in a box of the text automaton in a dictionary variable. To do that, you have to associate to your box an output of the form $\$: abc\$$ where abc will be the name of your variable. You can then use it as dictionary variable capture in morphological mode.

6.5 Exploring grammar paths

It is possible to generate the paths recognized by a grammar, if they are in finite number, for example to check that it correctly generates the expected forms. For that, open the main graph of your grammar, and ensure that the graph window is the active window (the active window has a blue title bar, while the inactive windows have a gray title bar). Now go to the "FSGraph" menu and then to the "Tools" menu, and click on "Explore Graph paths". The Window of figure 6.35 appears.

The upper box contains the name of the main graph of the grammar to be explored. The following options are connected to the outputs of the grammar and to subgraph calls:

- "Ignore outputs": outputs are ignored;
- "Separate inputs and outputs": outputs are displayed after inputs ($a\ b\ c\ /\ A\ B\ C$);
- "Merge inputs and outputs": each output is emitted immediately after the input to which it corresponds ($a/A\ b/B\ c/C$).
- "Only paths": calls to subgraphs are explored recursively;

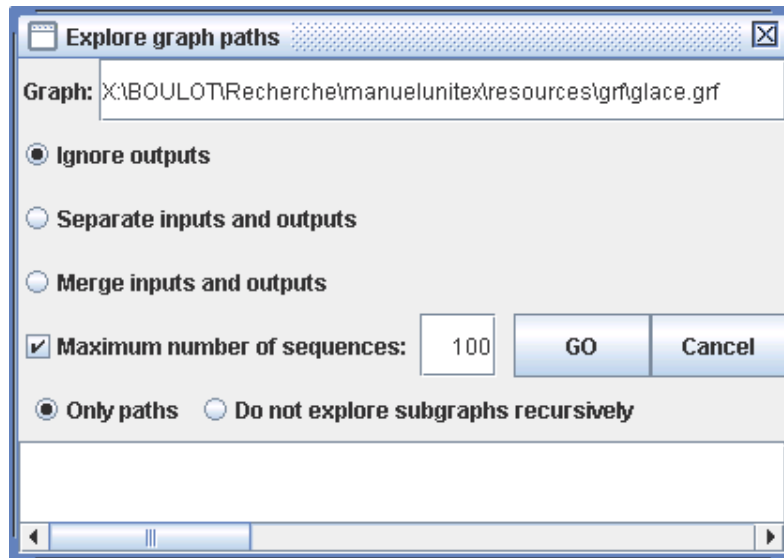


Figure 6.35: Exploring the paths of a grammar

- "Do not explore subgraphs recursively": calls to subgraphs are printed but not explored recursively.

If the option "Maximum number of sequences" is activated, the specified number will be the maximum number of generated paths. If the option is not selected, all paths will be generated, if they are in finite number.

Here you see what is created for the graph shown on Figure 6.36 with default settings (ignoring outputs, limit = 100 paths):

```

<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

```

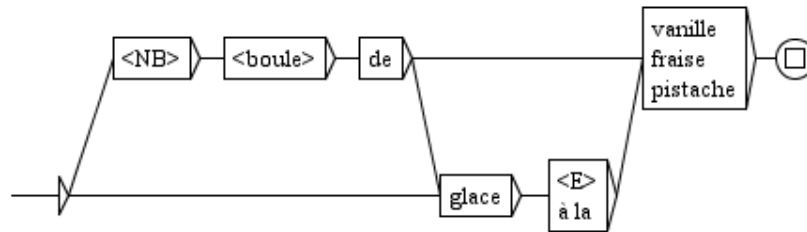



Figure 6.36: Sample graph

6.6 Graph collections

It can happen that one wants to apply several grammars located in the same directory. For that, it is possible to automatically build a grammar starting from a file tree structure. Let us suppose for example that one has the following tree structure:

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

If one wants to gather all these grammars in only one, one can do it with the "Build Graph Collection" command in the "FSGraph Tools" sub-menu. One configures this operation by means of the window seen in figure 6.37.

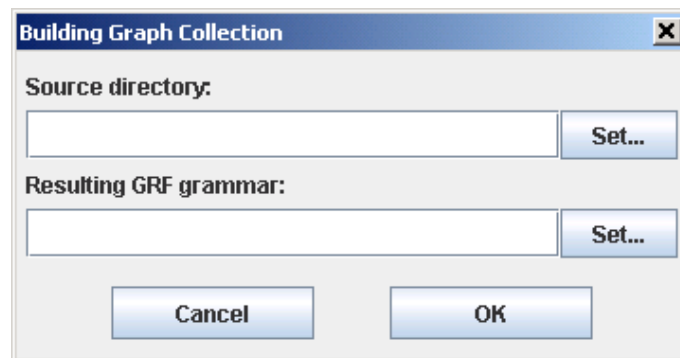


Figure 6.37: Building a graph collection

In the "Source Directory" field, select the root directory which you want to explore (in our example, the directory *Dicos*). In the field "Resulting GRF grammar", enter the name of the produced grammar.

WARNING: Do not place the output grammar in the tree structure which you want to explore, because in this case the program will try to read and to write simultaneously in this file, which will cause a crash.

When you click on "OK", the program will copy the graphs to the directory of the output grammar, and will create subgraphs corresponding to the various sub-directories, as one can see in figure 6.38, which shows the output graph generated for our example.

One can observe that one box contains the calls with subgraphs corresponding to sub-directories (here directories *Banque* and *Nourriture*), and that the other box calls all the graphs which were in the directory (here the graph `truc.grf`).

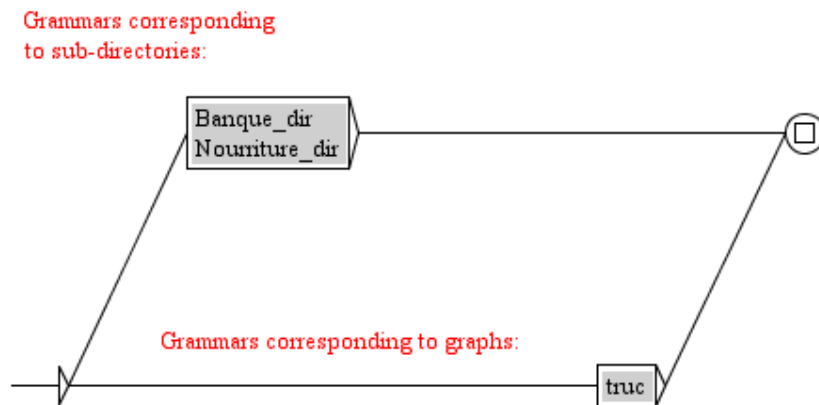


Figure 6.38: Main graph of a graph collection

6.7 Rules for applying transducers

This section describes the rules for the application of transducers along with the operations of preprocessing and the search for patterns. The following does not apply to inflection graphs and normalization graphs for ambiguous forms.

6.7.1 Insertion to the left of the matched pattern

When a transducer is applied in REPLACE mode, the output replaces the sequences that have been read in the text. When a box in a transducer has no output, it is processed as if it had an `<E>` output. In MERGE mode, the output is inserted to the left of the recognized sequences.

Look at the transducer in Figure 6.39. If this transducer is applied to the novel *Ivanhoe* by Sir Walter Scott in MERGE mode, the following concordance is obtained.



Figure 6.39: Example of a transducer

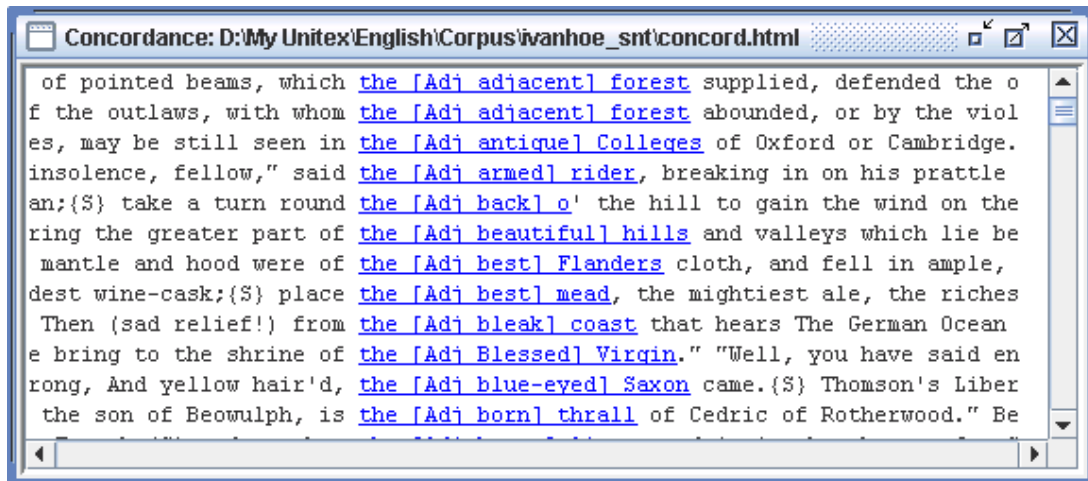


Figure 6.40: Concordance obtained in MERGE mode with the transducer of figure 6.39

6.7.2 Application while advancing through the text

During the preprocessing operations, the text is modified as it is being read. In order to avoid the risk of infinite loops, it is necessary that the sequences that are produced by a transducer will not be re-analyzed by the same one. Therefore, whenever a sequence is inserted into the text, the application of the transducer is continued after that sequence. This rule only applies to preprocessing transducers, because during the application of syntactic graphs, the transductions do not modify the processed text but a concordance file which is distinct from the text.

6.7.3 Priority of the leftmost match

During the application of a local grammar, overlapping occurrences are all indexed. Note that we talk about real overlapping occurrences like *abc* and *bcd*, not nested occurrences like *abc* and *bc*. During the construction of the concordance all these overlapping occurrences are presented (cf. Figure 6.41).

On the other hand, if you modify a text instead of constructing a concordance, it is necessary to choose among these occurrences the one that will be taken into account. Unitex applies the following priority rule for that purpose: the leftmost sequence is used.

If this rule is applied to the three occurrences of the preceding concordance, the occurrence

iver Don, there extended `[in ancient]` times a large forest, covering the gr
 r Don, there extended in `[ancient times]` a large forest, covering the great
 here extended in ancient `[times a]` large forest, covering the greater part

Figure 6.41: Overlapping occurrences in concordance

`[in ancient]` overlaps with `[ancient times]`. The first is retained because this is the leftmost occurrence and `[ancient times]` is eliminated. The following occurrence of `[times a]` is no longer in conflict with `[ancient times]` and can therefore appear in the result:

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

The rule of priority of the leftmost match is applied only when the text is modified, be it during preprocessing or after the application of a syntactic graph (cf. section 6.10.4).

6.7.4 Priority of the longest match

During the application of a syntactic graph it is possible to choose if the priority should be given to the shortest or the longest sequences or if all sequences should be retained. During preprocessing, the priority is always given to the longest sequences.

6.7.5 Transducer outputs with variables

As we have seen in section 5.2.5, it is possible to use variables to store some text that has been analyzed by a grammar. These variables can be used in preprocessing graphs and in syntactic graphs.

You have to give names to the variables you use. These names can contain non-accentuated lower-case and upper-case letters between A and Z, digits and the character `_` (underscore).

In order to define the boundaries of the zone to be stored in a variable, you have to create two boxes that contain the name of the variable enclosed in the characters `$` and `(` (`$` and `)` for the end of a variable). In order to use a variable in a transducer output, its name must be surrounded by the character `$` (cf. Figure 6.42).

Variables are global. This means that you can define a variable in a graph and reference it in another as is illustrated in the graphs of Figure 6.42.

If the graph `TitleName` is applied in MERGE mode to the text *Ivanhoe*, the concordance in Figure 6.43 is obtained.

Outputs with variables can be used to move word groups. In fact, the application of a transducer in REPLACE mode inserts only the produced sequences into the text. In order to invert two word groups, you just have to store them into variables and produce an output with these variables in the desired order. Thus, the application of the transducer in Figure 6.44 in REPLACE mode to the text *Ivanhoe* results in the concordance of Figure 6.45.

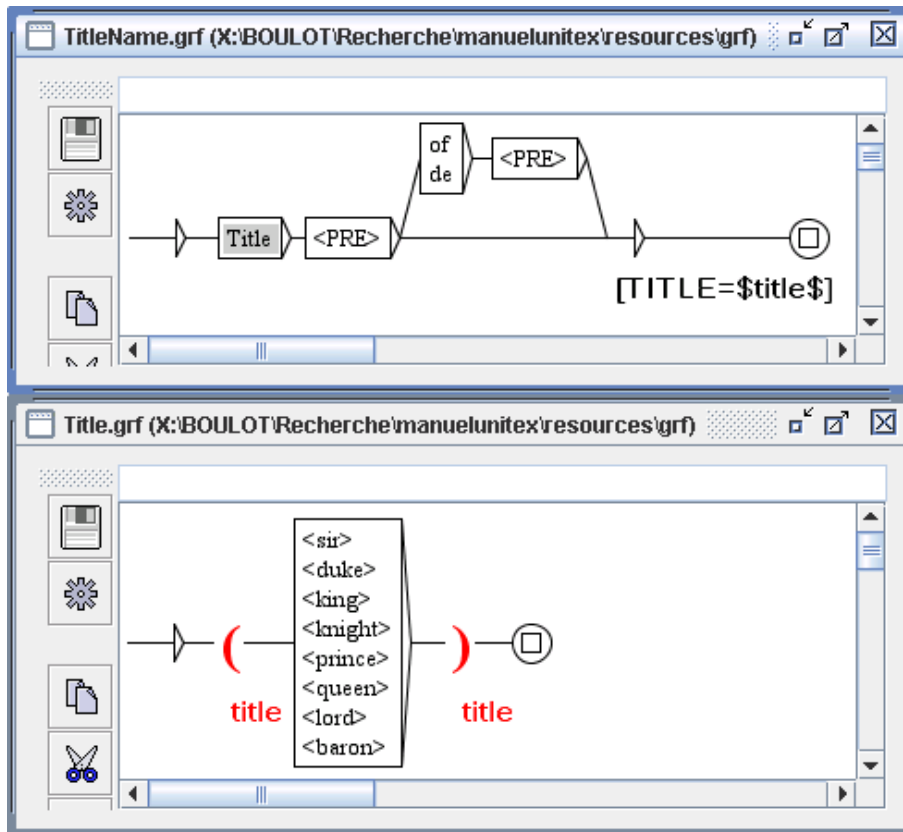


Figure 6.42: Definition of a variable in a subgraph

The window shows a text document with several lines of text, where titles are highlighted in blue and followed by a tag like [TITLE=Prince] or [TITLE=Sir].

```
lders and was silent. (S)Prince John[TITLE=Prince] resumed his retreat
he hermit---"his name is Sir Anthony of Scrabelstone[TITLE=Sir]---as if I
again passed round, "To Sir Athelstane of Coningsburgh[TITLE=Sir].
r shall call thee Saxon, Sir Baron[TITLE=Sir]," replied Cedric, offended
to say, lady," answered Sir Brian de Bois[TITLE=Sir]-Guilbert,
ory." "Sir Palmer," said Sir Brian de Bois[TITLE=Sir]-Guilbert
so unsafe, the escort of Sir Brian de Bois[TITLE=Sir]-Guilbert is not to
er to be a handmaiden to Sir Brian de Bois[TITLE=Sir]-Guilbert, after the
ghts of the Temple---and Sir Brian de BoisGuilbert[TITLE=Sir] well knows
have offended," replied Sir Brian[TITLE=Sir], "I crave your
```

Figure 6.43: Concordance obtained by application of graph TitleName

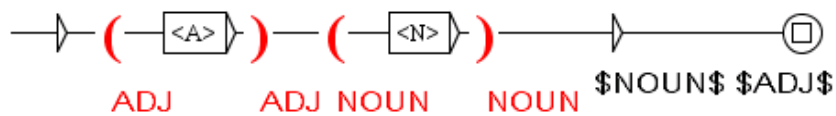


Figure 6.44: Inversion of words using two variables

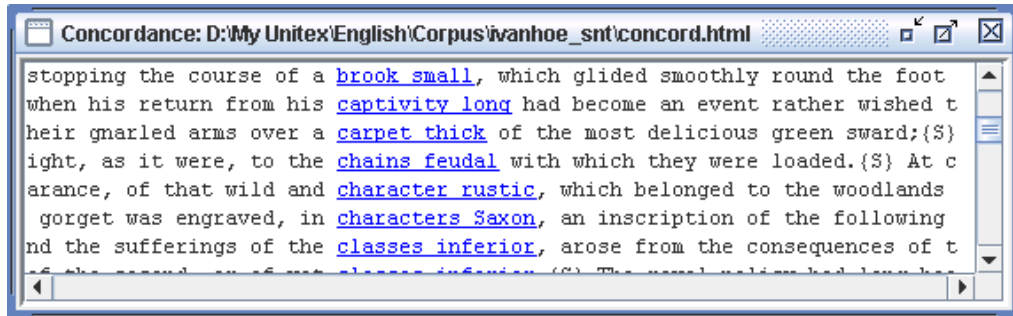


Figure 6.45: Result of the application of the transducer in figure 6.44

If the beginning or the end of variable is malformed (end of a variable before its beginning or absence of the beginning or end of a variable), by default, it will be ignored during the emission of outputs. See section 6.10.2 for other variable error policies.

There is no limit to the number of possible variables.

The variables can be nested and even overlap as is shown in figure 6.46.

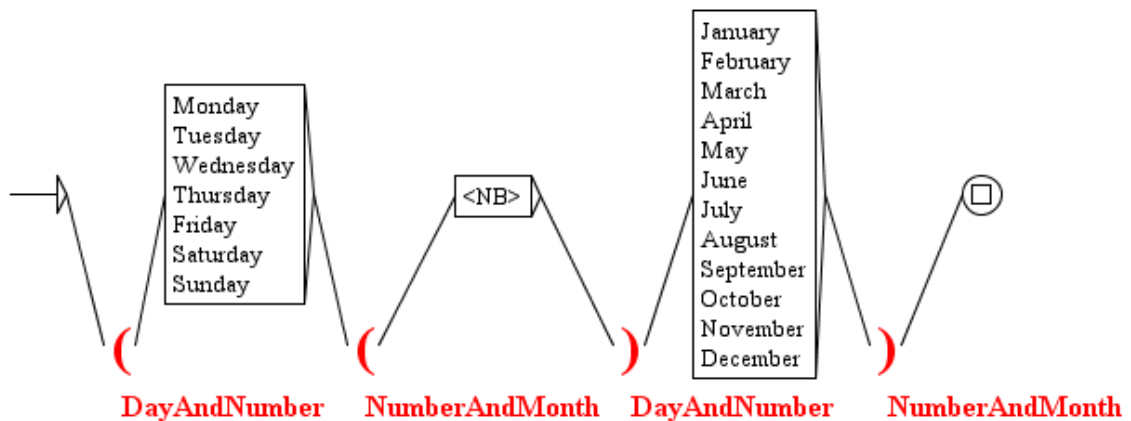


Figure 6.46: Overlapping variables

6.8 Output variables

Normal variables declared with `$xxx` (and `$xxx`) capture portions of the input text. It is also possible to capture portions of the outputs produced by your grammar with output variables. Such variables are declared with `$|xxx` (and `$|xxx`) . Those tags appear in blue as shown on Figure 6.47. Note that when an output variable is being declared, the outputs are not emitted in the output occurrence; they are just stored into the pending output variable(s). Note that outputs are processed first, so that if an output string contains something like `$A.LEMMA$`, the output variable will not contain this raw string but rather the lemma associated to variable `A`. Moreover, output variables only capture explicit outputs produced by your grammar. Thus, even if you work in MERGE mode, output variables never capture the input text.

For instance, this example grammar applied to *Ivanhoe* will produce in MERGE mode the concordance shown on Figure 6.48. Thus you can see that the outputs `ADJ` and `NOUN` have not been inserted to the left of the input text, as one may have expected.

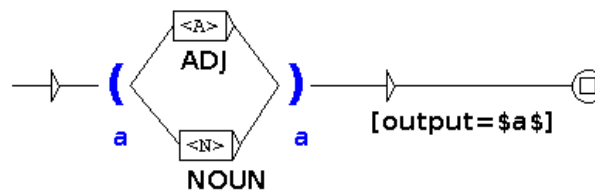


Figure 6.47: Output variables

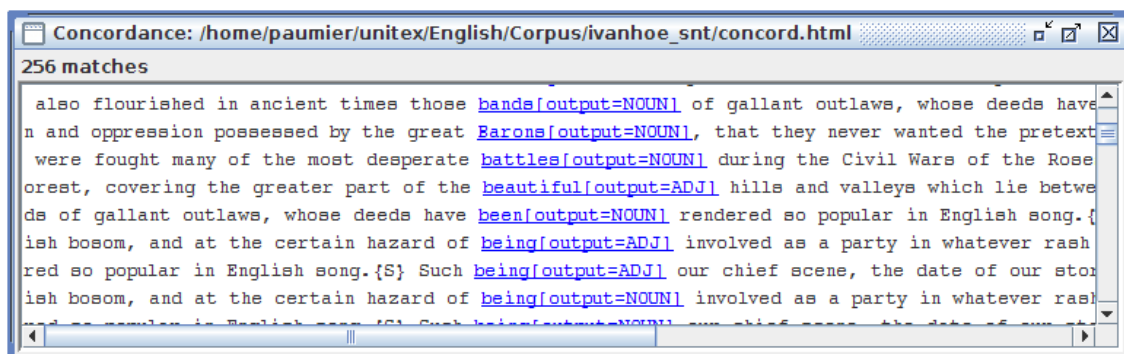


Figure 6.48: Concordance obtained with grammar of Figure 6.47

6.9 Operations on variables

6.9.1 Testing variables

It is possible to test whether a variable has been defined or not, in order to block the current matching operation if the condition is not verified. This is done by inserting the sequence `$xxx.SET$` in the output of a graph box. Then, if a variable named `xxx` has been defined, this sequence will be ignored in the output and the matching process will go on; otherwise, matching will be stopped and the program will backtrack. This operates on normal variables as well as on output ones and dictionary entry variables defined in morphological mode. You can check out if a variable has not been defined in the same way using `$xxx.UNSET$`. Figure 6.49 shows a graph that use a such a variable test. Figure 6.50 shows results obtained with this graph in MERGE mode.

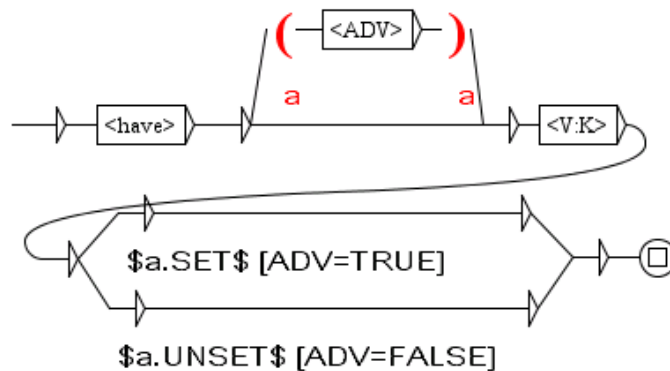


Figure 6.49: Testing a variable

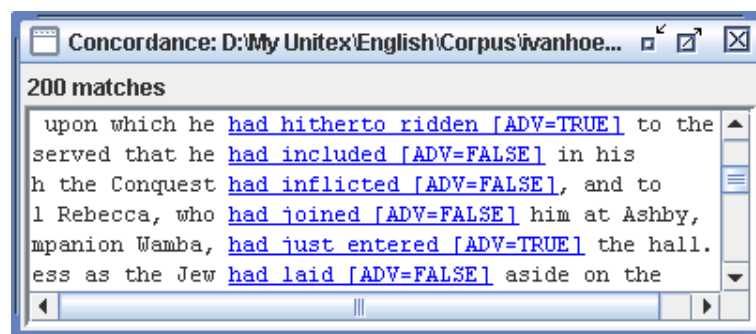


Figure 6.50: Results of a variable test

6.9.2 Comparing variables

Another kind of test you can perform consists of variable comparison. You can compare a variable (normal one, output one or dictionary one) against a constant string or another variable. To do that, you have to use the following syntax:

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

This acts like a switch that will block the grammar exploration if the value of variable `abc` is different from the value of variable `xyz`. Note that for dictionary variables, it is the inflected form as found in the dictionary (beware of case variations!) that is used in the test. If you want to compare variable `abc` against the constant string `JKL`, use the following test:

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

You can also test if contents differ with `UNEQUAL`.

6.10 Applying graphs to texts

This section only applies to syntactic graphs.

6.10.1 Configuration of the search

In order to apply a graph to a text, you open the text, then click on "Locate Pattern..." in the "Text" menu, or press `<Ctrl+L>`. You can then configure your search in the window shown in figure 6.51.

In the "Locate pattern in the form of" field, choose "Graph" and select your graph by clicking on the "Set" button. You can choose a graph in `.grf` format (Unicode Graphs) or a compiled graph in `.fst2` format (Unicode Compiled Graphs). If your graph is a `.grf` one, Unitex will compile it automatically before starting the search.

The "Index" field allows to select the recognition mode.

- "Shortest matches" : give precedence to the shortest matches;
- "Longest matches" : give precedence to the longest sequences. This is the default mode;
- "All matches" : give out all recognized sequences.

The "Search limitation" field allows you to limit the search to a certain number of occurrences. By default, the search is limited to the 200 first occurrences.

The "Grammar outputs" field concerns transducers. The "Merge with input text" mode allows you to insert the output sequences in input sequences. The "Replace recognized

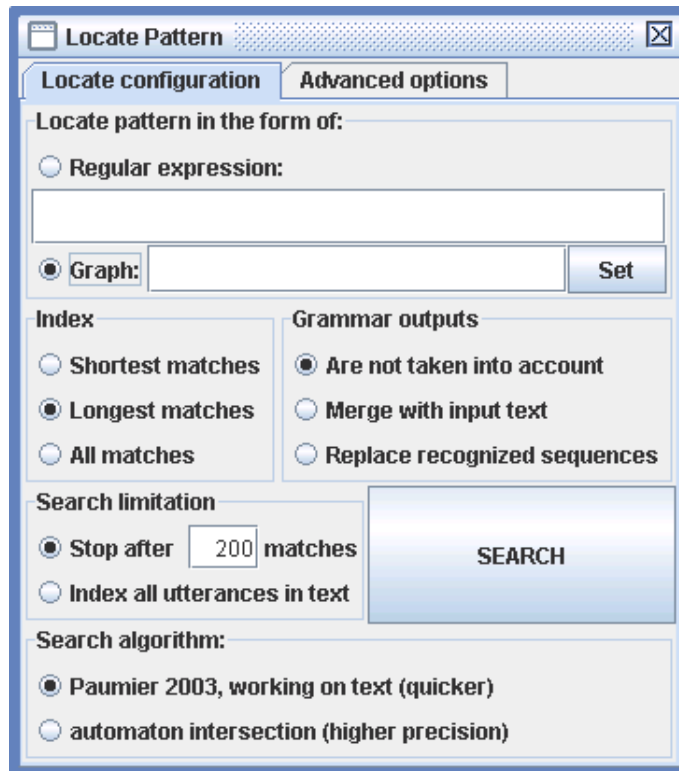


Figure 6.51: Locate pattern Window

sequences" mode allows you to replace the recognized sequences with the produced sequences. The third mode ignores all outputs. This latter mode is used by default.

After you have selected the parameters, click on "SEARCH" to start the search.

6.10.2 Advanced search options

If you select the "Advanced options" tab, you will see the frame shown on Figure 6.52.

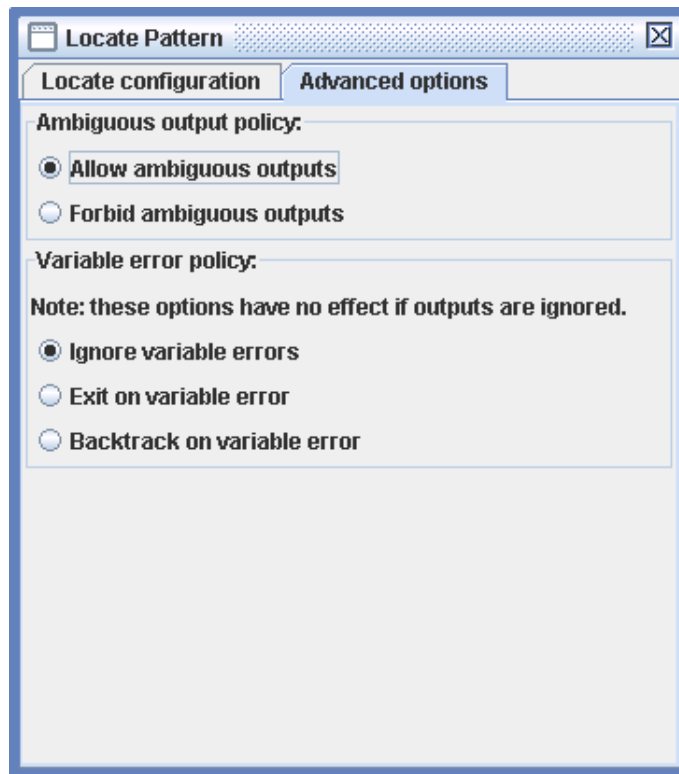


Figure 6.52: Advanced search options

The "Ambiguous output policy" option can be illustrated with the graph shown on Figure 6.53. When a determiner is followed by a word that can be either adjective or noun, it can produce two distinct outputs for the same text input sequence.

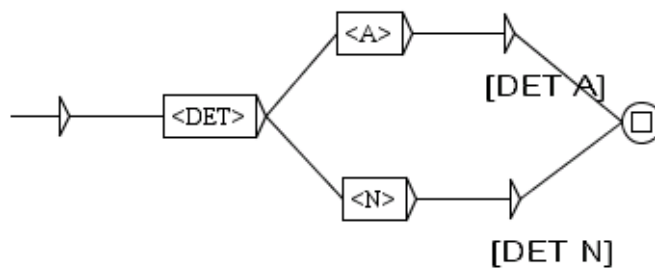


Figure 6.53: A graph with ambiguous outputs

If we apply this graph on *Ivanhoe* with the "Allow ambiguous outputs" option (the default one), we will obtain the text order concordance shown of Figure 6.54. As you can see, two outputs have been produced for the input sequence *the noble*.

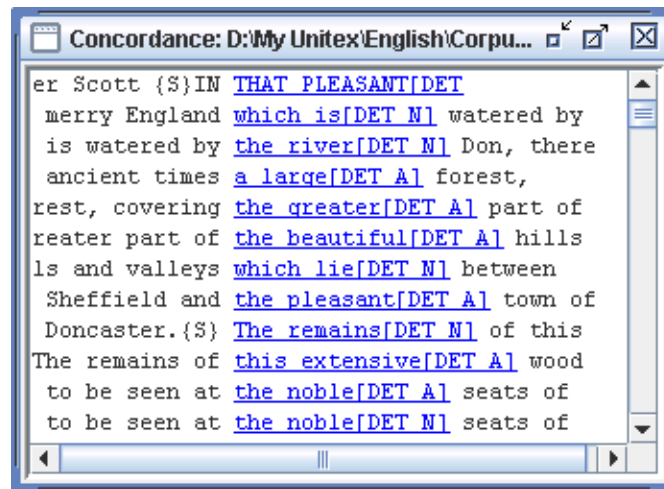


Figure 6.54: Ambiguous outputs for *the noble*

At the opposite, with the "Forbid ambiguous outputs" option, we will obtain the text order concordance shown of Figure 6.55, with only one arbitrarily chosen output for the input sequence *the noble*.

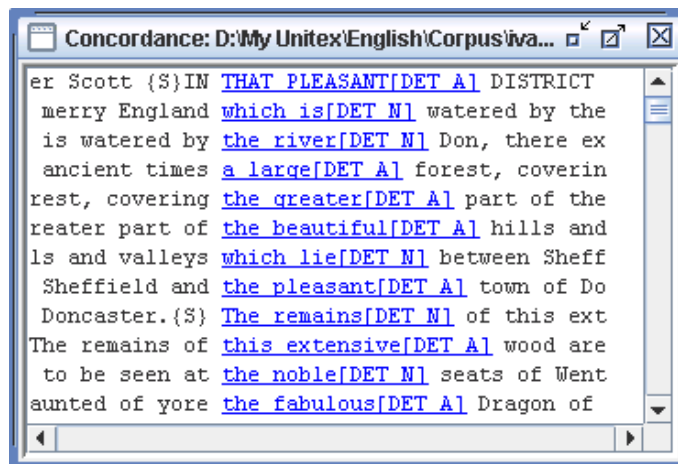


Figure 6.55: Single output for *the noble*

The "Variable error policy" option allows you to specify what `Locate/LocateTfst` is supposed to do when an output is found that contains a reference to a variable that has not been

correctly defined. Note that this parameter has no effect if outputs are to be ignored. For instance, let us consider the graph shown on Figure 6.56.

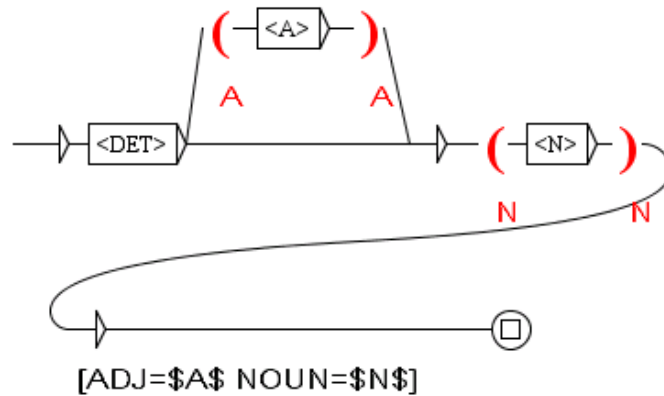


Figure 6.56: A variable *A* that may be undefined

With the "Ignore variable errors" option, *A* will just be ignored, as if it had an empty content, as shown on Figure 6.57.

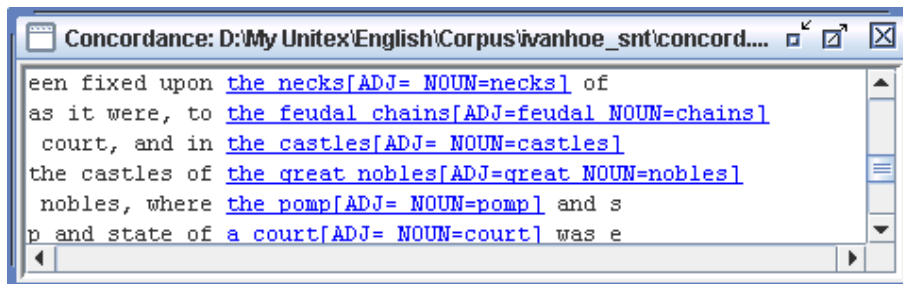


Figure 6.57: variable *A* that may be undefined

With the "Exit on variable error" option, *Locate/LocateTfst* will exit with an error message, as shown on Figure 6.58.

With the "Backtrack on variable error" option, *Locate/LocateTfst* will stop exploring the current path in the grammar. Thus, variables play the role of switches that cut paths when variables are undefined. For instance, the application of grammar 6.56 will only produce matches containing an adjective, as shown on Figure 6.59.

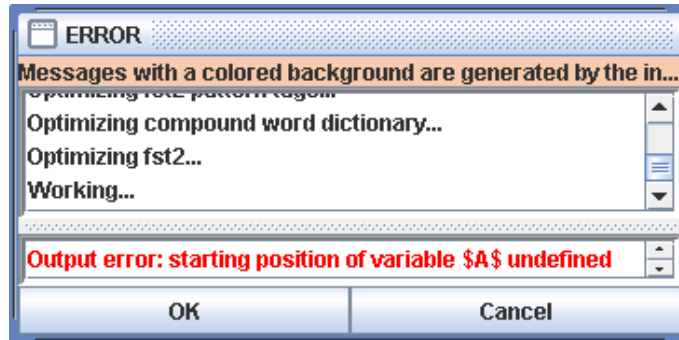


Figure 6.58: Exiting on variable error

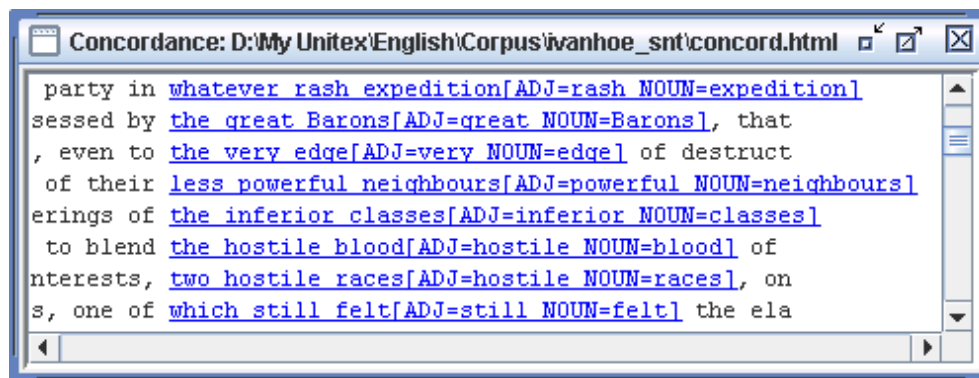


Figure 6.59: Backtracking on variable error

6.10.3 Concordance

The result of a search is an index file that contains the positions of all encountered occurrences. The window of Figure 6.60 lets you choose whether to construct a concordance or modify the text.

In order to display a concordance, you have to click on the "Build concordance" button. You can parameterize the size of left and right contexts in characters. You can also choose the sorting mode that will be applied to the lines of the concordance in the "Sort According to" menu. For further details on the parameters of concordance construction, refer to section 4.8.2.

The concordance is produced in the form of an HTML file. You can parameterize Unitex so that concordance files can be read using a web browser (cf. section 4.8.2).

If you display concordances with the window provided by Unitex, you can access a recognized sequence in the text by clicking on the occurrence. If the text window is not iconified and the text is not too long to be displayed, you see the selected sequence appear (cf. Figure 6.61).

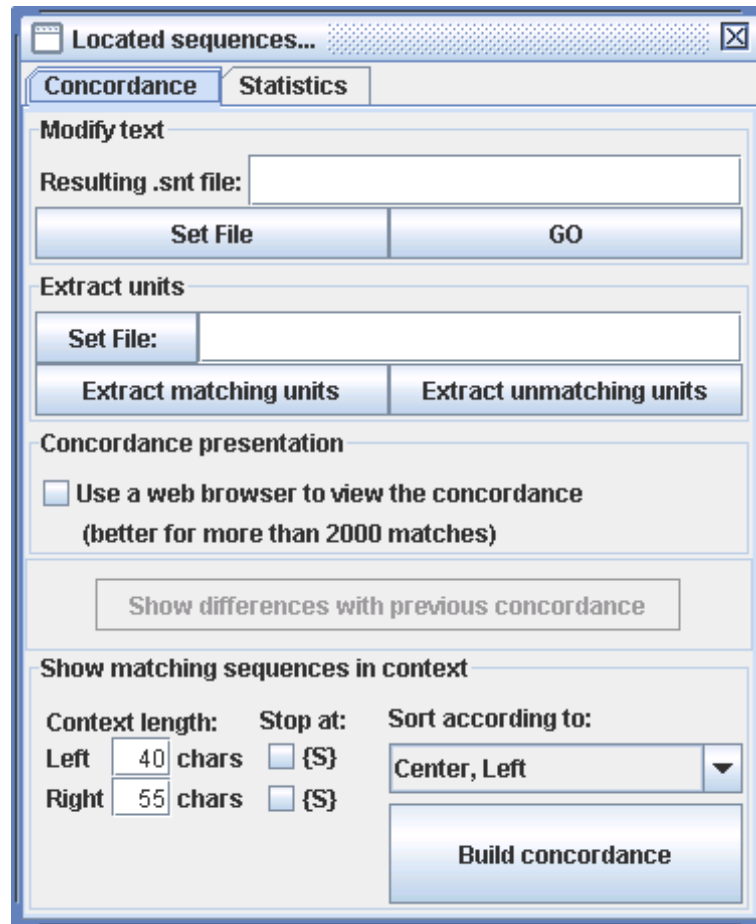


Figure 6.60: Configuration for displaying the encountered occurrences

Furthermore, if the text automaton has been constructed, and if the corresponding window is not iconified, clicking on an occurrence selects the automaton of the sentence that contains this occurrence.

6.10.4 Modification of the text

You can choose to modify the text instead of constructing a concordance. In order to do that, type a file name in the "Modify text" field in the window of Figure 6.60. This file has to have the extension `.txt`.

If you want to modify the current text, you have to choose the corresponding `.txt` file. If you choose another file name, the current text will not be affected. Click on the "GO" button to start the modification of the text. The precedence rules that are applied during these operations are described in section 6.7.

After this operation, the resulting file is a copy of the text in which transducer outputs have

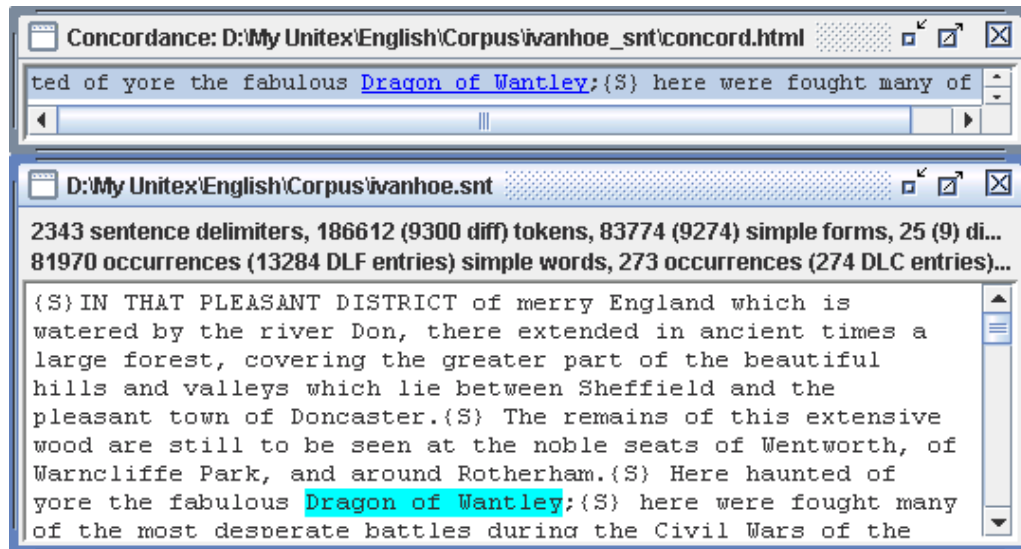


Figure 6.61: Selection of an occurrence in the text

been taken into account. Normalization operations and splitting into lexical units are automatically applied to this text file. The existing text dictionaries are not modified. Thus, if you have chosen to modify the current text, the modifications will be effective immediately. You can then start new searches on the text.

WARNING: if you have chosen to apply your graph ignoring the transducer outputs, all occurrences will be erased from the text.

6.10.5 Extracting occurrences

To extract from a text all sentences containing matches, set the name of your output text file using the "Set File" button in the "Extract units" frame (Figure 6.60). Then, click on "Extract matching units". At the opposite, if you click on "Extract unmatching units", all sentences that do not contain any match will be extracted.

6.10.6 Comparing concordances

With the "Show differences with previous concordance" option, you can compare the current concordance with the previous one. The `ConcorDiff` program builds both concordances according to text order and compares them line by line. The result is an HTML page that presents results in two columns. A blue line indicates that an utterance is common to the two concordances. A red line indicates that a match is common to both concordances but with different range, *i.e.* the two matches only overlap partially. A green line indicates an utterance that appears in only one concordance. Figure 6.62 gives an example.

NOTE: you cannot click on utterances in a concordance comparison.

If you have no previous concordance the button is deactivated.

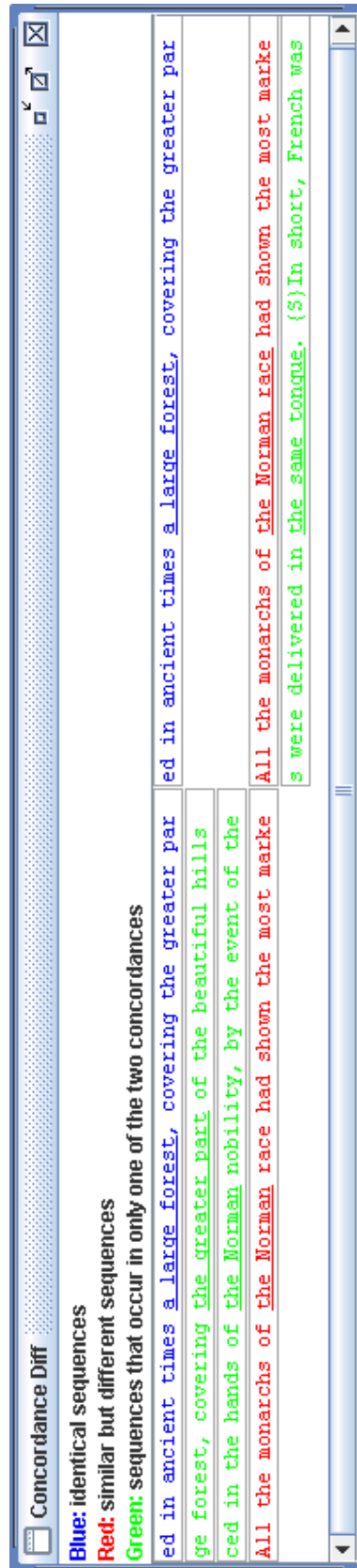


Figure 6.62: Example of a concordance comparison

Chapter 7

Text automaton

Natural languages contain much lexical ambiguity. The text automaton is an effective and visual way of representing such ambiguity. Each sentence of a text is represented by an automaton whose paths represent all possible interpretations.

This chapter presents the concept of text automaton, the details of their construction and the operations that can be applied, in particular ambiguity removal and linearization. Since version 2.1, it is possible to search the text automaton for patterns (see section 7.7).

7.1 Displaying text automaton

The text automaton explicit all possible lexical interpretations of the words. These different interpretations are the different entries presented in the dictionary of the text. Figure 7.1 shows the automaton of the fourth sentence of the text *Ivanhoe*.

You can see in Figure 7.1 that the word `Here` has three interpretations here (adjective, adverb and noun), `haunted` two (adjective and verb), etc. All the possible combinations are expressed because each interpretation of each word is connected to all the interpretations of the following and preceding words.

In case of an overlap between a compound word and a sequence of simple words, the automaton contains a path that is labeled by the compound word, parallel to the paths that express the combinations of simple words. This is illustrated in Figure 7.2, where the compound word `courts of law` overlaps with a combination of simple words.

By construction, the text automaton does not contain any loop. One says that the text automaton is *acyclic*.

NOTE: The term “text automaton” is an abuse of language. In fact, there is an automaton for each sentence of the text. Therefore, the combination of all these automata corresponds to the automaton of the text. This is why we use the term “text automaton” even if this object is not manipulated as a global automaton for practical reasons.

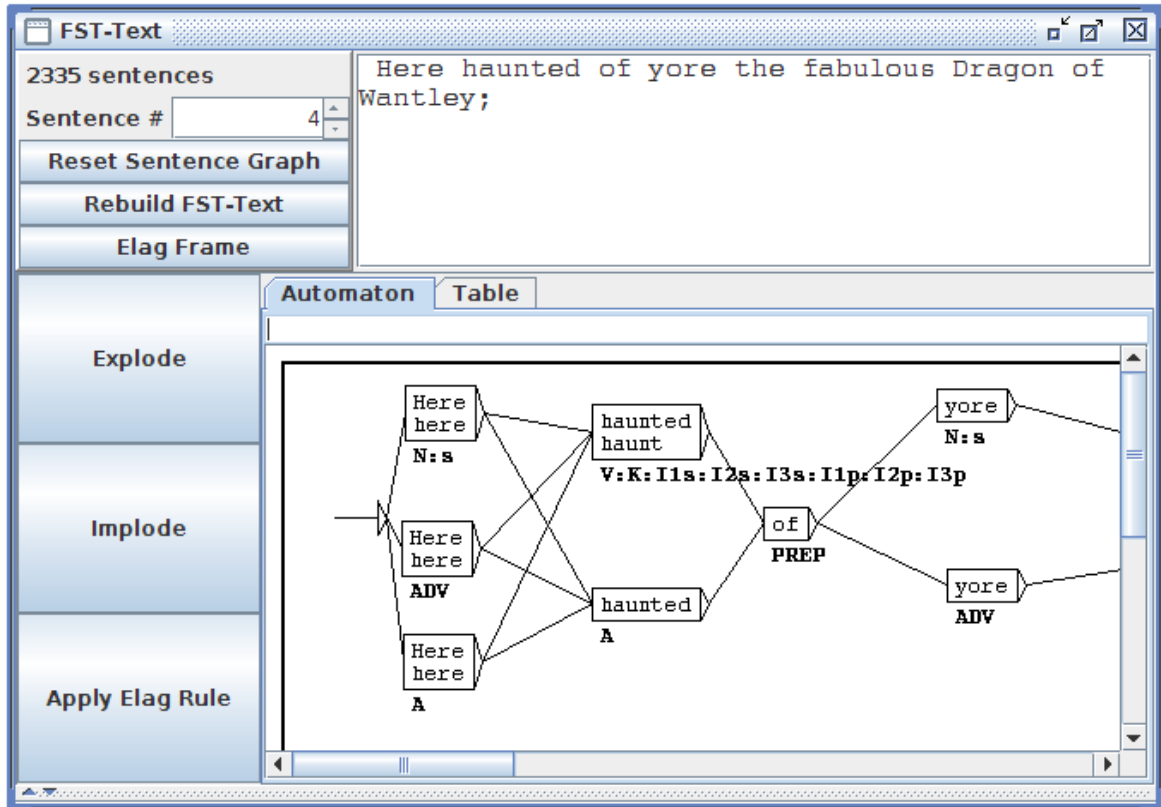


Figure 7.1: Sentence automaton example

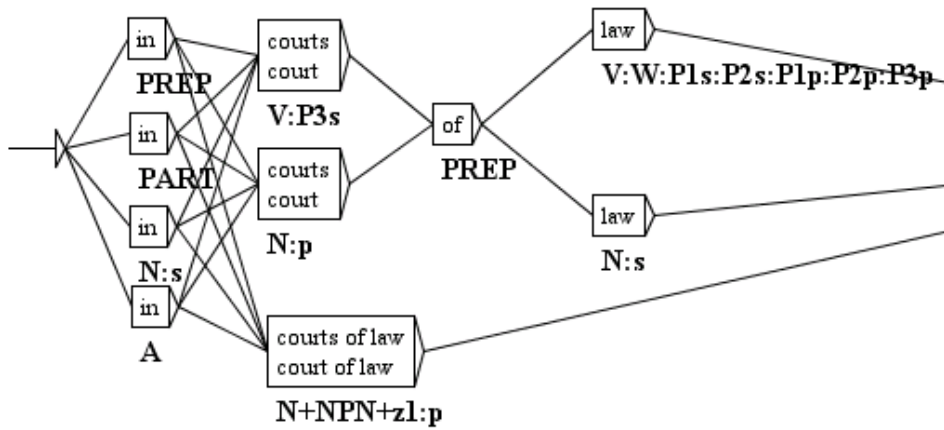


Figure 7.2: Overlap between a compound word and a combination of simple words.

7.2 Construction

In order to construct the text automaton, open the text, then click on "Construct FST-Text..." in the menu "Text". One should first split the text into sentences and apply dictionaries. If sentence boundary detection is not applied, the construction program will arbitrarily split the text in sequences of 2000 lexical units instead of constructing one automaton per sentence. If no dictionaries are applied, the text automaton that you obtain will consist of only one path made up of unknown words per sentence.

7.2.1 Construction rules for text automata

Sentence automata are constructed from text dictionaries. The resulting degree of ambiguity is therefore directly linked to the granularity of the descriptions of dictionaries. From the sentence automaton in figure 7.3, you can conclude that the word *which* has been coded twice as a determiner in two subcategories of the category *DET*. This granularity of descriptions will not be of any use if you are only interested in the grammatical category of this word. It is therefore necessary to adapt the granularity of the dictionaries to the intended use.

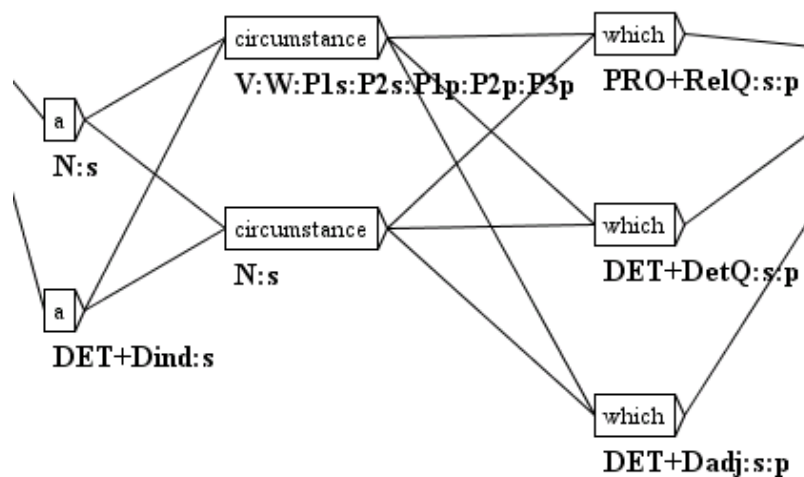


Figure 7.3: Double entry for *which* as a determiner

For each lexical unit of the sentence, Unitex searches the dictionary of the simple words of the text for all possible interpretations. Afterwards, all combination of lexical units that have an interpretation in the dictionary of the compound words of the text are taken into account. All the combinations of these information constitute the sentence automaton.

NOTE: If the text contains lexical labels (e.g. {out of date, .A+z1}), these labels are reproduced identically in the automaton without trying to decompose them.

In each box, the first line contains the inflected form found in the text, and the second line contains the canonical form if it is different. The other information is coded below the box. (cf. section 7.5.1).

The spaces that separate the lexical units are not copied into the automaton except for the spaces inside compound words.

The case of lexical units is retained. For example, if the word *Here* is encountered, the capital letter is preserved (cf. figure 7.1). This choice allows you to keep this information during the transition to the text automaton, which could be useful for applications where case is important as for recognition of proper names.

7.2.2 Normalization of ambiguous forms

During construction of the automaton, it is possible to effect a normalization of ambiguous forms by applying a normalization grammar. This grammar has to be called `Norm.fst2` and must be placed in your personal folder, in the subfolder `/Graphs/Normalization` of the desired language. The normalization grammars for ambiguous forms are described in section 6.1.3.

If a sequence of the text is recognized by the normalization grammar, all the interpretations that are described by the grammar are inserted into the text automaton. Figure 7.4 shows the part of the grammar used for the ambiguity of the sequence `l'` in French.

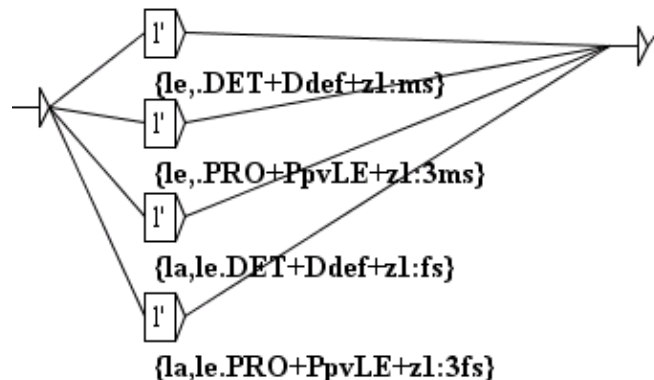


Figure 7.4: Normalization of the sequence `l'`

If this grammar is applied to a French sentence containing the sequence `l'`, a sentence automaton that is similar to the one in figure 7.5 is obtained.

You can see that the four rules for rewriting the sequence `l'` have been applied, which has added four labels to the automaton. These labels are not concurrent with the two preexisting paths for the sequence `l'`, because of the "keep best paths" heuristic (see section 7.2.4). The normalization at the time of the construction of the automaton allows you to add paths to the

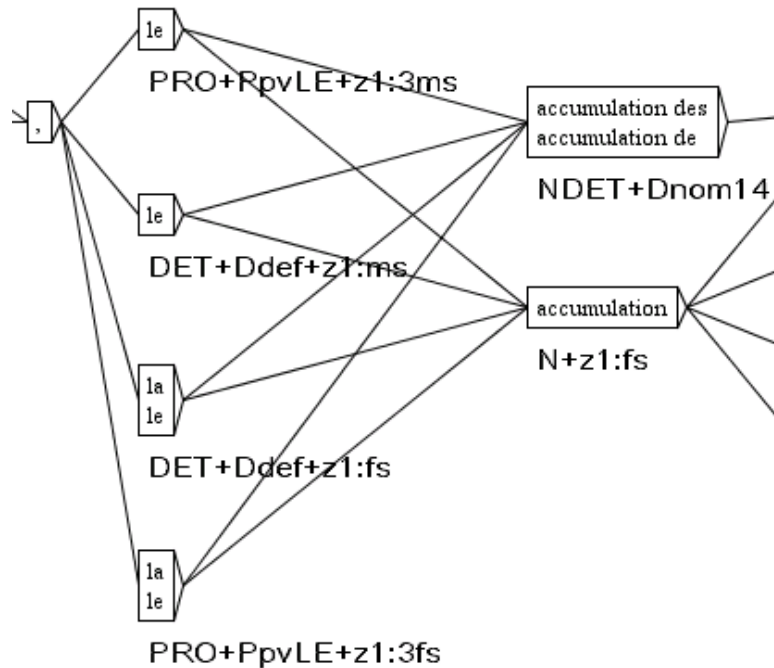


Figure 7.5: Automaton that has been normalized with the grammar of figure 7.4

automaton but not to remove ones. Removing paths will be partially done by the "keep best paths" heuristic, if enabled. To go further, you will need to use the ELAG disambiguation functionality.

7.2.3 Normalization of clitical pronouns in Portuguese

In Portuguese, verbs in the future tense and in the conditional can be modified by the insertion of one or two clitical pronouns between the root and the suffix of the verb. For example, the sequence *dir-me-ão* (*they will tell me*), corresponds to the complete verbal form *dirão*, associated with the pronoun *me*. In order to be able to manipulate this rewritten form, it is necessary to introduce it into the text automaton in parallel to the original form. Thus, the user can search one or the other form. The figures 7.6 and 7.7 show the automaton of a sentence after normalization of the clitics.

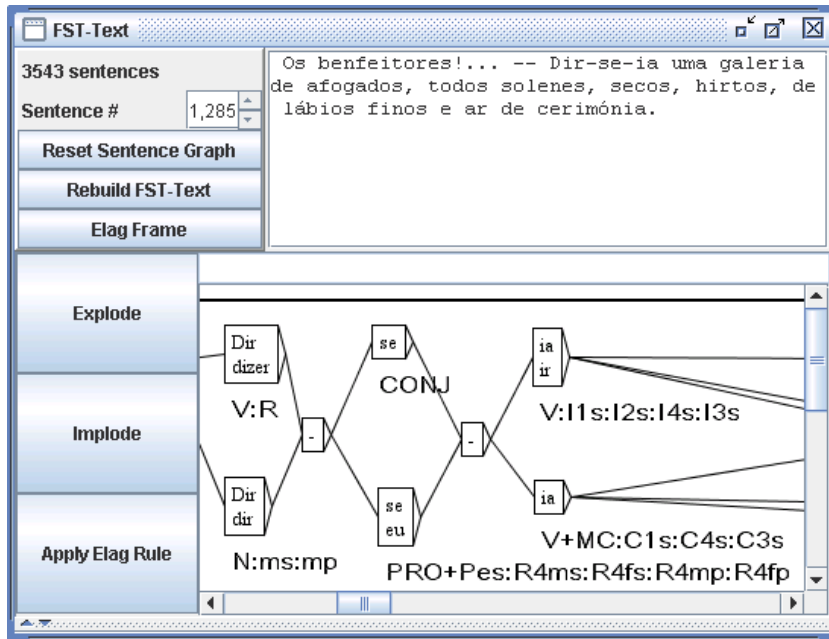


Figure 7.6: Non-normalized phrase automaton

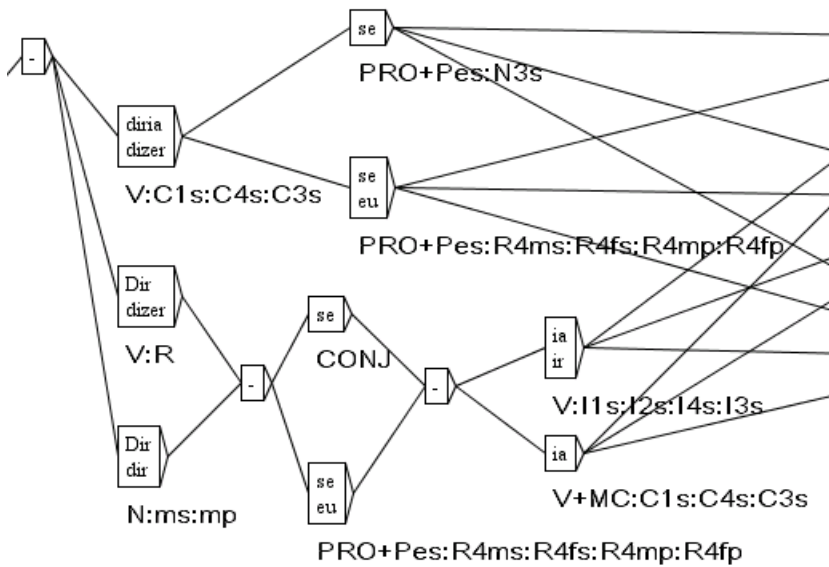


Figure 7.7: Normalized phrase automaton

The *Reconstrucao* program allows you to construct a normalization grammar for these forms for each text dynamically. The grammar thus produced can then be used for normalizing the text automaton. The configuration window of the automaton construction suggests an option "Build clitic normalization grammar" (cf. figure 7.10). This option automatically starts the construction of the normalization grammar, which is then used to construct the text automaton, if you have selected the option "Apply the Normalization grammar".

7.2.4 Keeping the best paths

An unknown word can perturb the text automaton by overlapping with a completely labeled sequence. Thus, in the automaton of figure 7.8, it can be seen that the adverb *aujourd'hui* overlaps with the unknown word *aujourd*, followed by an apostrophe and the past participle of the verb *huir*.

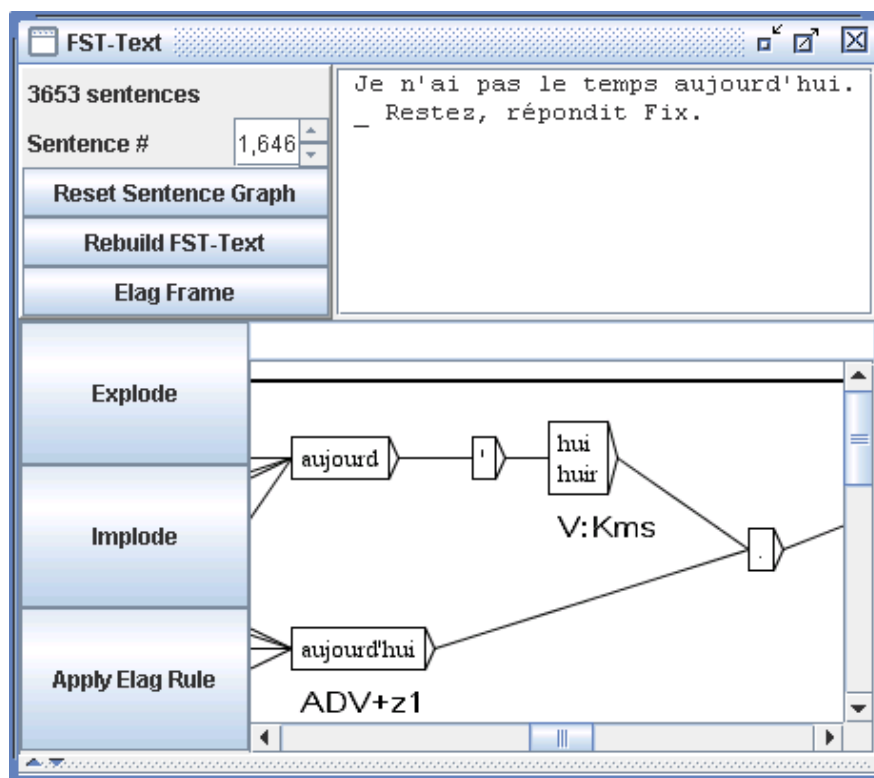


Figure 7.8: Ambiguity due to a sentence containing an unknown word

This phenomenon can also take place in the treatment of certain Asian languages like Thai. When words are not delimited, there is no other solution than to consider all possible combinations, which causes the creation of numerous paths carrying unknown words that are mixed with the labeled paths. Figure 7.9 shows an example of such an automaton of a Thai sentence.

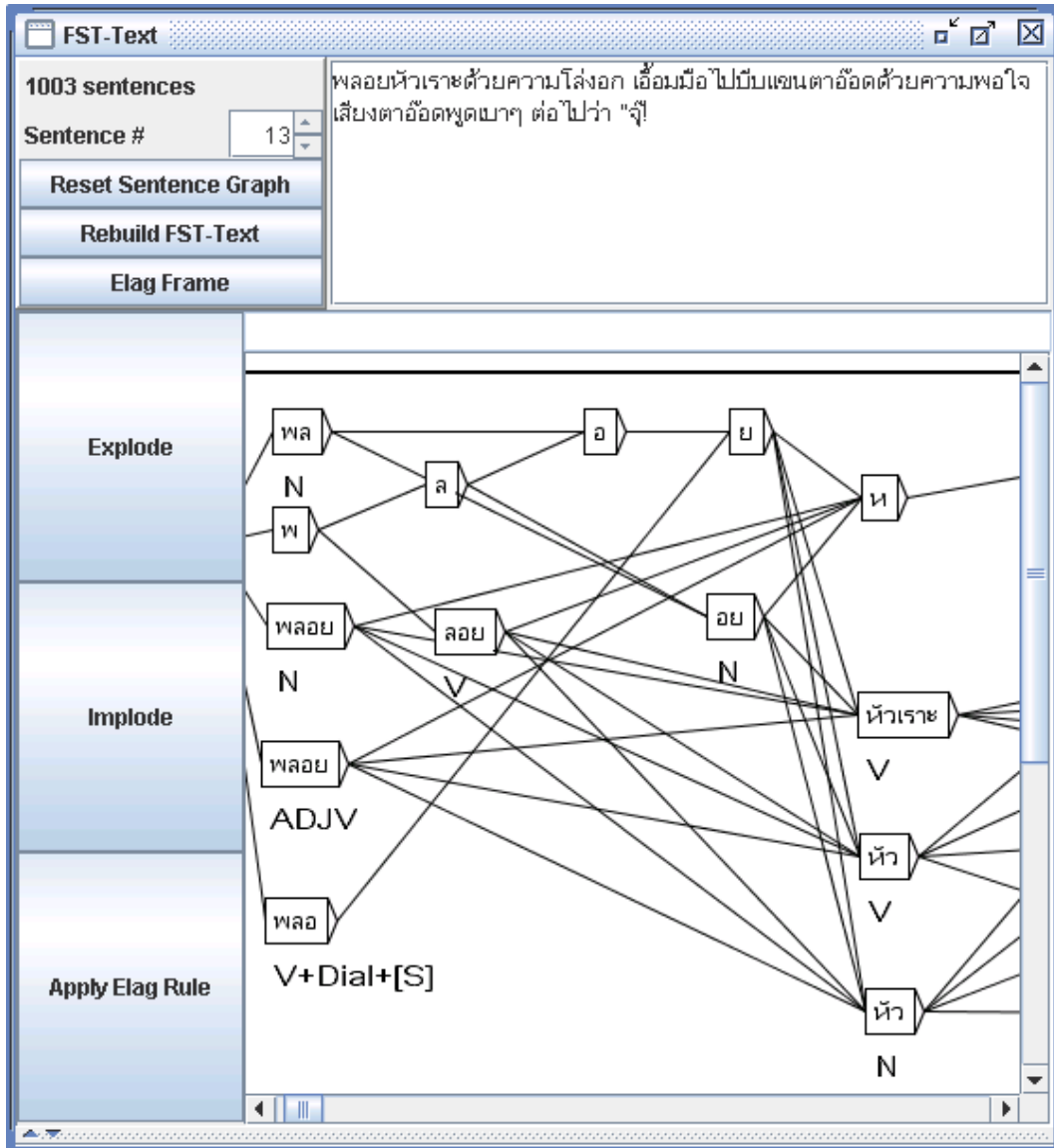


Figure 7.9: Automaton of a Thai sentence

It is possible to suppress parasite paths. You have to select the option "Clean Text FST" in the configuration window for the construction of the text automaton (cf. figure 7.10). This option indicates to the automaton construction program that it should clean up each sentence automaton.

This cleaning is carried out according to the following principle: if several paths are concurrent in the automaton, the program keeps those that contain the fewest unlabeled tokens. For instance, the compound adverb *aujourd'hui* is preferred to the sequence made of *aujourd* followed by a quote and *hui*, because *aujourd* and the quote are both unlabeled

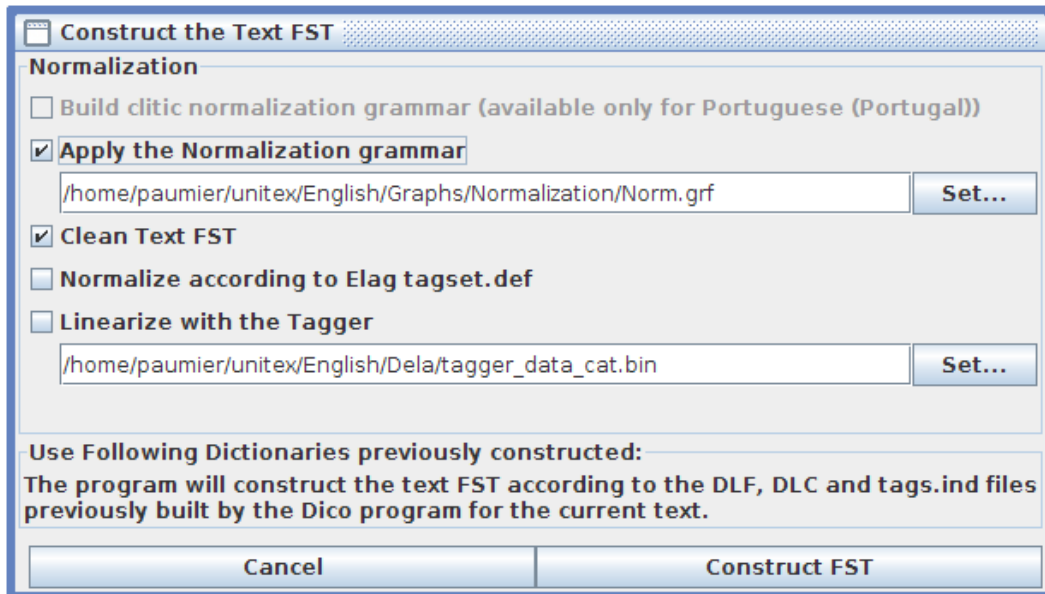


Figure 7.10: Configuration of the construction of the text automaton

tokens, while the compound adverb path does not contain any unknown word. Figure 7.11 shows the automaton of figure 7.9 after cleaning.

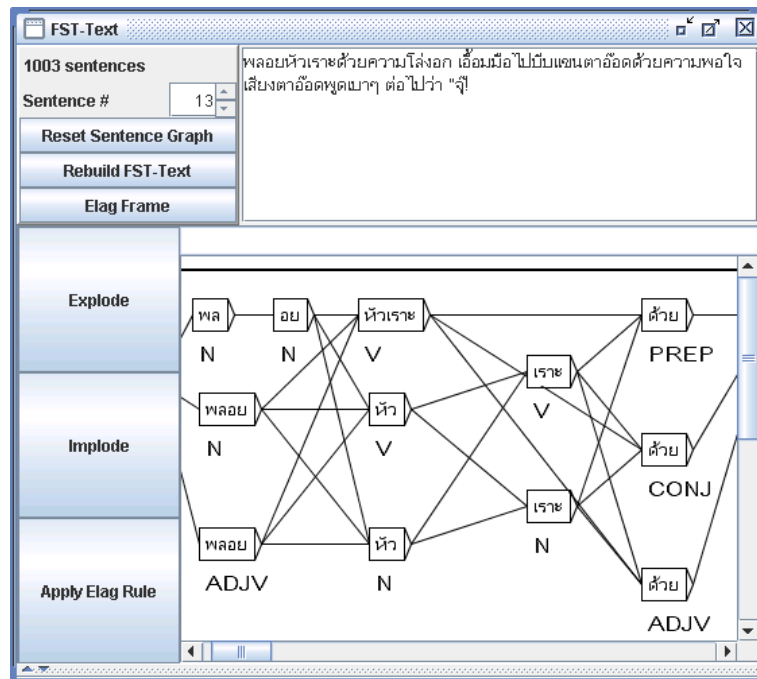


Figure 7.11: Automaton of figure 7.9 after cleaning

7.3 Resolving Lexical Ambiguities with ELAG

The ELAG program allows for applying grammars for ambiguity removal to the text automaton. This powerful mechanism makes it possible to write rules on independently from already existing rules. This chapter briefly presents the grammar formalism used by ELAG and describes how the program works. For more details, the reader may refer to [6] and [62].

7.3.1 Grammars For Resolving Ambiguities

The grammars used by ELAG have a special syntax. They consist of two parts which we call the *if* and *then* parts. The *if* part of an ELAG grammar is divided in two parts which are divided by a box containing the `<!>` symbol. The *then* part is divided the same way using the `<=>` symbol. The meaning of a grammar is the following: In the text automaton, if a path of the *if* part is recognized, then it must also be recognized by the *then* part of the grammar, or it will be withdrawn from the text automaton.

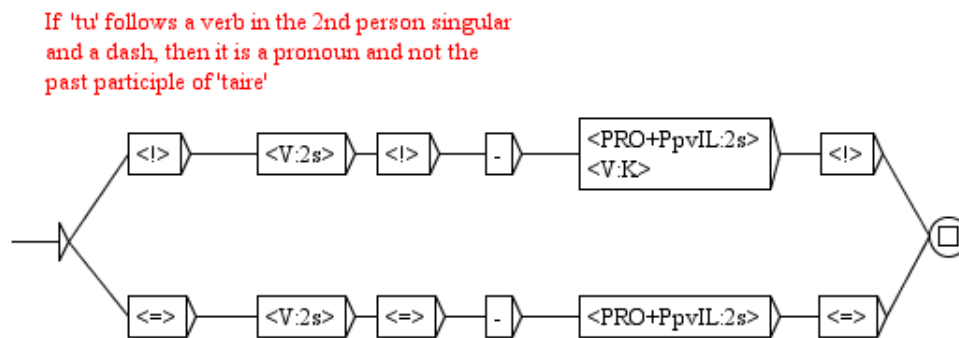


Figure 7.12: ELAG grammar `elag-tu.grf`

Figure 7.12 shows an example of a grammar. The *if* part recognizes a verb in the 2nd person singular followed by a dash and `tu`, either as a pronoun, or as a past participle of the verb `taire`. The *then* part imposes that `tu` is then regarded as a pronoun. Figure 7.13 shows the result of the application of this grammar on the sentence "*Feras-tu cela bientôt ?*". One can see in the automaton at the bottom that the path corresponding to `tu` past participle was eliminated.

Synchronization point

The *if* and *then* parts of an ELAG grammar are divided into two parts by `<!>` in the *if* part, and `<=>` in the *then* part. These symbols form a *synchronization point*. This makes it possible to write rules in which the *if* and *then* constraints are not necessarily aligned, as it is the case for example in figure 7.14. This grammar is interpreted in the following way: if a dash is found followed by `il`, `elle` or `on`, then this dash must be preceded by a verb, possibly

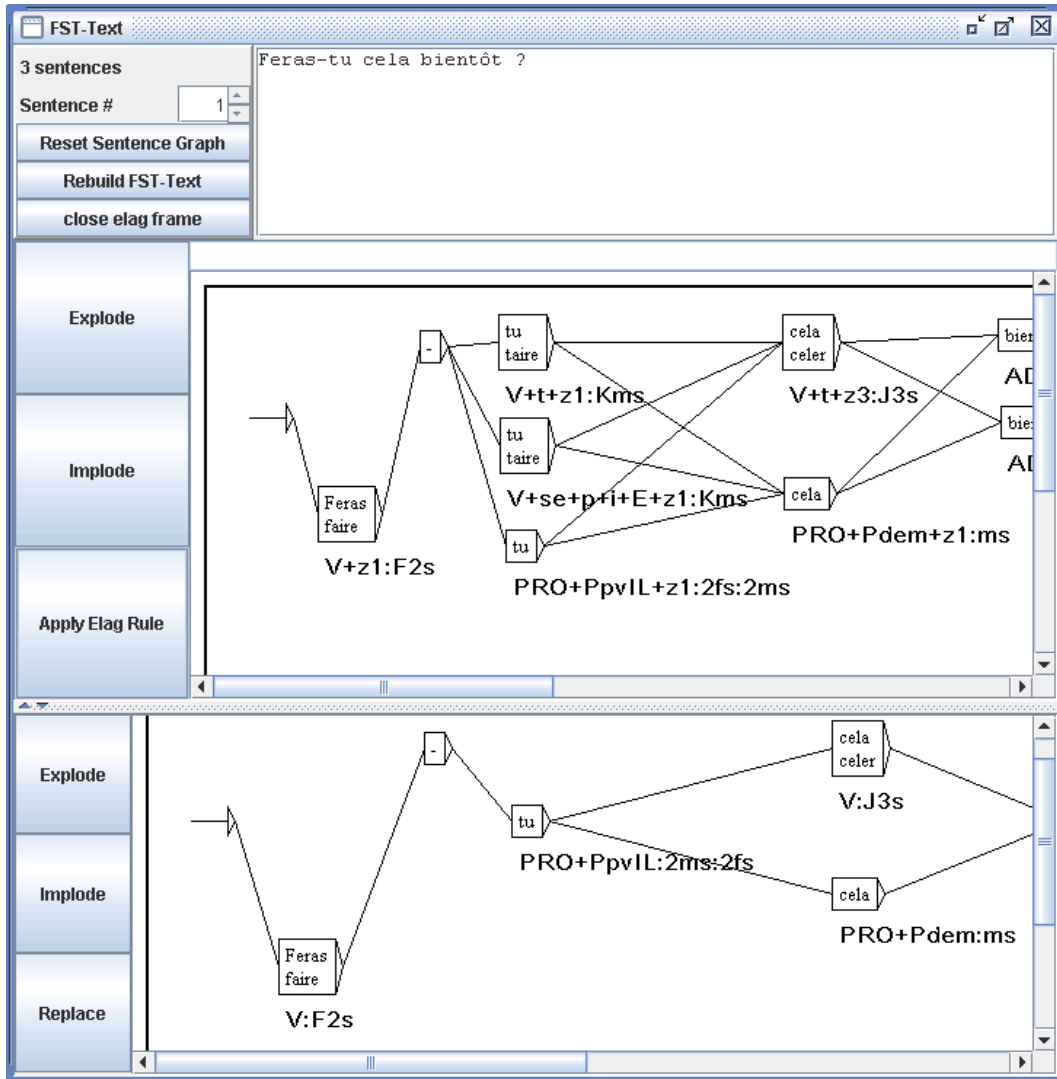


Figure 7.13: Result of applying the grammar in figure 7.12

followed by $-t$. So, if one considers the sentence of the figure 7.15 beginning with *Est-il*, one can see that all non-verb interpretations of *Est* were removed.

7.3.2 Compiling ELAG Grammars

Before an ELAG grammar can be applied to a text automaton, the grammar must be compiled in a `.rul` file. This operation is carried out via the "Elag Rules" command in the "Text" menu, which opens the windows shown in figure 7.16.

If the frame on the right already contains grammars which you don't wish to use, you can withdraw them with the " \ll " button. Then select your grammar(s) in the file explorer located

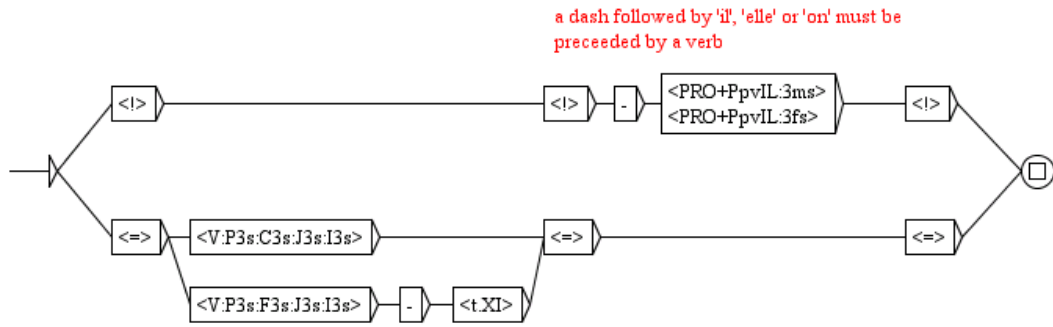


Figure 7.14: Use of the synchronization point

Figure 7.15: Result of the application of the grammar in figure 7.14

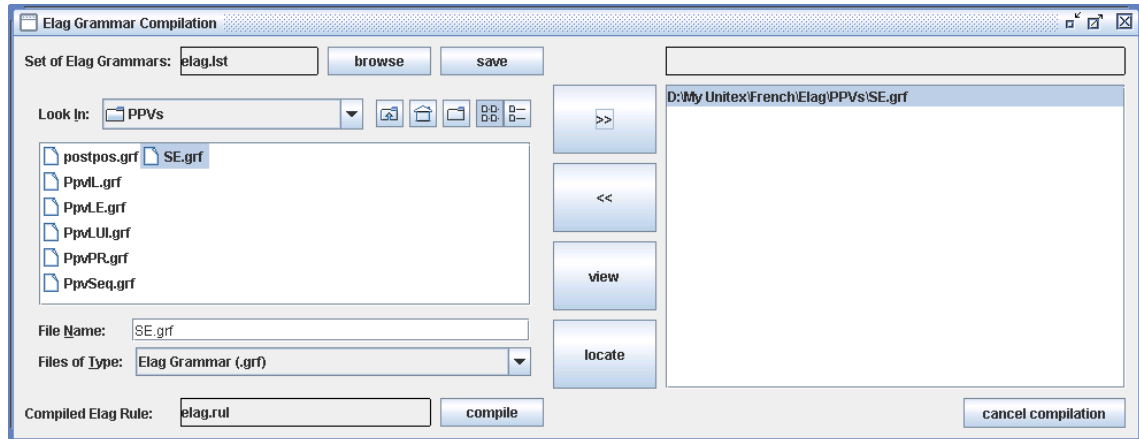


Figure 7.16: ELAG grammars compilation frame

in the left frame, and click on the ">>" button to add them to the list in the right frame. Then click on the "Compile" button. This will launch the `ElagComp` program which will compile the selected grammars and create a file named `elag.rul` by default.

If you have selected grammars in the right frame, you can search patterns within them by clicking on the "Locate" button. This opens the window "Locate Pattern" and automatically enters a graph name ending with `-conc.fst2`. This graph corresponds to the *if* part of the grammar. You can thus obtain the occurrences of the text to which the grammar will apply.

NOTE: The `-conc.fst2` file used to locate the *if* part of a grammar is automatically generated when ELAG grammars are compiled by means of the "Compile" button. It is thus necessary to have your grammar compiled before searching using the "Locate" button.

7.3.3 Resolving Ambiguities

Once you have compiled your grammar into an `elag.rul` file, you can apply it to a text automaton. In the text automaton window, click on the "Apply Elag Rule" button. A dialog box will appear which asks for the `.rul` file to be used (see figure 7.17). The default file is `elag.rul`. This will launch the `Elag` program which will try to resolve the ambiguity.

Once the program has finished you can view the resulting automaton by clicking on the "Open Elag Frame" button. As you can see in figure 7.18, the window is separated into two parts: The original text automaton can be seen on the top, and the result at the bottom.

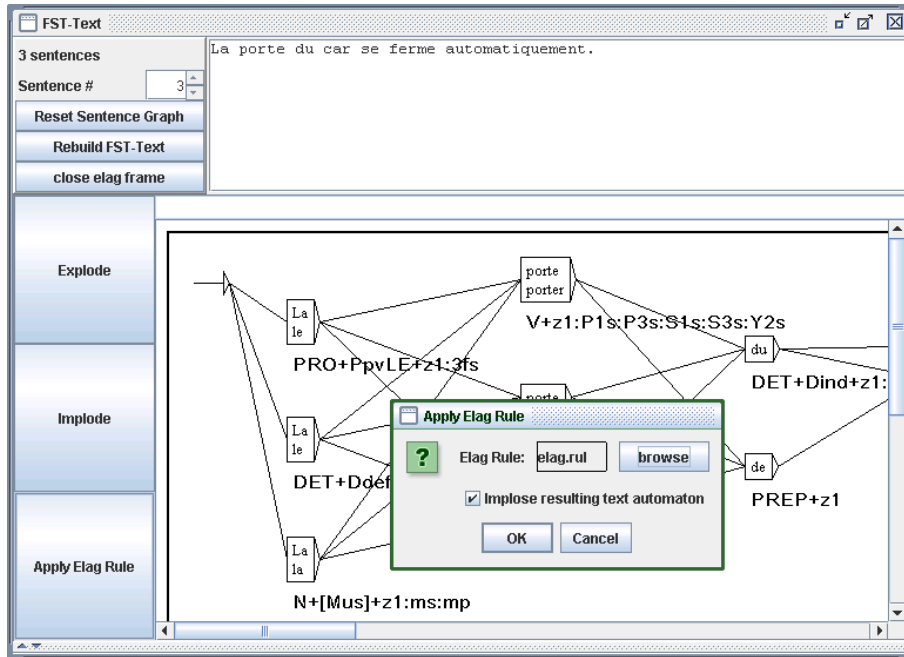


Figure 7.17: Text automaton frame

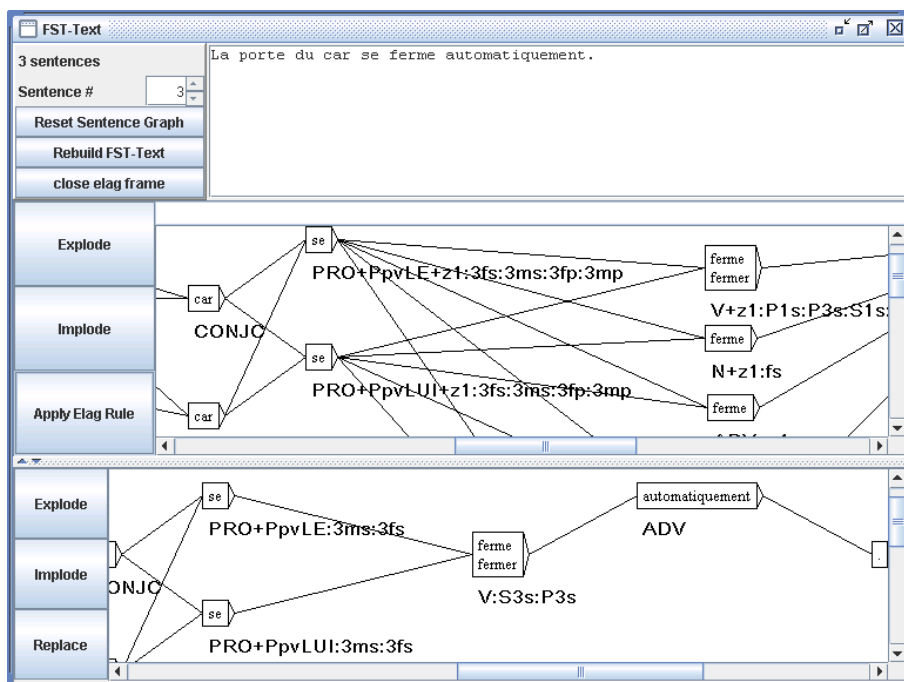


Figure 7.18: Splitted text automaton frame

Don't be surprised if the automaton shown at the bottom seems more complicated. This results from the fact that factorized lexical entries¹ were exploded in order to treat each inflectional interpretation separately. To refactorize these entries, click on the "Implode" button. Clicking on the "Explode" button shows you an exploded view of the text automaton.

If you click on the "Replace" button, the resulting automaton will become the new text automaton. Thus, if you use other grammars, they will apply to the already partially disambiguated automaton, which makes it possible to accumulate the effects of several grammars.

7.3.4 Grammar collections

It is possible to gather several ELAG grammars into a grammar collection in order to compile and apply them in one step. The sets of ELAG grammars are described in `.lst` files. They are managed through the window for compiling ELAG grammars (figure 7.16). The label on the top left indicates the name of the current collection, by default `elag.lst`. The contents of this collection are displayed in the right part of the window.

To modify the name of the collection, click on the "Browse" button. In the dialog box that appears, enter the `.lst` file name for the collection.

To add a grammar to the collection, select it in the file explorer in the left frame, and click on the "»" button. Once you have selected all your grammars, compile them by clicking on the "Compile" button. This will create a `.rul` file bearing the name indicated at the bottom right (the name of the file is obtained by replacing `.lst` by `.rul`).

You can now apply your grammar collection. As explained above, click on the "Apply Elag Rule" button in the text automaton window. When the dialog asks for the `.rul` file to use, click on the "Browse" button and select your collection. The resulting automaton is identical to that which would have been obtained by applying each grammar successively.

7.3.5 Window For ELAG Processing

At the time of disambiguation, the `Elag` program is launched in a processing window which displays the messages printed by the program during its execution.

For example, when the text automaton contains symbols which do not correspond to the set of ELAG labels (see the following section), a message indicates the nature of the error. In the same way, when a sentence is rejected (all possible analyses were eliminated by grammars), a message indicates the number of the sentence. That makes it possible to locate the source of the problems quickly.

¹Entries which gather several different inflectional interpretations, such as for example:
`{se, .PRO+PpvLE:3ms:3fs:3mp:3fp}`.

Evaluation of ambiguity removal

The evaluation of the ambiguity rate is not based solely on the average number of interpretations per word. In order to get a more representative measure, the system also takes into account the various combinations of words. While instances of ambiguities are resolved, the `Elag` program calculates the number of possible analyses in the text automaton before and after the modification (which corresponds to the number of possible paths through the automaton). On the basis of this value, the program computes the average ambiguity by sentence and word. It is this last measure which is used to represent the ambiguity rate of the text, because it does not vary with the size of the corpus, nor with the number of sentences within. The formula applied is:

$$\text{lexical ambiguity rate} = \exp \frac{\log(\text{number-of-paths})}{\text{text-length}}$$

The relationship between the ambiguity rate before and after applying the grammars gives a measure of their efficiency. All this information is displayed in the ELAG processing window.

7.3.6 Description of the tag sets

The `Elag` and `ElagComp` programs require a formal description of the tag set to be used in dictionaries. This description consists essentially of an enumeration of all the parts of speech present in the dictionaries, with, for each of them, the list of syntactic and inflectional codes compatible with it, and a description of their possible combinations. This description must be contained in a file called `tagset.def` and placed in your personal folder, in the `Elag` subfolder of the desired language.

tagset.def file

Here is an extract of the `tagset.def` file used for French.

```
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     <genre> <nombre> <pers>      # en y
PronQ     <genre> <nombre> <pers>      # ou qui que quoi
Dnom      <genre> <nombre> <pers>      # 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>
.

```

The # symbol indicates that the remainder of the line is a comment. A comment can appear at any place in the file. The file always starts with the word `NAME`, followed by an identifier (`french`, for example). This is followed by the `POS` sections for each part of speech. Each section describes the structure of the lexical tags of the lexical entries belonging to the part of speech concerned. Each section is composed of 4 parts which are all optional:

- `flex`: this part enumerates the inflectional codes belonging to the grammatical category. For example, the codes 1, 2, 3 which indicate the person of the entry are relevant for pronouns but not for adjectives. Each line describes an inflectional attribute (gender, time, etc.) and is made up of the attribute name, followed by the = character and the values which it can take. For example, the following line declares an attribute `pers` being able to taking the values 1, 2 or 3:

```
pers = 1 2 3
```

- `cat`: this part declares the syntactic and semantic attributes which can be assigned to the entries belonging to the part of speech concerned. Each line describes an attribute and the values which it can take. The codes declared for the same attribute must be exclusive. In other words, an entry cannot take more than one value for the same attribute.

On the other hand, all the tags in a given part of speech don't necessarily take values for all the attribute of the part of speech. For example, to define the attribute `niveau_de_langue` which can take the values `z1`, `z2` and `z3`, the following line can be written:

```
niveau_de_langue = z1 z2 z3
```

but this attribute is not necessarily present in all words.

- `discr`: this part consists of a declaration of a unique attribute. The syntax is the same as in the `cat` part and the attribute described here must not be repeated there. This part allows for dividing the grammatical category in *discriminating* sub categories in which the entries have similar inflectional attributes. For pronouns for example, a person feature is assigned to entries that are part of the personal pronoun sub category but not to relative pronouns. These dependencies are described in the `complete` part;

- `complete`: this part describes the inflectional part of the tags of the words in the current part of speech. Each line describes a valid combination of inflectional codes by their discriminating sub category (if such a category was declared). If an attribute name is specified in angle brackets (< and >), this signifies that any value of this attribute may occur. It is possible as well to declare that an entry does not take any inflectional feature by means of a line containing only the `_` character (underscore). So for example, if we consider that the following lines extracted from the section describing the verbs:

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

They make it possible to declare that verbs in the infinitive (indicated by the `W` code) do not have other inflectional features while the forms in the past participle (`K` code) are also assigned a gender and a number.

Description of the inflectional codes

The principal function of the `discr` part is to divide a part of speech into subcategories having similar inflectional behavior. These subcategories are then used to facilitate writing the `complete` part.

For the legibility of the ELAG grammars, it is desirable that the elements of the same subcategory all have the same inflectional behavior; in this case the `complete` part is made up of only one line per subcategory. Let us consider for example the following lines from the pronoun description:

```
Pdem <genre> <nombre>
PpvI1 <genre> <nombre> <pers>
PpvPr
```

These lines mean:

- all the demonstrative pronouns (`PRO+Pdem`) have only a gender and a number;
- clitic pronouns in the nominative (`<PRO+PpvI1>`) are labelled grammatically in person, gender and number;
- the prepositional pronouns (`en, y`) do not have any inflectional feature.

All combinations of inflectional features and discriminant subcategories which appear in the dictionaries must be described in the `tagset.def` file; otherwise, the information in the corresponding entries will be discarded by ELAG.

If words of the same subcategory differ by their inflectional profile, it is necessary to write several lines into the `complete` part. The disadvantage of this method of description is that it becomes difficult to make the distinction between such words in an ELAG grammar.

If one considers the description given by the previous example of a `tagset.def` file, certain adjectives of French take a gender and a number, whereas others do not have any inflectional feature. This allows for coding fixed sequences like *de bonne humeur* as adjective, on the basis of their syntactic behavior.

Consider a French dictionary with such sequences as invariable adjectives without inflectional features. The problem is that if one wants to refer exclusively to this type of adjectives in a disambiguation grammar, the `<A>` symbol is not appropriate, since it will recognize all adjectives. To circumvent this difficulty, it is possible to deny an inflectional attribute by writing the `@` character right before one of the possible values for this attribute. Thus, the `<A:@m@p>` symbol recognizes all the adjectives which have neither a gender nor a number. Using this operator, it is possible to write grammars like those in figure 7.19, which imposes agreement in gender and number between a name and an adjective which suits². This grammar will preserve the correct analysis of sentences like: *Les personnes de bonne humeur m'insupportent*.

It is however recommended to limit the use of the `@` operator, because it harms the legibility of the grammars. It is preferable to distinguish the labels which accept various inflectional combinations by means of discriminating subcategories defined in the `discr` part.

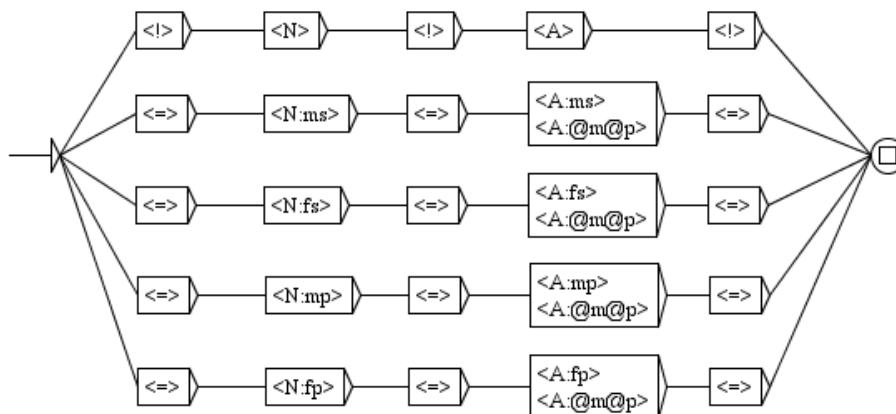


Figure 7.19: ELAG grammar that verifies gender and number agreement

Optional Codes

The optional syntactic and semantic codes are declared in the `cat` part. They can be used in ELAG grammars like other codes. The difference is that these codes do not intervene to

²This grammar is not completely correct, because it eliminates for example the correct analysis of the sentence: *J'ai reçu des coups de fil de ma mère hallucinants*.

decide if a label must be rejected as an invalid one while loading of the text automaton.

In fact optional codes are independent of other codes, such as for example the attribute of the language level (z_1 , z_2 or z_3). In the same manner as for inflectional codes, it is possible to deny an inflectional attribute by writing the `!` character right before the name of the attribute. Thus, with our example file, the `<A!gauche:f>` symbol recognizes all adjectives in the feminine which do not have the `gauche` code³.

All codes which are not declared in the `tagset.def` file are discarded by ELAG. If a dictionary entry contains such a code, ELAG will produce a warning and will withdraw the code from the entry.

Consequently, if two concurrent entries differ in the original text automaton only by undeclared codes, these entries will become indistinguishable by the programs and will thus be unified into only one entry in the resulting automaton.

Thus, the set of labels described in the file `tagset.def` file is compatible with the dictionaries distributed with Unitex, by factorizing words which differ only by undeclared codes, and this independently of the applied grammars.

For example, in the most complete version of the French dictionary, each individual use of a verb is characterized by a reference to the lexicon grammar table which contains it. We have considered until now that this information is more relevant to syntax than to lexical analysis and we thus don't have integrated them into the description of the tagset. They are thus automatically eliminated at the time when the text automaton is loaded, which reduces the rate of ambiguity.

In order to distinguish the effects bound to the tagset from those of the ELAG grammars, it is advised to proceed to a preliminary stage of normalization of the text automaton before applying disambiguation grammars to it. This normalization is carried out by applying to the text automaton a grammar not imposing any constraint, like that of figure 7.20. Note that this grammar is normally present in the Unitex distribution and precompiled in the file `norm.rul`.

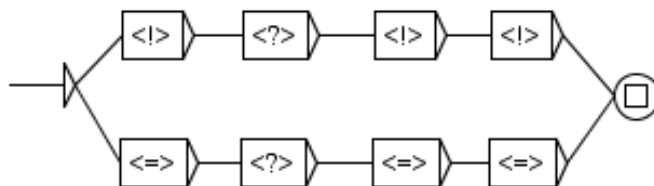


Figure 7.20: ELAG grammar without any constraint

³This code indicates that the adjective must appear on the left of the noun to which it refers to, as is the case for *bel*.

The result of applying such a grammar is that the original is cleaned of all the codes which either are not described in the `tagset.def` file, or do not conform to this description (because of unknown grammatical categories or invalid combinations of inflectional features). By then replacing the text automaton by this normalized automaton, one can be sure that later modifications of the automaton will only be effects of ELAG grammars.

7.3.7 Grammar Optimization

Compilation of ELAG grammars by the `ElagComp` program consists in building an automaton whose language is the set of the sequences of lexical tags (or lexical analyses of a sentence) which are not accepted by the grammars. This task is complex and can take a lot of time. It is however possible to appreciably speed it up by observing certain principles at the time of writing gramars.

Limiting the number of branches in the *then* part

It is recommended to limit the number of *then* parts of a grammar to a minimum. This can reduce considerably the compile time of a grammar. Generally, a grammar having many *then* parts can be rewritten with one or two *then* parts, without a loss of legibility. It is for example the case of the grammar in figure 7.21, which imposes a constraint between a verb and the pronoun which follows it.

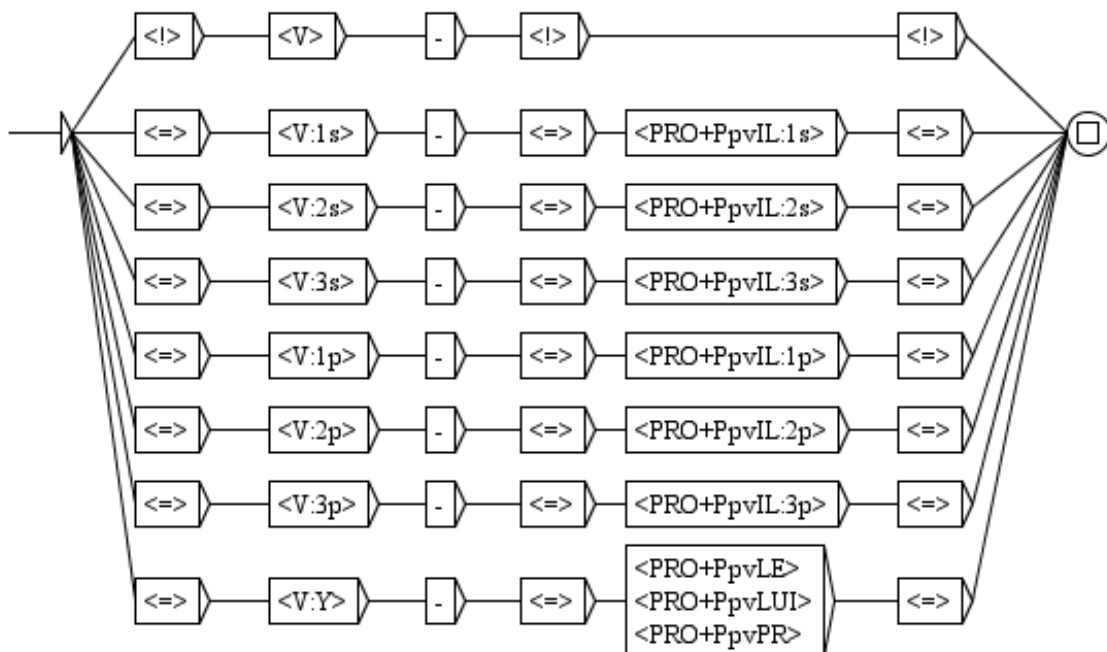


Figure 7.21: ELAG grammar checking verb-pronoun agreement

As one can see in figure 7.22, one can write an equivalent grammar by factorizing all the

then parts into only one. The two grammars will have exactly the same effect on the text automaton, but the second one will be compiled much more quickly.

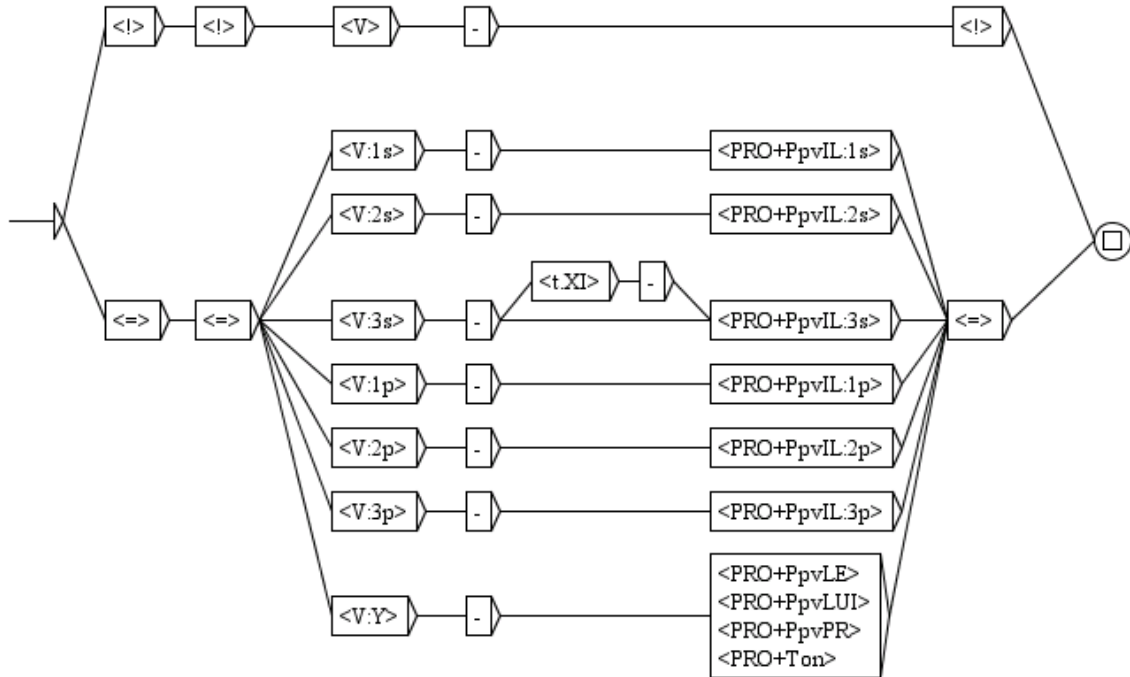


Figure 7.22: Optimized ELAG grammar checking verb-pronoun agreement

Using lexical symbols

It is better to use lemmas only when it is necessary. That is particularly true for some grammatical words, when their subcategories carry almost as much of information as the lemmas themselves. In any case, it is recommended to specify its syntactic, semantic and inflectional features as much as possible. For example, with the dictionaries provided for French, it is preferable to replace symbols like `<je.PRO:1s>`, `<je.PRO+PpvIL:1s>` and `<je.PRO>` with the symbol `<PRO+PpvIL:1s>`. Indeed, all these symbols are identical insofar as they can recognize only the single entry of the dictionary `{je, PRO+PpvIL:1ms:1fs}`. However, as the program does not deduce this information automatically, if all these features are not specified, the program will consider nonexistent labels such as `<je.PRO:3p>`, `<je.PRO+PronQ>` etc. in vain.

7.4 Linearizing text automaton with the tagger

By default, the text automaton contains many paths of tags because of lexical ambiguity. The linearization process consists in selecting a single path, a sequence of tags with one tag per token, and remove the others. The output of the process is a text automaton with a

single path (see section 7.6 for converting a linear automaton into linear text). The selection of a path depends on its score. The path with the best score is chosen and the others are removed. The score of a path is calculated using a statistical model trained on an annotated corpus. This model uses tagger data files generated by the TrainingTagger program (see section 12.39). For instance, you can see on Figure 7.23, the original text automaton of the French sentence *Les insectes nuisibles envahissent la maison*. The corresponding text automaton after linearization is shown on Figure 7.24.

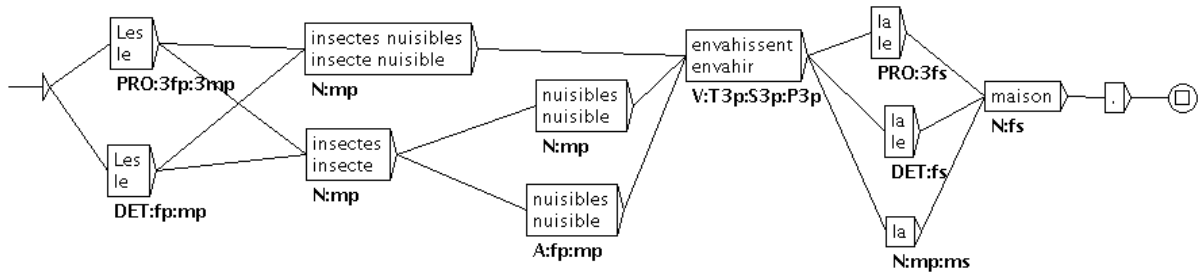


Figure 7.23: Text automaton of *Les insectes nuisibles envahissent la maison*.



Figure 7.24: Text automaton linearized

7.4.1 Compatibility of the tagset

The tagset of the tagger is identical to that of the training corpus or is a variant (see below). However, in order to use the tagger on a text automaton, one should pay attention to tagset and morphology. The tagset of the model must be identical to that of the text automaton. For example, if the statistical model has been computed with the tag `DET` for the word `the`, the corresponding tag in the text automaton must be `DET`. Unitex provides functionality to modify word forms in the text, for example to normalize `doesn't` into `does not`. Applying replacing or normalization graphs could cause some morphological modifications on words. If such processing is applied to the text, it must have been applied to the training corpus as well. If these rules are not respected, the tagger might not be able to keep the good path from the text automaton.

The TrainingTagger program produces two variants of the tagger. The first one prunes transitions on the basis of grammatical, semantic, syntactic and inflectional codes (for example, `the.DET+Ddef:s` versus `the.DET+Ddef:p`). The second one prunes transitions on the

basis of grammatical, semantic and syntactic codes (that .DET+Ddem versus that .PRO+Pdem). This option makes the training quicker and inflectional features are not needed for all applications.

7.4.2 Use of the Tagger

In order to linearize the text automaton, you have to select the option "Linearize with the Tagger" in the configuration window for the construction of the text automaton (cf. figure 7.25). With this option, the program will linearize each sentence automaton. You must also select the tagger data file (with ".bin" extension) by clicking on the "Set" button. Tagger data file suffixed by "morph" is the first variant of the tagger (with inflectional codes) and the one suffixed by "cat" is the second variant (without inflectional codes). If you want to use the "morph" data, you also need to click on "Normalize according to Elag tagset.def" (for more details, see section 12.33 about Tagger program).

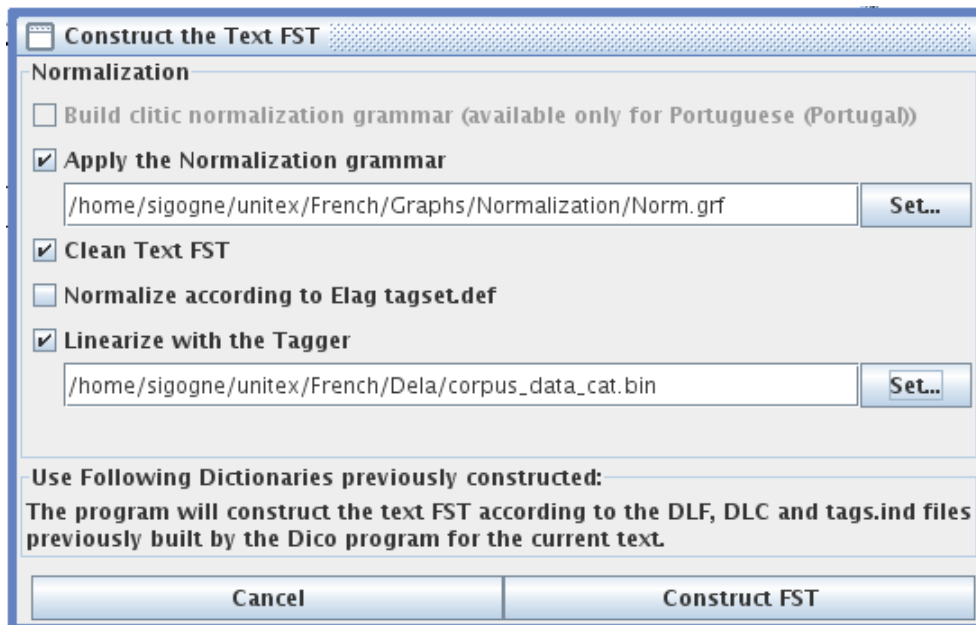


Figure 7.25: Configuration of the linearization of the text automaton

For instance, the text automaton, shown on Figure 7.24, is the output of linearization of the text automaton shown on Figure 7.23 with "cat" tagger data. Linearization of the automaton with "morph" tagger data is shown on Figure 7.26.

7.4.3 Creation of a new tagger

In order to create a new tagger for your language, you need to launch the TrainingTagger program on your own annotated corpus. The format of the annotated corpus is described in

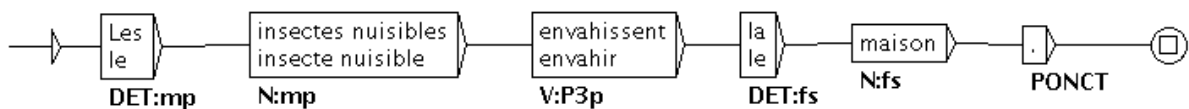


Figure 7.26: Text automaton linearized with "morph" tagger data

13.10.1. As we discuss in Section 7.4.1, you need to pay attention on tagset and morphology. Before computing a statistical model, you have to decide which dictionaries and normalization graphs you will use to construct the text automaton. And then, you will have to do modifications on the annotated corpus if word forms or tagset do not match completely. For example, if the normalization graph transforms the word *jusqu'* into *jusque*, the corresponding word into the annotated corpus must be *jusque*.

A French tagger is distributed with Unitex. It has been created with an annotated corpus composed of tags without semantic and syntactic codes.

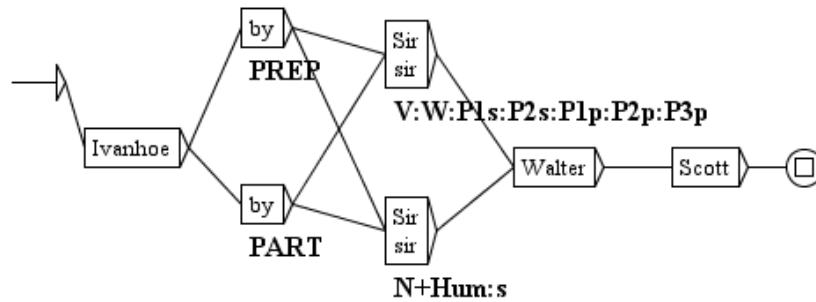
7.5 Manipulation of text automata

7.5.1 Displaying sentence automata

As we have seen above, the text automaton is in fact the collection of the sentence automata of a text. This structure can be represented using the format `.fst2`, also used for representing the compiled grammars. This format does not allow the system to directly display the sentence automata. Instead, the system uses the `Fst2Grf` program to convert the sentence automaton into a graph that can be displayed. This program is called automatically when you select a sentence in order to generate the corresponding `.grf` file.

The generated `.grf` files are not interpreted in the same manner as the `.grf` files that represent graphs constructed by the user. In fact, in a normal graph, the lines of a box are separated by the `+` symbol. In the graph of a sentence, each box represents either a lexical unit without a tag or a dictionary entry enclosed by curly brackets. If the box only represents an unlabeled lexical unit, this unit appears alone in the box. If the box represents a dictionary entry, the inflected form is displayed, followed in another line by the canonical form if it is different. The grammatical and inflectional information is displayed below the box as a transducer output.

Figure 7.27 shows the graph obtained for the first sentence of *Ivanhoe*. The words *Ivanhoe*, *Walter* and *Scott* are considered unknown words. The word *by* corresponds to two entries in the dictionary. The word *Sir* corresponds to two dictionary entries as well, but since the canonical form of these entries is *sir*, it is displayed because it differs from the inflected form by a lower case letter.

Figure 7.27: Automaton of the first sentence of *Ivanhoe*

7.5.2 Modifying the text automaton

It is possible to manually modify the sentence automaton. You can add or erase boxes or transitions. When a graph is modified, it is saved to the text file `sentenceN.grf`, where N represents the number of the sentence.

When you select a sentence, if a modified graph exists for this sentence, this one is displayed. You can then reset the automaton of that sentence by clicking on the button "Reset Sentence Graph" (cf. figure 7.28).

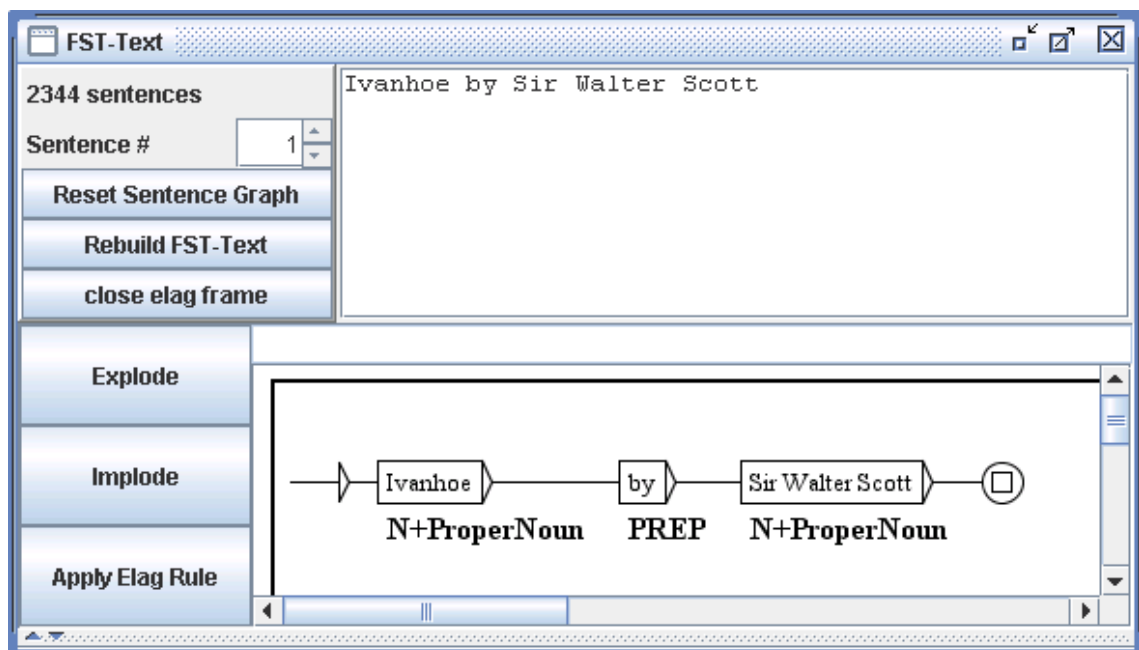


Figure 7.28: Modified sentence automaton

During the construction of the text automaton, all the modified sentence graphs in the text

file are erased.

NOTE: After you reconstruct the text automaton, you can save your manual modifications. In order to do that, click on the button "Rebuild FST-Text". All sentences that have been modified are then replaced in the text automaton by their modified versions. The new text automaton is then automatically reloaded.

7.5.3 Display configuration

Sentence automata are subject to the same presentation options as the graphs. They use the same colors and fonts as well as the antialiasing effect. In order to configure the appearance of the sentence automata, you modify the general configuration by clicking on "Preferences..." in the "Info" menu. For further details, refer to section 5.3.5.

You can also print a sentence automaton by clicking on "Print..." in the "FSGraph" menu or by pressing <Ctrl+P>. Make sure that the printer's page orientation is set to landscape mode. To configure this parameter, click on "Page Setup" in the "FSGraph" menu.

7.6 Converting the text automaton into linear text

If the text automaton does not contain any lexical ambiguity, it is possible to build a text file corresponding to the unique path of the automaton. Go into the "Text" menu and click on "Convert FST-Text to Text...". You can set the output text file in the window as shown on Figure 7.29.

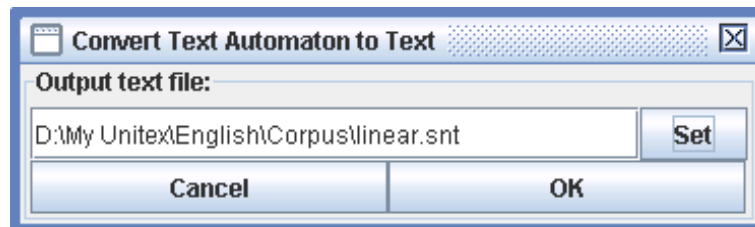


Figure 7.29: Setting output file for linearization of the text automaton

If the automaton is not linear, an error message will give you the number of the first sentence that contain ambiguity. Otherwise, the `Tfst2Unambig` program will build the output file according to the following rules:

- the output file contains one line per sentence;
- every line but the last is ended by `{S}`;
- for each box, the program writes its content followed by a space.

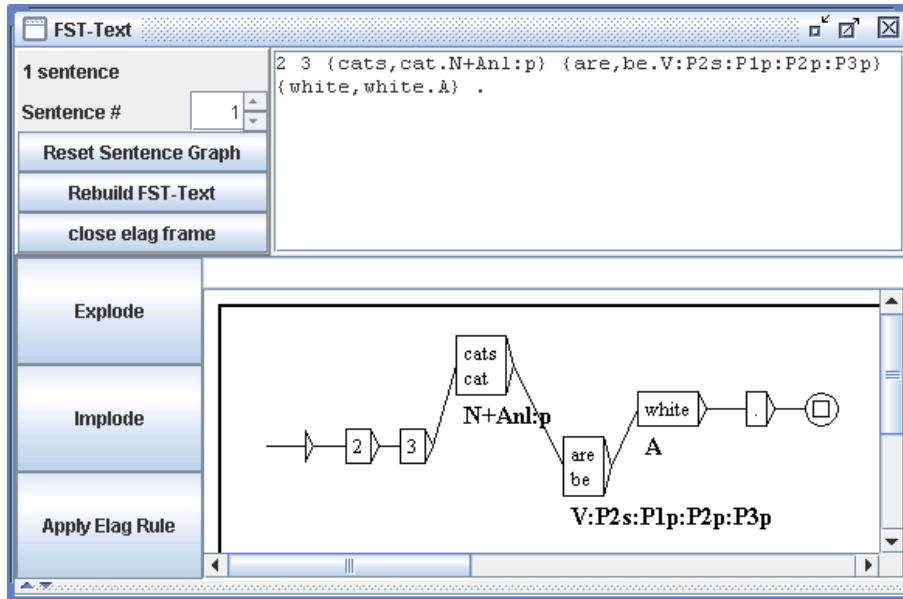


Figure 7.30: Example of a linear text automaton

NOTE: correcting spaces in the output text can only be done manually. If the original text is the one of the text automaton shown on Figure 7.30, the output text will be:

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

7.7 Searching patterns in the text automaton

With the `LocateTfst` program, Unitex can perform search operations on the text automaton. The main advantages are that you can:

- benefit from ambiguity removal;
- benefit from the application of normalization grammar (see below);
- work at several morphological levels (multi-word units, simple words, morphemes). This is particularly interesting since you can now easily manipulate agglutinative languages like Korean (for Korean, see section 7.9).

The rules are very similar to the ones that apply to classical searches with `Locate`. Here are the differences:

- you cannot capture sequences with variables inside right contexts, as it is possible with `Locate` (see Figure 6.17, page 120)

- you cannot match things that are not in the text automaton: if the text automaton only contains a compound word tag and not its concurrent simple word tags, you won't be able to match simple words. For instance, in the sentence automaton shown on Figure 7.31, it is not possible to match *soixante* or *huit*, since there are no such paths.

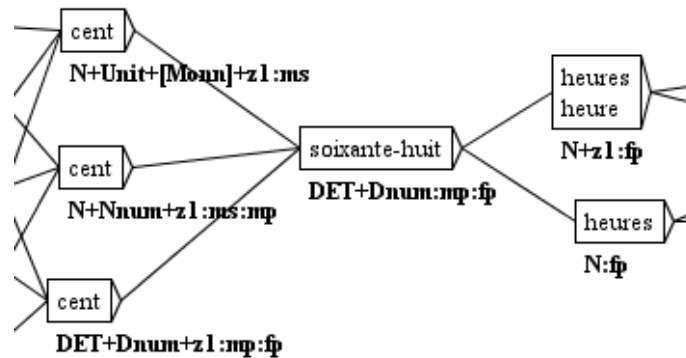


Figure 7.31: Sentence automaton that cannot match with pattern *huit*

- matched sequences can differ from sequences that will appear in concordances. In fact, the text automaton may contain tags that do not correspond to the raw input text, in particular when a normalization grammar has been applied. For instance, if you look for the pattern `<le.DET>` in *80journs's* text automaton, you will obtain 7703 matches, while `Locate` only finds 5763 matches. This is because some words have been normalized, like `au` → `iε`; `le` or `du` → `de le`. So, when you look for `<le.DET>`, `LocateTfst` matches those tags that were added to the text automaton by the normalization grammar, and `Concord` uses the original sequence in the text to produce the concordance file, as shown on Figure 7.32.

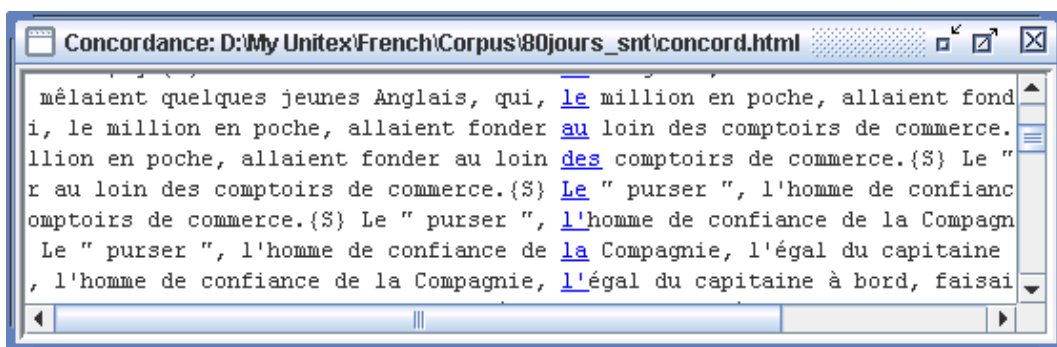


Figure 7.32: A surprising concordance for pattern `<le.DET>`

- `<TOKEN>` does not match tokens as defined in `tokens.txt`. It matches any tag of the text automaton. Matched tags can be either longer than text tokens if they are

compound word tags, or even shorter, if the text automaton contains morphological analysis like `un` as shown on Figure 3.16, page 68.

7.8 Table display

Sentence automata can be displayed in a table format. To do that, you just have to select the "Table" tab in the text automaton frame. You will then see a table as shown on Figure 7.33.

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 7.33: Table display

This table is not fully equivalent to the sentence automaton, since it only displays all possible POS for each simple or multiple word unit. It should be considered as an approximate compact view of information contained in the automaton. You can also filter grammatical/semantic codes to be displayed. Select "All" and you will see all codes. Select "Only POS category" and only first codes (supposed to represent the POS category) will be displayed. If you select "Use filter" and set a regular expression X , codes that do not contain something matched by X will be discarded. Any POSIX regular expression is accepted as filter. Check "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 7.34 shows a filtering result, obtained with the filter $^[A-Z]$ that matches any code starting with an uppercase letter, thus discarding codes like `z1`.

The "Export all text as POS list" button can be used to export this table display of the whole text automaton as a text file following a special format. Currently, it is only an experimental feature that may change in the future. Here is an example of output:

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,maitre.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 7.34: Filtered table display

(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}

7.9 The special case of Korean

Korean is an agglutinative language that has a very special morphological system: words are made of Hangul syllabic characters, but one Hangul character corresponds to several Jamo alphabetic characters. For instance, you can see on Figure 7.35 two examples of Hangul characters followed by their equivalent Jamo letter sequences.

능	생
ㄴ ㅡ ㅇ	ㅅ ㅅ ㅣ ㅇ

Figure 7.35: Hangul characters and their equivalent Jamo sequences

Moreover, morphemes do not correspond necessarily to Hangul characters. For instance, Figure 7.36 shows that a given token (shown in green) must be analyzed as a combination of two elements: a verb and a modifier. The point is that the modifier is only made of one Jamo letter that combines with the last Hangul character of the verb in order to give the last Hangul character of the whole word (in green). The green tokens correspond to untagged

tokens. Untagged tokens are not highlighted in green for other languages.

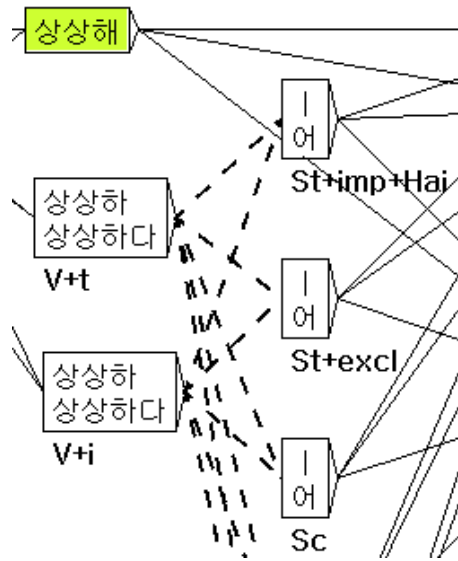


Figure 7.36: Decomposition of a Hangul character

As a consequence, it can be convenient for Korean users to write grammars with mixes of Hangul and Jamo characters. Thus, a grammar like the one shown on Figure 7.37 will match sequences like the one shown Figure 7.38.

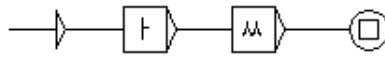


Figure 7.37: A grammar with two Jamo letters

REMARKS:

1. Jamo letters are not in the Korean alphabet file (`Alphabet.txt`). DO NOT ADD THEM TO THIS FILE, because it would induce dysfunctions in programs.
2. This alphabet file contains equivalences between some Chinese characters and some Hangul ones. In practice, if a grammar contains a Chinese character that has such an equivalent Hangul, it will match this Hangul in the text automaton. For instance, the grammar shown on Figure 7.39 will match the sentence of Figure 7.38, because the Korean alphabet file contains an equivalence for that character, as shown on Figure 7.40.

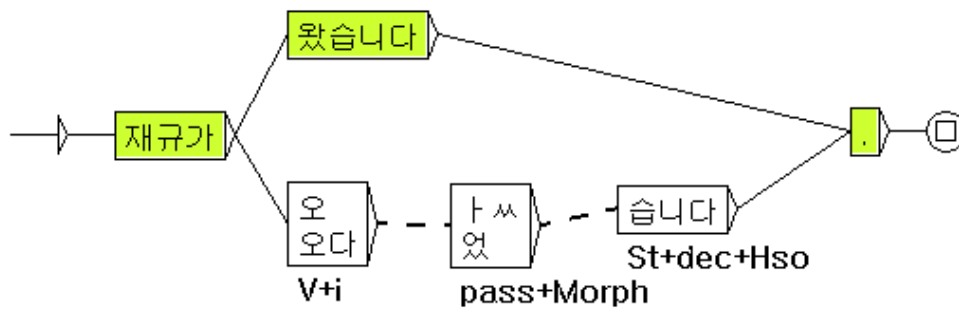


Figure 7.38: Sentence automaton matched by grammar of Figure 7.37



Figure 7.39: A grammar with a Chinese character

伍오
 悟오
 傲오
 午오
 棼오

Figure 7.40: Extract of Korean alphabet file

Chapter 8

Lexicon-grammar

The tables of lexicon-grammar are a compact way for representing syntactical properties of the elements of a language. It is possible to automatically construct local grammars from such tables, due to a mechanism of parameterized graphs.

In the first part of the chapter the formalism of tables is presented. The second part describes parameterized graphs and a mechanism of automatically lexicalizing them with lexicon-grammar tables.

8.1 Lexicon-grammar tables

Lexicon-grammar is a methodology developed by Maurice Gross and the LADL team ([9], [10], [38], [51], [49], [50], [48], [47], [44], [43], [42], [41], [40], [64], [83]) based on the following principle: every verb has an almost unique set of syntactical properties. Due to this fact, these properties need to be systematically described, since it is impossible to predict the exact behavior of a verb. These descriptions are represented by matrices where rows correspond to verbs and columns to syntactical properties. The considered properties are formal properties such as the number and nature of allowed complements of the verb and the different transformations the verb can undergo (passivization, nominalisation, extraposition, etc.). The matrices, or tables, are mostly binary: a + sign occurs at the intersection of a row and a column of a property if the verb has that property, a – sign if not. More information in <http://infolingu.univ-mlv.fr>, including some lexicon-grammar tables that you can freely download.

This type of description has also been applied to adjectives ([67]), predicative nouns ([33], [34], [32], [39], [80]), adverbs ([45], [69]), as well as frozen expressions, in many languages ([14], [26], [27], [73], [74], [77], [87], [88], [89], [81], [78], [46]).

Figure 8.1 shows an example of a lexicon-grammar table. The table contains verbs that, among other definitional properties, do not admit passivization.

	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 = Nnum	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 o	<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\$affecté\$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

Figure 8.1: Lexicon-grammar Table 32NM

8.2 Conversion of a table into graphs

8.2.1 Principle of parameterized graphs

The conversion of a table into graphs is carried out by a mechanism involving parameterized graphs. The principle is the following: a graph that describes the possible constructions is constructed manually. That graph refers to the columns of the table in the form of parameters or variables. Afterwards, for each line of the table a copy of this graph is constructed where the variables are replaced with the contents of the cell at the intersection of line and the column that corresponds to the variable. If a cell of the table contains the + sign, the corresponding variable is replaced by <E>. If the cell contains the - sign, the box containing the corresponding variable is removed, interrupting the paths through that box. In all other cases the variable is replaced by the contents of the cell.

8.2.2 Format of the table

The lexicon-grammar tables are usually encoded with the aid of a spreadsheet like OpenOffice.org Calc ([72]). To make them usable with Unitex, the tables have to be encoded in Unicode text format in accordance with the following convention: the columns need to be

separated by a tab and the lines by a newline.

In order to convert a table with OpenOffice.org Calc, save it in text format (.csv extension). You can then parameterize the output format with a window as shown on Figure 8.2. Choose "Unicode", select tabulation as column separator and do not set any text delimiter.

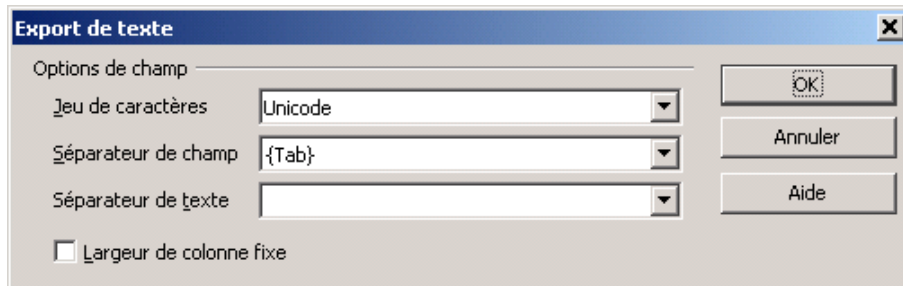


Figure 8.2: Saving a table with OpenOffice.org Calc

During the generation of the graphs, Unitex skips the first line, considering that it contains the headings of the columns. It is therefore necessary to ensure that the headings of the columns occupy exactly one line. If there is no line for the heading, the first line of a table will be ignored anyway, and if there are multiple heading lines, from the second line on they will be interpreted as lines of the table.

8.2.3 Parameterized graphs

Parameterized graphs are graphs with variables referring to the columns of a lexicon-grammar table. This mechanism is usually used with syntactical graphs, but nothing prevents the construction of parameterized graphs for inflection, preprocessing, or for normalization.

Variables that refer to columns are formed with the @ symbol followed by the name of the column in capital letters (the columns are named starting with A).

Example: @C refers to the third column of the table.

Whenever a variable takes the value of a + or - sign, the - sign corresponds to the removal of a path through that variable. It is possible to swap the meaning of these signs by typing an exclamation mark in front of the @ symbol. In that case, the path is removed when there is a + sign and kept where there is a - one. In all other cases, the variable is replaced by the content of the table cell.

The special variable @% is replaced by the number of the line in the table. The fact that its value is different for each line allows for its use as a simple characterization of a line. That variable is not affected by an exclamation point to the left of it.

Figure 8.3 shows an example of a parameterized graph designed to be applied to the lexicon-grammar table 31H presented in figure 8.4.

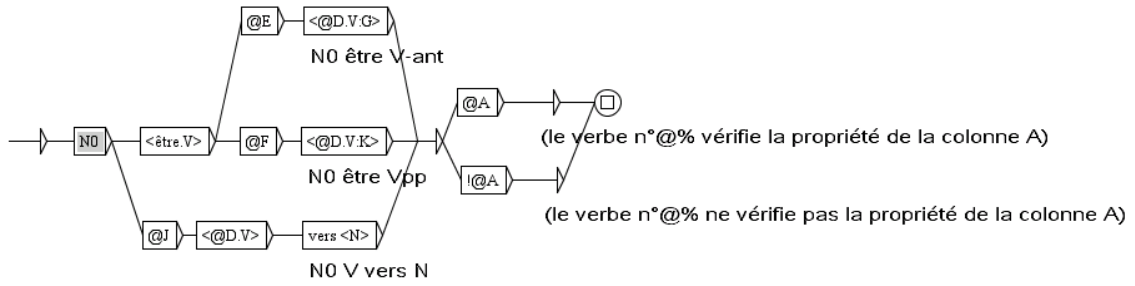


Figure 8.3: Example of parameterized graph

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
1	NO =: N-hum	NO =: V-n	Aux =: avoir	<ENT>	NO est V-ant	NO est Vpp	NOpc lui V	NO V de NOpc	Nhum V sur ce point	NO V vers N	il V NO W	idée Loc esprit	Nhum Loc Nabs	<OPT>NO =: V-n	<OPT>E
2	-	-	+	abandonner	-	-	-	-	-	-	-	-	-	-	Paul a§abandonné§
3	-	-	+	abuser	-	-	-	-	+	-	-	-	-	-	Max§abuse§
4	-	-	+	acquiescer	-	-	-	+	+	-	-	-	-	-	Max a§acquiescé§(E+de
5	-	-	+	adouber	-	-	-	-	-	-	-	-	-	-	Paul§adoube§ écheçs
6	-	-	+	agioter	-	-	-	-	-	-	+	-	-	-	Max§agioté§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 8.4: Lexicon-grammar table 31H

8.2.4 Automatic generation of graphs

In order to be able to generate graphs from a parameterized graph and a table, first of all the table must be opened by clicking on "Open..." in the "Lexicon-Grammar" menu (see figure 8.5). The table must be in Unicode text format.

The selected table is then displayed in a window (see figure figure 8.6). If it does not appear

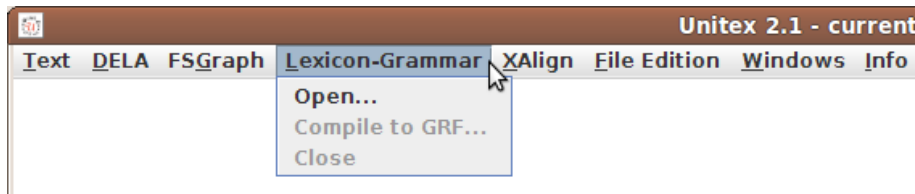


Figure 8.5: Menu "Lexicon-Grammar"

on your screen, it may be hidden by other Unitex windows.

NO =: N-hum	NO =: V-n	Aux =: avoir	<ENT>	NO est V-ant	NO est Vpp	NOpc lui V	NO V de NOpcNhum V sur...	Ni
-	-	+	abando...	-	-	-	-	-
-	-	+	abuser	-	-	-	+	-
-	-	+	acquie...	-	-	-	+	-
-	-	+	adouber	-	-	-	-	-
-	-	+	agioter	-	-	-	-	-
+	-	+	agoniser	+	-	-	-	-
-	-	+	archaiser	+	-	-	+	-
-	-	+	arquer	-	-	-	+	+
-	-	-	arriver	-	+	-	-	-
-	-	+	atermoyer	-	-	-	+	-
-	+	+	badauder	-	-	-	-	+
+	-	+	baisser	-	-	-	+	-
-	-	+	bambocher	-	-	-	-	-
+	-	+	bander	-	-	-	+	-

Figure 8.6: Displaying a table

To automatically generate graphs from a parameterized graph, click on "Compile to GRF..." in the "Lexicon-Grammar" menu. The window in figure 8.7 shows this.

In the "Reference Graph (in GRF format)" frame, indicate the name of the parameterized graph to be used. In the "Resulting GRF grammar" frame, indicate the name of the main graph that will be generated. This main graph is a graph that invokes all the graphs that are going to be generated. When launching a search in a text with that graph, all the generated graphs are simultaneously applied.

The "Name of produced subgraphs" frame is used to set the name of each graph that will be generated. Enter a name containing @%, because for each line of the table, @% will be replaced the line number, which guarantees that each graph name will be unique. For example, if the main graph is called "TestGraph.grf" and if subgraphs are called "TestGraph_@%.grf", the graph generated from the 16th line of the line will be named "TestGraph_0016.grf".

Figures 8.8 and 8.9 show two graphs generated by applying the parameterized graph of figure 8.3 at table 31H.

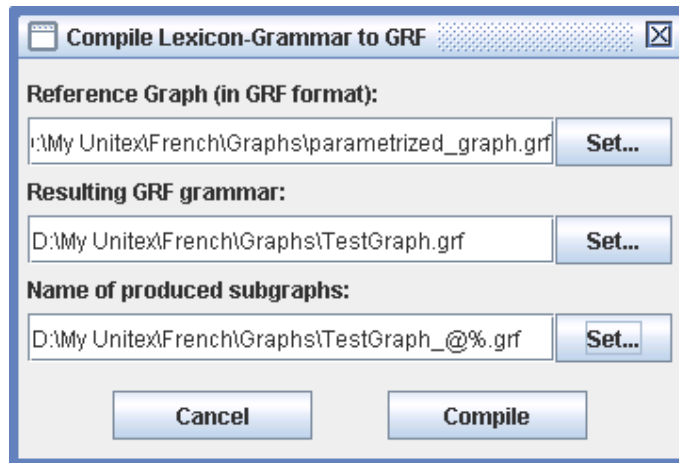
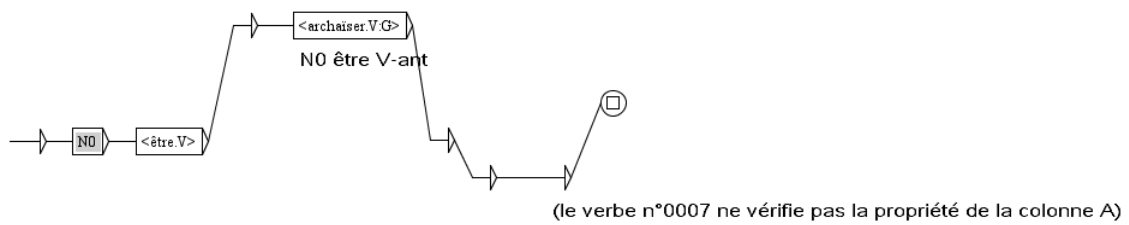
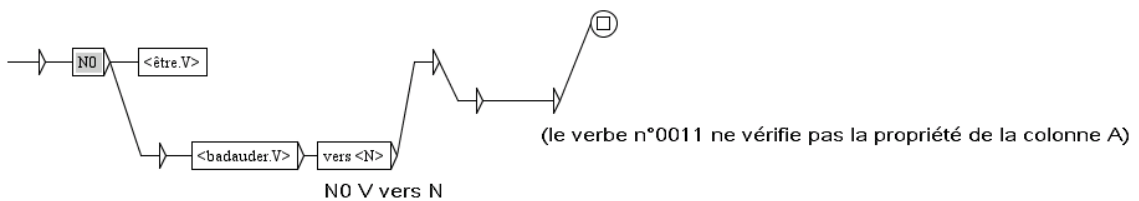


Figure 8.7: Configuration of the automatic generation of graphs

Figure 8.10 shows the resulting main graph.

Figure 8.8: Graph generated for the verb *archaïser*Figure 8.9: Graph generated for the verb *badauder*

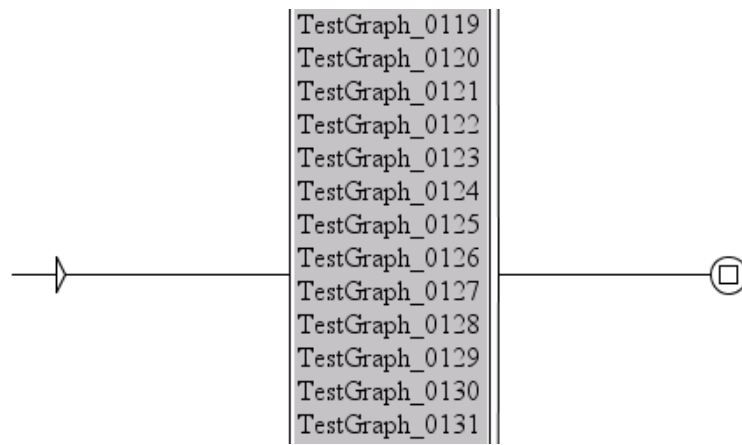


Figure 8.10: Main graph referring to all the generated graphs

Chapter 9

Text alignment

The principle of text alignment is simple: aligning two (or more) texts, one supposed to be the source, and the other(s) supposed to be its translation(s). The alignment is made at the sentence level, because word alignment is not possible yet, and certainly not relevant. Then, one can look for an expression *A* in one of the texts and look for its translations in the sentences aligned with those containing occurrences of *A*.

To include such a functionality into Unitex, Patrick Watrin integrated the Open Source text alignment tool XAlign, developed at the LORIA ([66]). In this chapter, we will explain how to use the alignment module. The reader interested in details about the integration of XAlign can consult [23] or [75], and [91] for an illustration of what can be done with this module.

9.1 Loading texts

First, you need to select your 2 texts. To do that, go into "XAlign>Open files...", and you will see the frame shown on Figure 9.1. You provide texts under two formats: raw unicode text (as you do for your corpus) or TEI-encoded texts (an XML format; see [54]). In the last text field, you can select a XML alignment file, if you have already built one. If you select a raw text, Unitex will need to build a basic TEI version of it (for more details, see section 12.45 about the XMLizer program). So, when you click on "OK", you will be asked to provide a XML file name as shown on Figure 9.2. Then, Unitex builds the XML versions of your texts, if needed, and displays the frame shown on Figure 9.3. As you can see, each text is presented as a list, each cell representing a sentence.

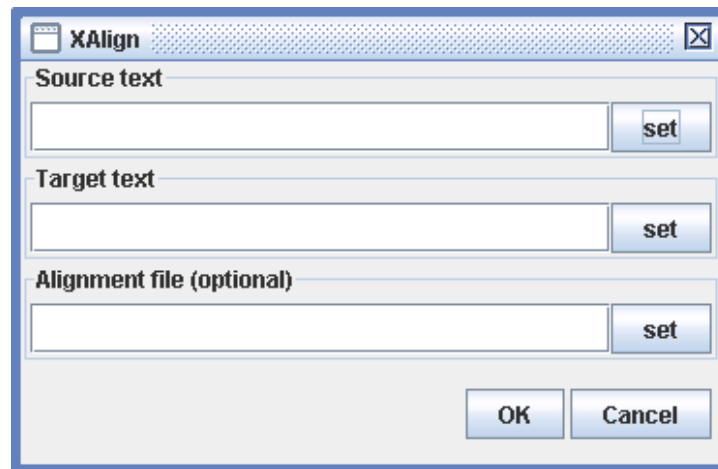


Figure 9.1: Text alignment selection frame

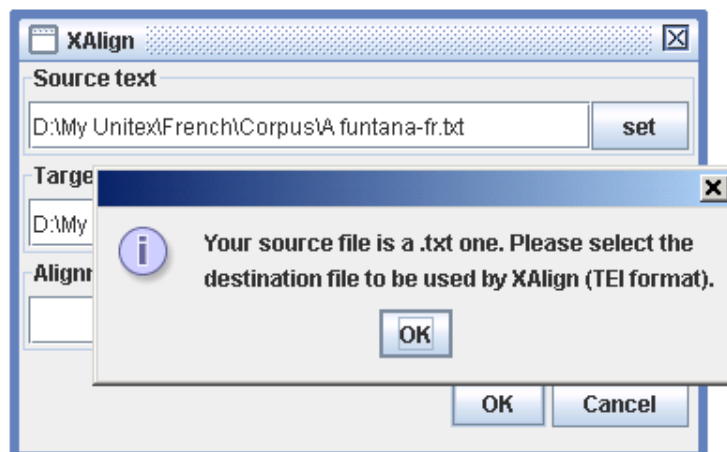


Figure 9.2: Warning about raw texts

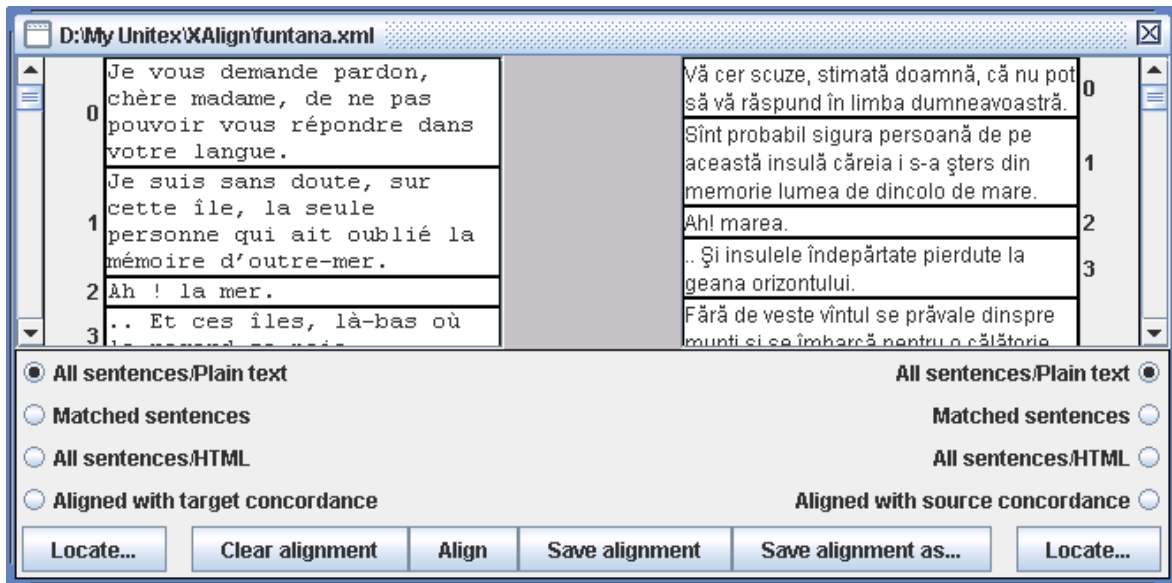


Figure 9.3: Text alignment frame

9.2 Aligning texts

Once you have loaded your texts, you can align them by clicking on the "Align" button. You will be asked to provide the name of the XML file that will contain all the information about the alignment. Then, Unitex launches the XAlign program and you will visualize the alignment under the form of red links between aligned sentences, as shown on Figure 9.4.

You can edit the alignment links with the mouse. Clicking on a link removes it. To add a link (or remove it, if it already exists), click on one sentence (in the text you want, source or destination), and then move your mouse over the corresponding sentence in the other text. The link about to be added will appear in yellow, as shown on Figure 9.5. When you click, the link is actually added and becomes red. When you have made all your corrections, you can save your modified alignment using the "Save alignment" and "Save alignment as..." buttons.

An interesting feature of XAlign is that it is *reentrant*. It means that you can take an existing alignment as a set of mandatory links in input of the alignment process. This can be useful if you want to work with *cognates*. For more details about cognates and XAlign, see discussion in [75].

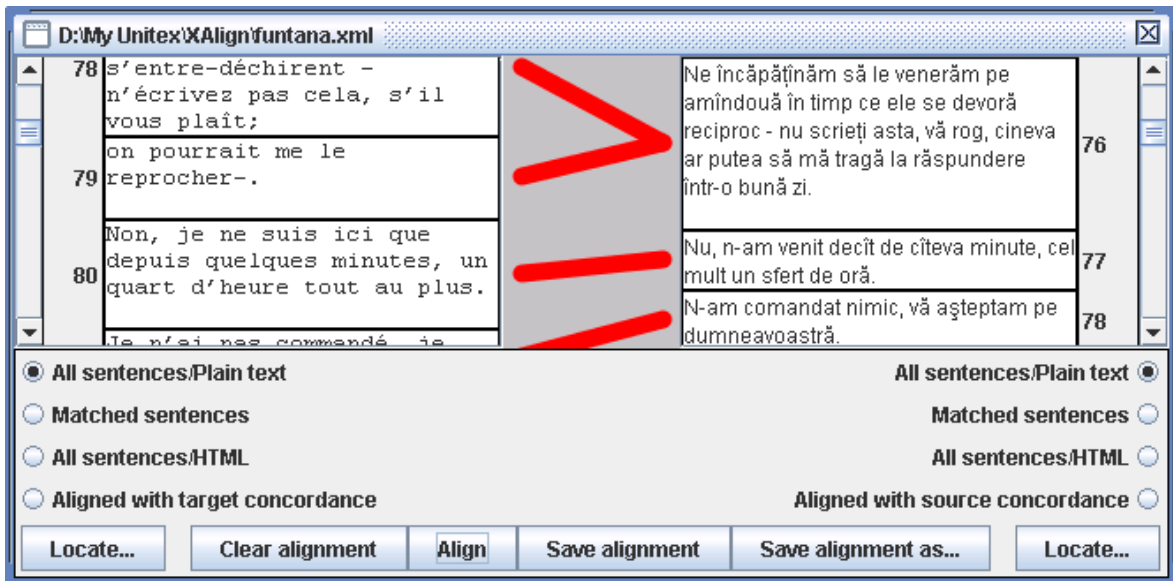


Figure 9.4: Aligned sentences

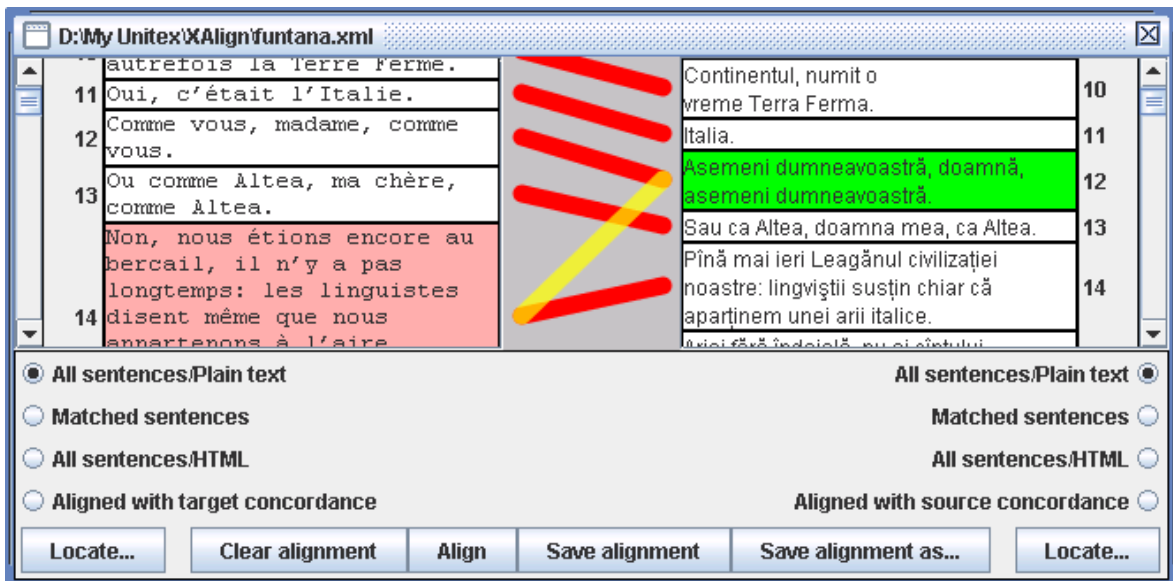


Figure 9.5: Adding a link

9.3 Pattern matching

You can perform pattern matching queries on any of your texts, by clicking on its "Locate" button. The first time you click, Unitex will ask you to build a working version of your text, as shown on Figure 9.6. This text version will be preprocessed according to the text language (in particular, the default dictionaries will be applied).

WARNING: the text language is determined on the basis of the path name. For instance, if your text file is located in `.../MyUnitex/Klingon/Corpus`, the language will be considered to be `Klingon`. So, if your text is not in a subdirectory of your personal Unitex directory, its language will not be identified.

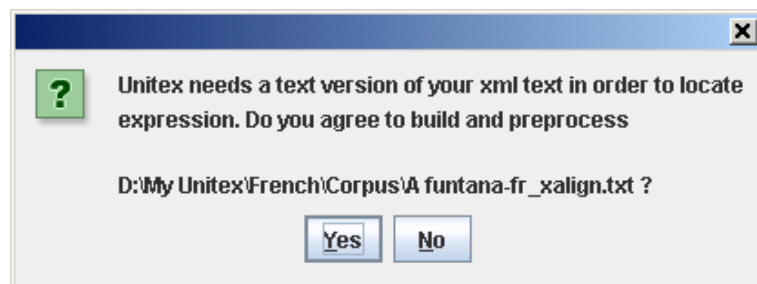


Figure 9.6: Unitex needs to build a working version of your text

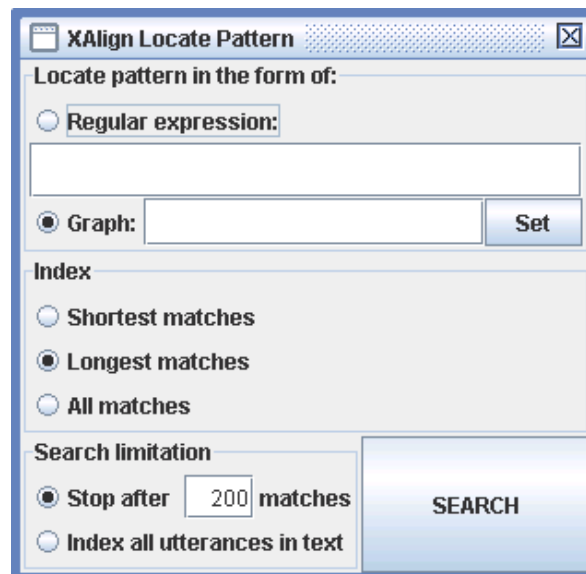


Figure 9.7: Pattern matching frame for aligned texts

Once Unitex has created and preprocessed the working version of the text, you can perform

your query using the frame shown on Figure 9.7. As the matching operation is performed by the `Locate` program, you can perform the same queries than you would perform on a normal corpus. The only restriction is that you cannot exploit the outputs of your grammars, if any.

For instance, let us lookup for the pattern `<manger>` (*to eat*) in the French text of our example. First, we see no result, because we have not changed yet the display mode for the French text, which by default is "All sentences/Plain text". Clicking on "Matched sentences", we only see sentences that contain occurrences, highlighted as usual in blue, as shown on Figure 9.8. Clicking on "All sentences/HTML" will display all sentences, highlighting occurrences in blue.

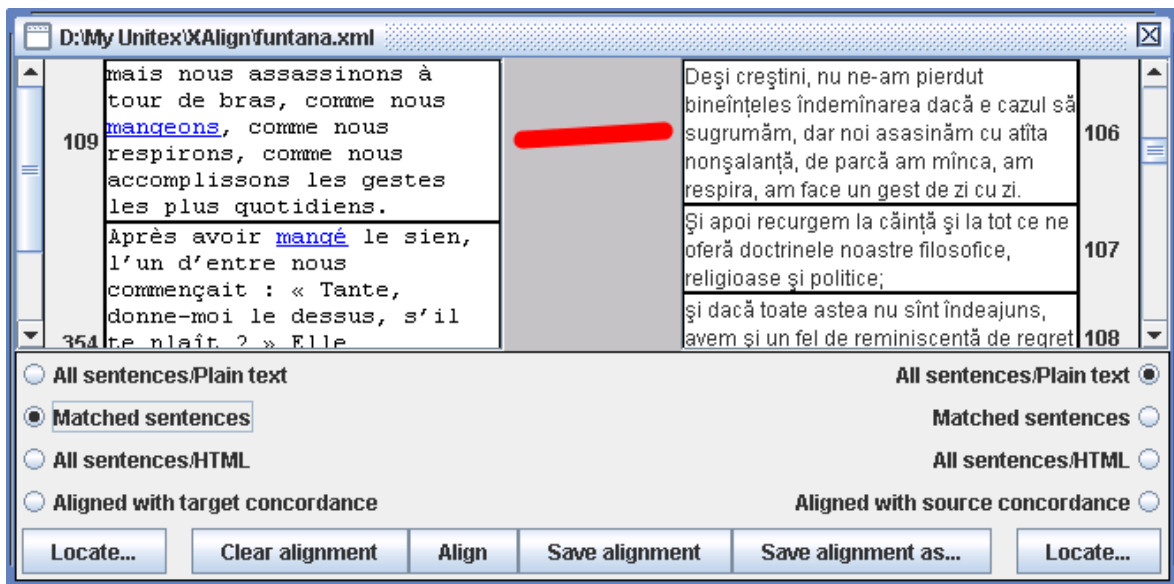


Figure 9.8: Displaying matched sentences

To exploit parallel texts, it is then interesting to retrieve sentences aligned with matched sentences. This can be done by selecting *for the other text*, the display mode "Aligned with source concordance". In this mode, Unitex filters sentences that are not linked to matched sentences in the source text. So, it is easy to lookup for an expression in one text and to find the corresponding sentences in the other, as shown on Figure 9.9.

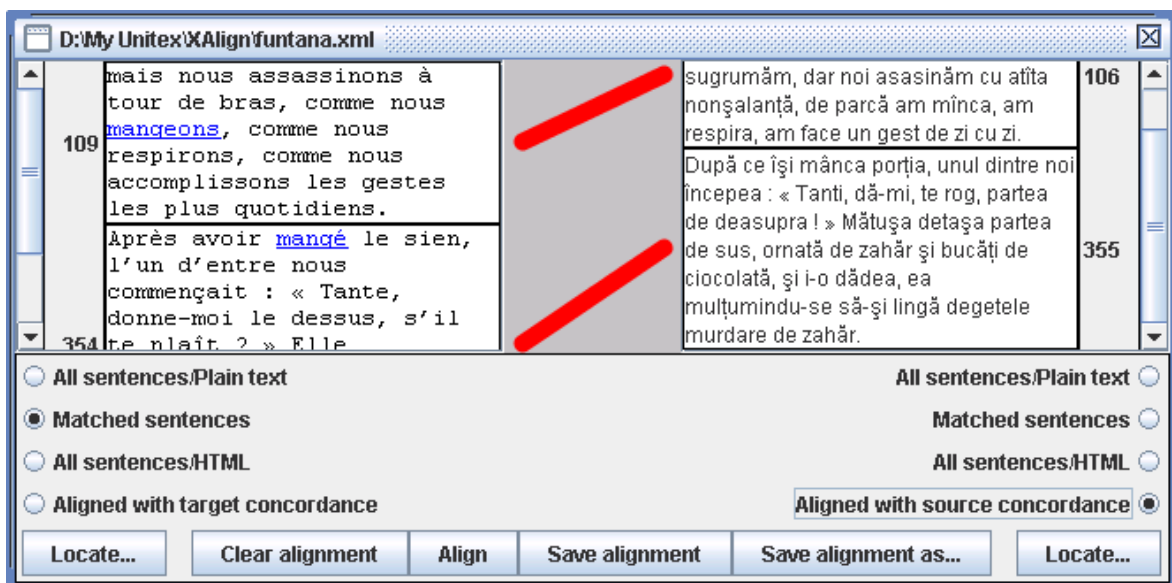


Figure 9.9: Displaying matched sentences and sentences they are linked to

Chapter 10

Compound word inflection

MULTIFLEX is a multi-lingual Unicode-compatible platform for automatic inflection of *multi-word units* (MWUs), also known as *compound words*. It is meant in particular for the creation of morphological dictionaries of MWUs. It implements a unification-based formalism ([85]) for the description of inflectional behavior of MWUs which supposes the existence of a module for the inflectional morphology of simple words.

In this chapter, we present the notion of multi-word unit and we describe the method to inflect them with MULTIFLEX.

This chapter is derived from the MULTIFLEX manual, written by Agata Savary, the author of MULTIFLEX.

10.1 Multi-Word Units

Multi-word units (MWUs) encompass a bunch of hard-to-define and controversial linguistic objects (cf. [52], [18]). Their numerous linguistic and pragmatic definitions ([5], [22], [65], [4], [36], [3], [86], [37], [13]) invoke three major points:

- they are composed of two or more words
- they show some degree of morphological, distributional or semantic non-compositionality
- they have unique and constant references

However, the basic notions (a word, a reference, the non-compositionality) and measures (degree of non-compositionality), used in those definitions are themselves controversial.

Pragmatically, we consider a MWU as a contiguous sequence of *graphical units* which, for some application-dependent reasons, has to be listed, described (morphologically, syntactically, semantically, etc.) and processed as a unit.

10.1.1 Formal Description of the Inflectional Behavior of Multi-word Units

The main issue in MULTIFLEX is the inflectional morphology of MWUs. This phenomenon has been linguistically analyzed for English, Polish and French in [84].

Obviously, a reliable inflection processing of single words is a necessary condition for the inflection processing of MWUs. However, this condition is rarely a sufficient one. For example, in order to obtain the plural form of

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

in English, not only do we need to know how to generate the plural of *battle*, *royal* and *cry*, but also to know how different inflected forms of these constituents combine:

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

but not

- * *battles cries*
- * *battles royals*
- * *battles of nerve_*

Formally, a fully explicit description of the inflectional paradigms of MWUs requires an answer to the following questions:

- What is the MWU's morphological class (noun, adjective, etc.) and thus what inflection categories (number, gender, case, etc.) are relevant to it? [76] argue for a morphosyntactically motivated definition of morphological classes: a morphological class should fully determine the inflection categories the word inflects for as well as those that are lexically fixed for the word, e.g. in Polish, a noun has a gender and inflects for number and case.
- What are the exceptions to the inflection categories determined above? E.g. in Polish
 - *wybory powszechne*
(general election)

is a compound noun but it doesn't have a singular form (although its head word *wybory* does).

- What are the inflectional characteristics (base form, morphological class, inflection paradigm, etc.) of the single constituents of the MWU? E.g. in French, *porte* (door) is an uninflected verb in

– *porte-avion*
(aircraft carrier)

while it is an inflected noun in

– *porte-fenêtre*
(French window)

which takes an *s* in plural

– *portes-fenêtres*

- How should we combine the inflected forms of the single constituents in order to generate the inflected forms of the whole compound? E.g. to inflect *battle of nerves* and *battle cry* in number we need to inflect the first and the last constituent, respectively.

10.1.2 Lexicalized vs. Grammar-Based Approach to Morphological Description

A previous study ([84]) has confirmed the status of MWUs as units on the frontier between morphology and syntax. Their compound structure suggests productivity which can hardly be processed without a grammar-based approach. However some of their morphological, syntactic and semantic properties exclude their processing merely in terms of the properties of their constituents. For example, in both examples below:

- *chief justice*
- *lord justice*

there are few automatically accessible hints indicating that the former one is morphologically a standard English *Noun Noun* phrase taking an *s* at its last constituent in plural, while the plural of the latter has three variants:

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

Thus, at least one of the above examples has to be considered as lexicalized in order for the automatic morphological processing to be reliable.

MULTIFLEX implements a unification-based formalism for the description of the inflectional behavior of MWUs presented in [85]. Its features are described in section 10.2. This formalism requires the description to be *fully* lexicalized: each MWU listed in a dictionary

obtains a code (e.g. *NC_NN*, *NC_NN2*, etc.) representing its inflectional paradigm, for instance, in the DELA-like format:

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

However, only a few codes, which can be seen as a phrase grammar of the language, represent the big majority of all MWUs. Thus, the lexicalization of the description mainly consists of pointing out the MWUs which respect or don't respect the "grammar".

10.2 Formalism for the Computational Morphology of MWUs

In [85] was proposed a formalism for describing the morphological paradigms of MWUs. It has been based on studies of English, Polish and French, and further tested for Serbian [57]. It consists of a language-independent kernel which is to be completed by a set of morphological elements characteristic for the given language. In this section we give an in-depth description of this formalism.

10.2.1 Morphological Features of the Language

When processing MWUs of a given language we have to provide some general data about that language. These data are included in two textual files.

The `Morphology.txt` file gives the morphological classes (noun, adjective,...), categories (number, gender, case,...) and values (masculine, feminine, singular, nominative,...). Consider the following example:

```
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:
```

The above file says that, for Polish, three inflection categories are considered: the number (*Nb*), the case (*Case*) and the gender (*Gen*). Each category is given an exhaustive list of its possible values (singular and plural for number, etc.). Further, each morphological class is described with respect to the categories it inflects for, and those that are fixed for it. For example, a noun inflects for number and case, and has a (fixed) gender. The presence of

such a file is necessary if we wish to express the fact that a certain word inflects for number, gender or case, without having to explicitly enumerate each time which inflectional values (singular, plural, masculine, etc.) it can take.

Similarly, for French the `Morphology.txt` file may be as follows:

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

However, in the existing systems for computational morphology, such a description of classes, categories and values is not always present. For example, according to the DELA conventions ([20]) the morphological values of each simple word are plain sequences of characters (e.g. *ms* for masculine singular) without any explicit mention of their corresponding categories. In order for the program to be compatible with such systems, we use a list (contained in a file called `Equivalences.txt`) that describes which foreign inflectional feature corresponds to which category-value pair in our description. For example, the following lists:

<i>Polish</i>	<i>French</i>
<i>s : Nb = sing</i>	<i>s : Nb = s</i>
<i>p : Nb = pl</i>	<i>p : Nb = p</i>
<i>M : Case = Nom</i>	<i>f : Gen = f</i>
<i>D : Case = Gen</i>	<i>m : Gen = m</i>
<i>C : Case = Dat</i>	
<i>B : Case = Acc</i>	
<i>I : Case = Inst</i>	
<i>L : Case = Loc</i>	
<i>V : Case = Voc</i>	
<i>o : Gen = masc_pers</i>	
<i>z : Gen = masc_anim</i>	
<i>r : Gen = masc_inanim</i>	
<i>f : Gen = fem</i>	
<i>n : Gen = neu</i>	

describe the equivalences between the previous `Morphology.txt` file for Polish and French, respectively, and the single-character features that might be used in DELA dictionaries for those languages under Unitex.

10.2.2 Decomposition of a MWU into Units

The notion of an elementary graphical unit is controversial and varies across languages and NLP systems. For instance in nitex an alphabet, i.e. a set of characters, is first defined for each language. Each non alphabet character is called a separator. A graphical unit is then either a single separator (usually a punctuation mark, a digit, etc.) or a contiguous sequence of alphabet characters (e.g. *aujourd'hui* in French consists, according to this definition, of 3 units). In other systems a graphical unit may contain a punctuation mark (e.g. *c'est-à-dire*), or a limit between two graphical units may occur within a sequence of alphabet characters (*widział|bym*, cf [76]).

This variety of possible definitions of a graphical unit obviously has an impact on the definition of a multi-word unit. However, we wish our formalism for MWUs to be adaptable to different morphological systems for “simple words”. Thus, the definition of a graphical unit is a parameter to our system: each time MULTIFLEX is used with an external module for single units, this module has to decide how a sequence of characters is to be divided into units.

In our formalism, units are referred to by numerical variables \$1, \$2, \$3, etc. For example with Unitex, a sequence like

- *Athens '04*

consists of five constituents referred to in MULTIFLEX as:

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

Each simple unit subject to inflection within a MWU has to be morphologically identified. The identification means providing sufficient data so that any inflected form of the same item may be generated on demand. For instance in:

- *mémoire vive*

we need to know that *vive* is the feminine singular form of a lemma, and we have to be able to generate the feminine plural form of the same lemma, *vives*. We suppose that the external module for single units working with MULTIFLEX is responsible for such identification and generation of inflected forms of single units.

In Unitex, the generation of forms is strongly inspired by the DELA system ([20]). In order to be able to generate one or more inflected forms of a word we have to know:

- its lemma

- its inflection paradigm (called inflection code)
- the inflection features of forms to be generated

Thus, within the Unitex/MULTIFLEX interface the description of a single unit is done as follows:

- *vive(vif.A54:fs)*

where *A54* is the inflection code of *vif* and *fs* is the DELA-style description using morphological features appearing in `Equivalences.txt` file (cf section 10.2.1). Knowing that *vive* is a feminine singular form of *vif* we may demand the generation of its plural without having to explicitly indicate the plural of which gender we are interested in: since we only wish to change the number, the gender remains as in the original word *vive*, i.e. feminine.

10.2.3 Inflection paradigm of a MWU

The morphological description of MWUs in our formalism is inspired by the DELA system in the sense that:

- each MWU is attributed an inflection code
- a MWU's inflection code explicitly describes each inflected form of a MWU in terms of actions to be performed on the lemma, and inflectional features to be attached to each form

In the Unitex-interfaced version, MULTIFLEX uses inflection codes represented as Unitex graphs compiled into the `.fst2` format. For example, Figure 10.1 contains the inflection graph for *battle royal*.

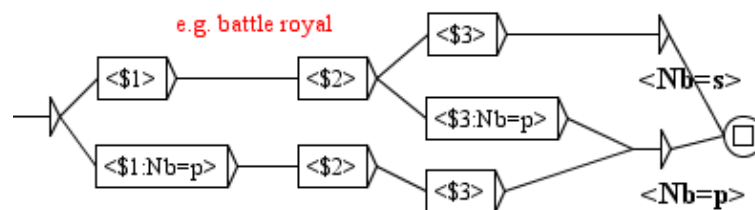


Figure 10.1: Inflection graph for *battle royal*

According to the Unitex convention, three constituents are present in *battle royal*: *battle* referred to as *\$1*, a space referred to as *\$2*, and *royal* referred to as *\$3*. If a variable appears alone in a box the constituent has to be the same as in the lemma of the MWU. For instance, *<\$3>* in the uppermost path means that the unit *royal* is to be recopied as such. If the variable is

accompanied by a set of category-feature equations, the constituent has to be inflected to the required form. E.g. $\langle \$3:Nb=p \rangle$ means that the plural form of *royal* is needed.

In order to generate all inflected forms of the MWU we have to explore all the paths existing in the graph. Each path starts at the leftmost right arrow and ends at the final encircled box. Each time we come to a node we perform the action contained in the box (a recopy or an inflection of a constituent) and we accumulate the morphological features contained under the box. The total of the accumulated node outputs should result in the complete morphological description of the inflected form.

For example in the graph on Figure 10.1 if we follow the intermediate path shown on Figure 10.2:

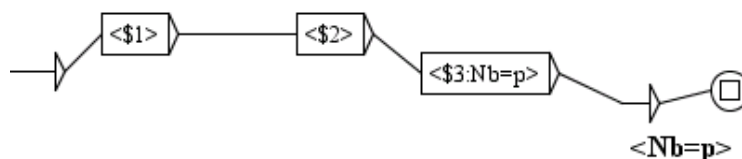


Figure 10.2: One path of the inflection graph for *battle royal*

we recopy *battle* ($\$1$) and the space ($\2), and we put *royal* into plural, which yields the plural form *battle royals* of the whole MWU. As the graph on Figure 10.1 contains three different paths the whole set of inflected forms generated for *battle royal* would be:

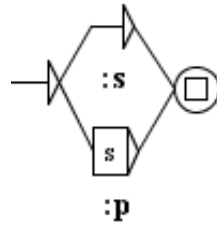
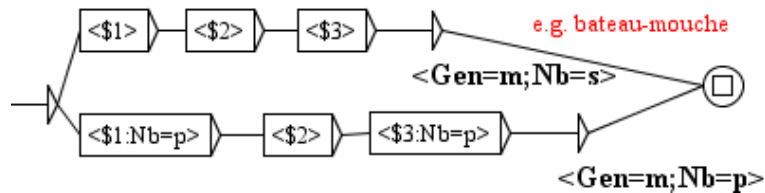
battle royal $\langle Nb=s \rangle$
battle royals $\langle Nb=p \rangle$
battles royal $\langle Nb=p \rangle$

After rewriting these forms into the Unitex DELACF format we obtain the following entries:

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

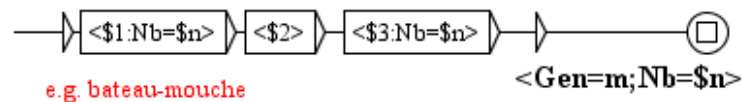
Note that this description is independent of the way we generate inflected forms of single words because we suppose that this problem is handled by an existing external morphological system for single words. In the Unitex-interfaced version of MULTIFLEX, we would generate the plural of *royal* due to the fact that its lemma is known as having the inflection code *N1* represented on Figure 10.3.

In an inflection paradigm of a MWU, each constituent is accompanied only by those morphological categories which it should inflect for. The categories that remain unchanged don't have to be mentioned. For instance, in *bateau-mouche* in French (a Paris-style riverboat), both noun constituents have their gender set but they inflect in number: *bateaux-mouches*. That's why on Figure 10.4 containing the inflection graph for this MWU, the corresponding boxes contain value assignments for number only. Note that both constituents may or may not agree in gender, here *bateau* is masculine while *mouche* is feminine.

Figure 10.3: Inflection graph *N1* for simple words inflecting like *royal*Figure 10.4: Inflection graph for MWUs inflection like *bateau-mouche*

Unification Variables

An important feature of our formalism are *unification variables*. They are introduced by the dollar sign followed by an identifier which may contain any number of characters, e.g. $\$g1$, $\$num_{10}$, $\$c$, etc. For example, Figure 10.5 shows a graph roughly equivalent¹ to the one on Figure 10.4 in the sense that it allows to generate the same inflected forms for the same MWUs. However, this time a single path represents both the singular and the plural form. That is possible due to the unification variable $\$n$ which may be instantiated to any value of the domain of its category (Nb), here $\$n=s$ or $\$n=p$. The instantiation is unique for all elements on a path: if we fix the singular value for the first constituent the same value has to be set for the third one, as well as for the whole MWU. Similarly, if we fix $\$n$ to p while processing the first node it has to remain p until the end of the path.

Figure 10.5: Inflection graph for *bateau-mouche* with a unification variable

The inflection graph on Figure 10.5 applies to most kinds of French compounds of types *Noun Noun* and *Noun Adjective* (*bateau-mouche*, *ange gardien*, *circuit séquentiel*, etc.) which are of masculine gender. That is because the output of the final node contains $Gen=m$. For all compounds of the same types but of feminine gender, e.g. *main courante*, *moissonneuse-batteuse*, etc., a new graph has to be created which is identical to Figure 10.5 up to the final output containing $\langle Gen=f;Nb=\$n \rangle$. That is not very intuitive since *circuit séquentiel* and *main*

¹Up to the case when single constituents appearing in the lemma of a MWU are already in plural, as in *cross-roads*.

courante inflect in the same way, in the sense that in both cases we need to put the first and the last constituent to plural in order to obtain the plural form of the whole MWU.

That's why another type of instantiation for unification variables has been introduced. It is accompanied by a double equal sign ($=$) (as opposed to the single equal sign $=$ as for $\$n$ on Figure 10.5). If a unification variable is assigned to a category by this symbol then it inherits the value of this category from the corresponding constituent, as it appears in the lemma of the MWU. For instance, Figure 10.6 contains a graph describing the inflected forms for both masculine and feminine French compounds of types *Noun Noun* and *Noun Adjective*. Its first box contains the double assignment of the gender to variable $\$g$ which means that this variable has its value fixed to the gender value of the first constituent. For *bateau-mouche* it is fixed to masculine because *bateau* is masculine while for *main courante* it is fixed to feminine.

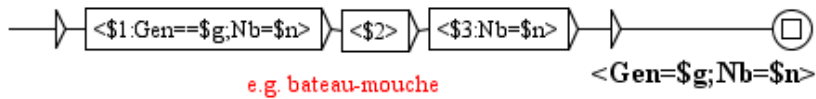
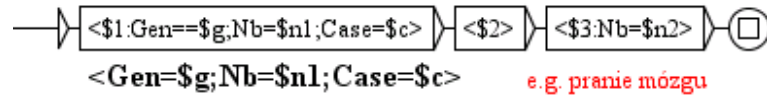


Figure 10.6: Inflection graph for *bateau-mouche* with two types of instantiation

Note that the double assignment, contrary to the single assignment, no longer means that the variable is to be instantiated to all values of the corresponding category domain. It has a unique value all through the path on which it appears, even if it is concerned by another, single, assignment somewhere else on the same path. For example, on Figure 10.6 the final output contains $Gen=\$g$ but $\$g$ may only take one value determined by the first constituent.

Unification variables are particularly useful in highly inflected languages. For example, in Polish most nouns inflect for number (2 values) and case (7 values), which implies at least 14 different forms (if variants and syncretic forms are distinguished). This score is even higher for adjectives which inflect for number, case and gender (3 till 9 values, according to different approaches). If no unification mechanism were available each of these numerous forms would have to be described by a separate path in the graph. The use of unification variables allows to dramatically reduce the size of the graph (to one path only in most cases).

For example, Figure 10.7 shows the graph for Polish compounds that inflect like *pranie mózgu* (*brainwashing*) or *powożenie koniem* (*horse coaching*). Their third constituent has its case fixed (most often to genitive or instrumental). Their first and third constituent inflect in number independently from each other (*pranie mózgow*, *prania mózgu*, *prania mózgow*, etc.). That's why either of them has a different unification variable for number inflection ($\$n1$ and $\$n2$). The three variables $\$n1$, $\$n2$, and $\$c$ may be instantiated to any value from their respective domains ($\{sing,pl\}$, $\{sing,pl\}$, and $\{Nom,Gen,Dat,Acc,Inst,Loc,Voc\}$; cf `MORPHOLOGY.txt` file in section 10.2.1). The whole MWU inherits its gender, number and case from its first constituent. Its gender is fixed ($Gen==\$g$) while its number and case are instantiated to any of the 14 possible combinations. The single path in this graph would have to be replaced by 28 different ones if the use of unification variables were not allowed.

Figure 10.7: Inflection graph for *pranie mózgu*

Orthographic and Other Variants

Our formalism allows for any constituent to be omitted or moved within different inflected forms if there is a need for that. It also enables the insertion of extra graphical units which do not appear in the base form of the MWU. This allows to extend an inflection paradigm to a more general variation description, e.g. orthographic or, partly, syntactic variation (see [55] for an extensive study on term variation). For example, in English, *student union* appears in corpus also as *students union*, and *students' union*, in singular or plural in each case. Our formalism allows to include both types of variation in one description (cf. Figure 10.8).

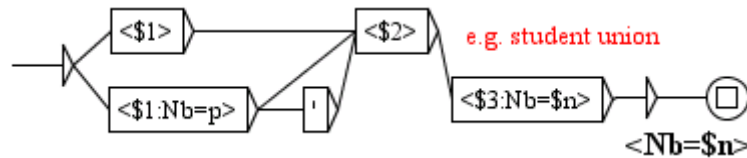
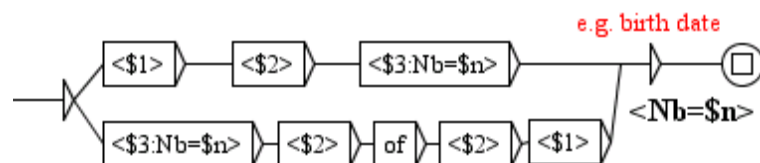
Figure 10.8: Inflection graph for *student union*

Figure 10.9 shows an example in which, additionally to the insertion of a new constituent, the order of constituents may be reverted. The upper path allows to generate e.g. *birth date* and *birth dates* while the lower one represents the syntactic variants of the previous forms: *date of birth* and *dates of birth*.

Figure 10.9: Inflection graph for *birth date*

Interface with the Morphological System for Simple Words

MULTIFLEX is an implementation of the formalism for the inflectional morphology of MWUs presented above. It supposes the existence of a morphological system for single words which satisfies the following interface constraints:

- For a given sequence of characters it returns its segmentation into indivisible graphical units (tokens) (cf section 10.2.2). For instance, in case of Unitex' definition of a token,

sequence *Athens '04* is to be divided into 5 tokens:

"Athens '04" → ("*Athens*", " ", "'", "0", "4")

- For a given simple inflected form it returns all its possible morphological identifications. A morphological identification has to allow the generation of any other inflected form of the same lemma on demand by the same morphological module. For instance, in case of Unitex, the form *porte* yields 7 morphological identifications (6 of which are factorized with respect to their inflection code):

porte → ((*porte,porte.N21:s*),(*porte,porter.V3:P1s:P3s:S1s:S3s:Y2s*))

In case of ambiguity, as above, the proper identification has to be done, for the time being, by the user during the edition of the MWU lemma to be inflected (in future, this task will be partly automated). For instance, in case of *porte-fenêtre* the first constituent has to be identified by the user as a noun rather than a verb.

- For a given morphological identification and a set of inflectional values it returns all corresponding inflected forms. For instance, in Polish, if the instrumental forms of the word *ręka* are to be produced, three forms should be returned: *ręka* (singular instrumental), *rękami* and *rękoma* (two variants of the plural instrumental).

(*ręka*,<Case=Inst>) → ((*ręka*,<Nb=sing;Gen=fem;Case=Inst>),
(*rękami*,<Nb=pl;Gen=fem;Case=Inst>),
(*rękoma*,<Nb=pl;Gen=fem;Case=Inst>))

Such definition of an interface between the morphological system for simple words and the one for MWUs allows a better modularity and independence of one another. The latter doesn't need to know how inflected forms of simple words are described, analyzed and generated. It only requires a set of correct inflected forms of a MWU's constituents. Conversely, the former system knows nothing about how the latter one combines the provided forms to produce multi-word sequences.

10.3 Integration in Unitex

One of the major design principles of MULTIFLEX is to be as independent as possible of the morphological system for simple words. However, the existence of such a system is inevitable because MWUs consist of simple words which we need to be able to inflect in order to inflect a MWU as a whole.

In its present version, MULTIFLEX relies on the Unitex simple word inflection system:

- MULTIFLEX uses the same character encoding standards as Unitex, i.e. Unicode 3.0.
- MULTIFLEX uses the Unitex' graph editor for the representation of inflectional paradigms of MWUs.

- MULTIFLEX admits similar principles of the morphological description as those admitted in the DELA system implemented in Unitex. Thus, an inflection paradigm is a set of actions to be performed on the lemma in order to generate its inflected forms, and of corresponding inflection features to be attached to each generated form.
- MULTIFLEX allows to extend the Unitex dictionary treatment to the inflection of a DELAC (DELA electronic dictionary of compounds) into a DELACF (DELA electronic dictionary of compounds' inflected forms). The format of the generated DELACF is compatible with Unitex, while the format of the DELAC is novel but inspired from the one of the DELAS (DELA electronic dictionary of simple words).

The following sections present, for several languages, complete examples of a DELAC into DELACF inflection within the MULTIFLEX/Unitex interface.

10.3.1 Complete Example in English

Let us assume that the description of morphological features of English is given by the following `Morphology.txt` file:

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

and that the equivalences between these features and their corresponding codes in DELA dictionaries are given by the following `Equivalences.txt` file:

```
English
s : Nb=s
p : Nb=p
```

Consider the following sample English DELAC file:

```
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
```

The corresponding inflection graphs *N1* and *N3* for simple words are represented on figures 10.10 and 10.11 while those for compounds are shown on figures 10.12 through 10.20.

The DELACF dictionary resulting from the inflection, via MULTIFLEX, of the above DELAC is as follows:

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

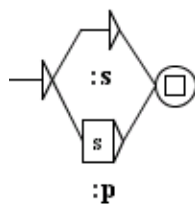


Figure 10.10: Inflection graph *N1* for English simple words

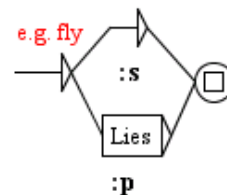


Figure 10.11: Inflection graph *N3* for English simple words

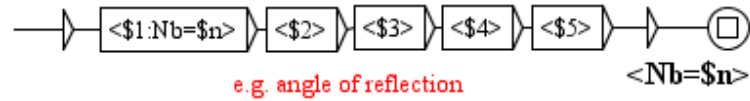


Figure 10.12: Inflection graph *NC_NXXXX* for English MWUs

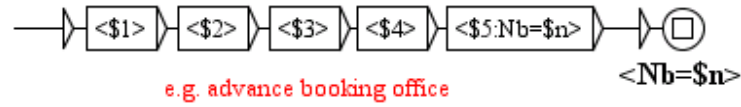


Figure 10.13: Inflection graph *NC_XXXXN* for English MWUs

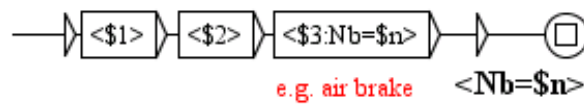


Figure 10.14: Inflection graph *NC_XXN* for English MWUs

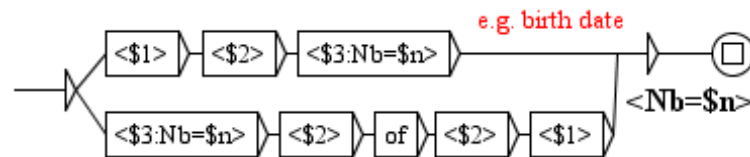


Figure 10.15: Inflection graph *NC_NN_NofN* for English MWUs

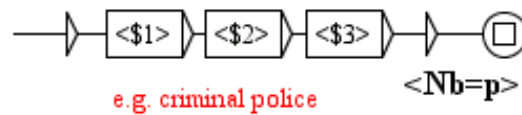
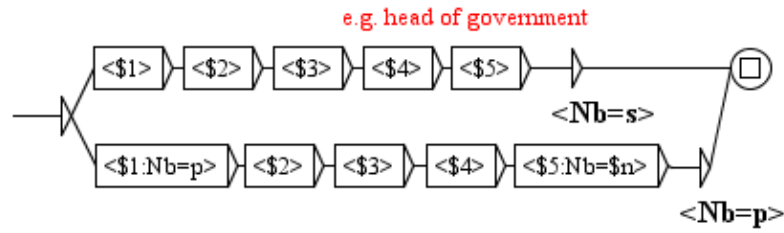
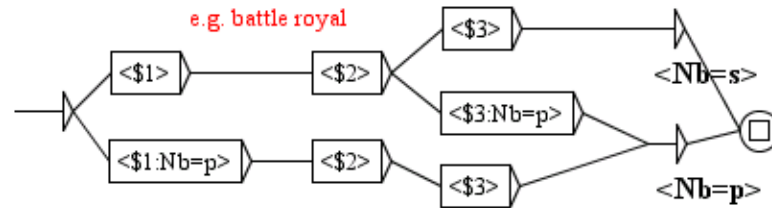
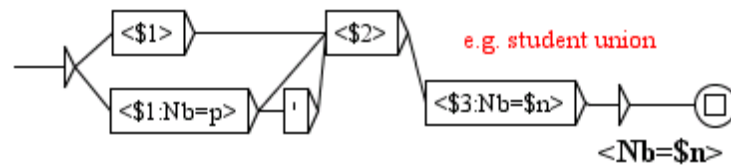


Figure 10.16: Inflection graph *NC_XXXinv* for English MWUs



Figure 10.17: Inflection graph *NC_XXNs* for English MWUs

Figure 10.18: Inflection graph NC_NofNs for English MWUsFigure 10.19: Inflection graph NC_NsNs for English MWUsFigure 10.20: Inflection graph $NC_Ns'N$ for English MWUs

10.3.2 Complete Example in French

Let us assume that the description of morphological features of French is given by the following `Morphology.txt` file:

```

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

```

and that the equivalences between these features and their corresponding codes in DELA

dictionaries are given by the following `Equivalences.txt` file:

```
French
s : Nb=s
p : Nb=p
m : Gen=m
f : Gen=f
```

Consider the following sample French DELAC file (the DELAS inflection codes may vary from those present in 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_NNm f
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_NXXXXXXX
porte-serviette (serviette.N21:fs) , NC_VNm
```

The corresponding inflection graphs for MWUs are shown on figures [10.21](#) through [10.27](#).

The DELACF dictionary resulting from the inflection, via MULTIFLEX, of the above DELAC is as follows:

```
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_NNm f:ms
cousins germains , cousin germain.NC_NNm f:mp
cousine germaine , cousin germain.NC_NNm f:fs
cousines germaines , cousin germain.NC_NNm f: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

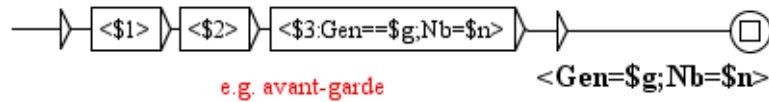


Figure 10.21: Inflection graph NC_XXN for French MWUs

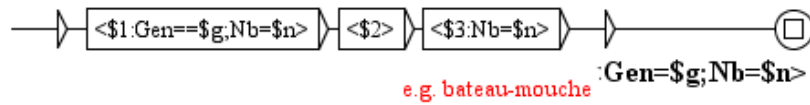


Figure 10.22: Inflection graph NC_NN for French MWUs

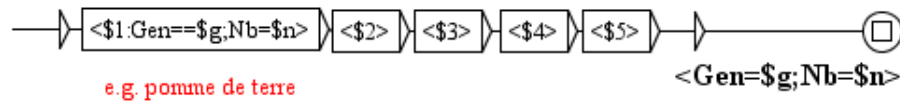


Figure 10.23: Inflection graph NC_NXXXX for French MWUs

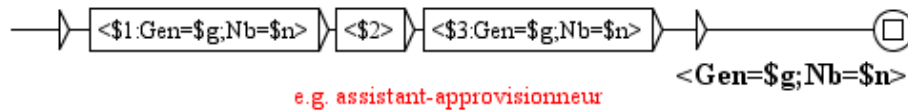


Figure 10.24: Inflection graph NC_NNmf for French MWUs

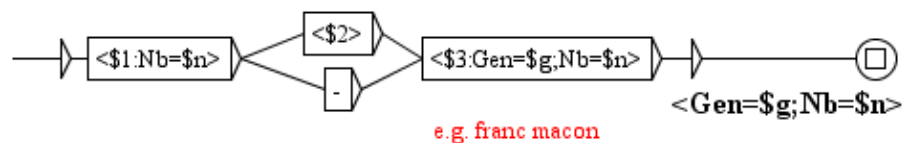
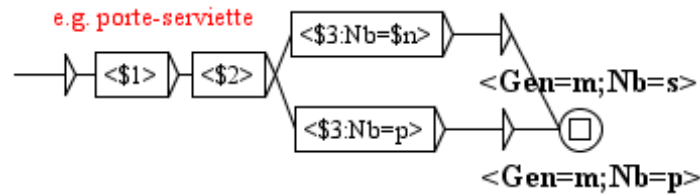


Figure 10.25: Inflection graph NC_AN1 for French MWUs

Figure 10.26: Inflection graph *NC_NXXXXXX* for French MWUsFigure 10.27: Inflection graph *NC_VNm* for French MWUs

10.3.3 Complete Example in Serbian

Let us assume that the description of morphological features of Serbian is given by the following `Morphology.txt` file:

```

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:

```

The peculiarity of this morphological model is not only its reachness but also the existence of *no-care* features like *Anim=g* or *Det=e*. These features agree with all other features in the same category. They are used only for some particular subclasses of nouns or adjectives and are necessary for a better compactness of the inflection paradigms of simple words which are already considerably huge, and would be even larger if no *no-care* symbols were used.

Let us assume that the equivalences between the above features and their corresponding

codes in DELA dictionaries are given by the following `Equivalences.txt` file:

```
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
```

Consider the following sample Serbian DELAC file (the DELAS inflection codes may vary from those present in 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:aefplg) naci je(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
istraxzni(istraxzni.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
```

The corresponding inflection graphs for MWUs are shown on figures [10.28](#) through [10.35](#).

The DELACF dictionary resulting from the inflection, via MULTIFLEX, of the above DELAC is as follows:

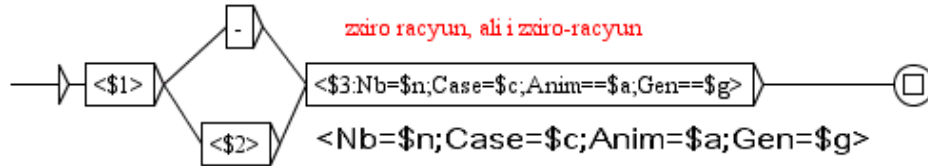
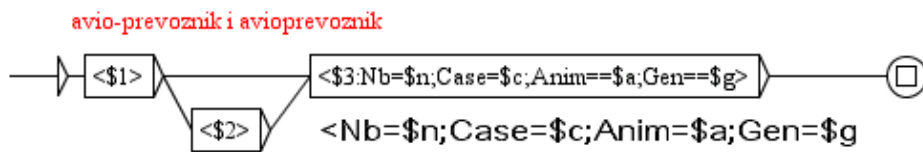
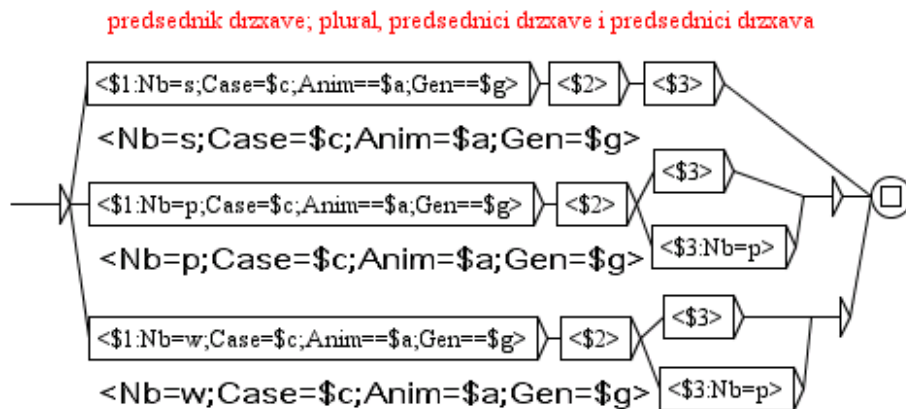
```
zxiro-racyun,zxiro racyun.NC_2XN1+N+Comp:s1qm
zxiro-racyuna,zxiro racyun.NC_2XN1+N+Comp:s2qm
zxiro-racyunu,zxiro racyun.NC_2XN1+N+Comp:s3qm
zxiro-racyun,zxiro racyun.NC_2XN1+N+Comp:s4qm
zxiro-racyune,zxiro racyun.NC_2XN1+N+Comp:s5qm
```

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
 Ujedinjene nacije, Ujedinjene nacije.NC_AXN3+N+Comp+NProp+Org:fp1q
 Ujedinjenih nacija, Ujedinjene nacije.NC_AXN3+N+Comp+NProp+Org:fp2q
 Ujedinjenima nacijama, Ujedinjene nacije.NC_AXN3+N+Comp+NProp+Org:fp3q
 Ujedinjenim nacijama, Ujedinjene nacije.NC_AXN3+N+Comp+NProp+Org:fp3q
 Ujedinjene nacije, Ujedinjene nacije.NC_AXN3+N+Comp+NProp+Org:fp4q
 Ujedinjene nacije, Ujedinjene nacije.NC_AXN3+N+Comp+NProp+Org:fp5q
 Ujedinjenima nacijama, Ujedinjene nacije.NC_AXN3+N+Comp+NProp+Org:fp6q
 Ujedinjenim nacijama, Ujedinjene nacije.NC_AXN3+N+Comp+NProp+Org:fp6q
 Ujedinjenima nacijama, Ujedinjene nacije.NC_AXN3+N+Comp+NProp+Org:fp7q
 Ujedinjenim nacijama, Ujedinjene 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
 istraxzne sudije, istraxzni sudija.NC_AXNF+N+Comp:1vfp
 istraxznih sudija, istraxzni sudija.NC_AXNF+N+Comp:2vfp
 istraxznima sudijama, istraxzni sudija.NC_AXNF+N+Comp:3vfp
 istraxznim sudijama, istraxzni sudija.NC_AXNF+N+Comp:3vfp
 istraxzne sudije, istraxzni sudija.NC_AXNF+N+Comp:4vfp
 istraxzne sudije, istraxzni sudija.NC_AXNF+N+Comp:5vfp
 istraxznima sudijama, istraxzni sudija.NC_AXNF+N+Comp:6vfp
 istraxznim sudijama, istraxzni sudija.NC_AXNF+N+Comp:6vfp
 istraxznima sudijama, istraxzni sudija.NC_AXNF+N+Comp:7vfp

istraxxnim sudijama,istraxxni sudija.NC_AXNF+N+Comp:7vfp
 istraxxne sudije,istraxxni sudija.NC_AXNF+N+Comp:2vfw
 istraxxne sudije,istraxxni sudija.NC_AXNF+N+Comp:4vfw
 istraxxnoga sudiju,istraxxni sudija.NC_AXNF+N+Comp:ms4v
 istraxxnog sudiju,istraxxni sudija.NC_AXNF+N+Comp:ms4v
 istraxxni sudija,istraxxni sudija.NC_AXNF+N+Comp:1vms
 istraxxnoga sudije,istraxxni sudija.NC_AXNF+N+Comp:2vms
 istraxxnog sudije,istraxxni sudija.NC_AXNF+N+Comp:2vms
 istraxxnomu sudiji,istraxxni sudija.NC_AXNF+N+Comp:3vms
 istraxxnome sudiji,istraxxni sudija.NC_AXNF+N+Comp:3vms
 istraxxnom sudiji,istraxxni sudija.NC_AXNF+N+Comp:3vms
 istraxxnomu sudiji,istraxxni sudija.NC_AXNF+N+Comp:7vms
 istraxxnome sudiji,istraxxni sudija.NC_AXNF+N+Comp:7vms
 istraxxnom sudiji,istraxxni sudija.NC_AXNF+N+Comp:7vms
 istraxxni sudijo,istraxxni sudija.NC_AXNF+N+Comp:5vms
 istraxxni sudija,istraxxni sudija.NC_AXNF+N+Comp:5vms
 istraxxnim sudijom,istraxxni 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

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 10.28: Inflection graph NC_2XN1 for Serbian MWUsFigure 10.29: Inflection graph NC_2XN2 for Serbian MWUsFigure 10.30: Inflection graph NC_N2X1 for Serbian MWUs

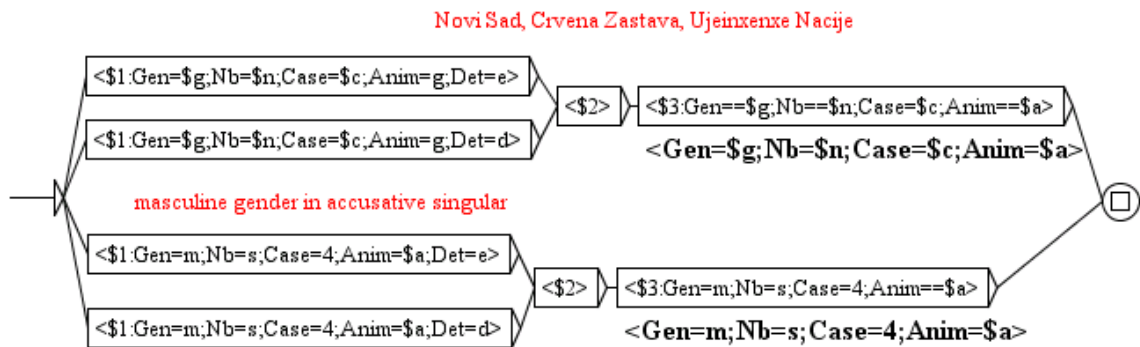


Figure 10.31: Inflection graph NC_AXN3 for Serbian MWUs

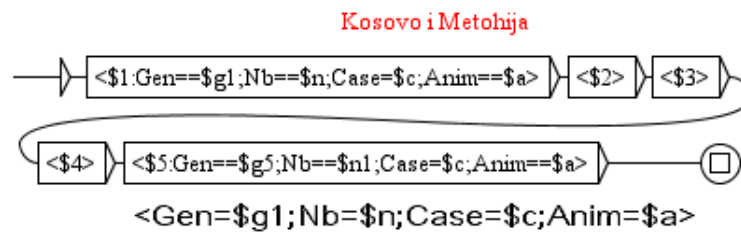


Figure 10.32: Inflection graph NC_N3XN for Serbian MWUs

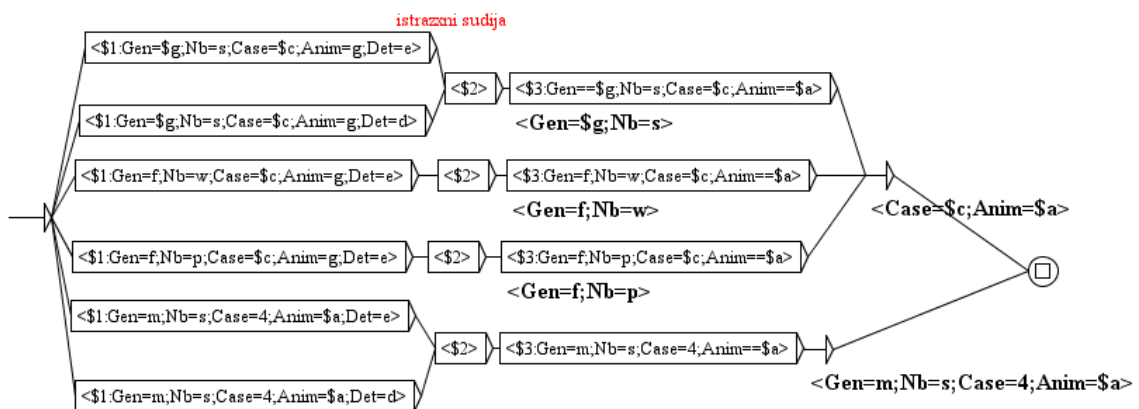


Figure 10.33: Inflection graph NC_AXNF for Serbian MWUs

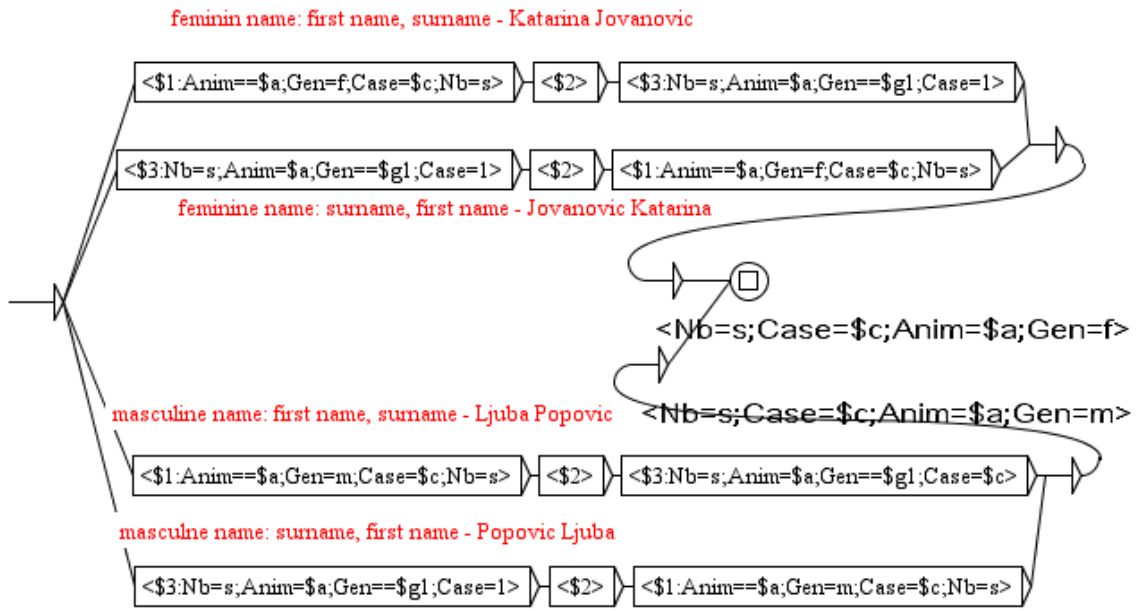


Figure 10.34: Inflection graph *NC_ImePrezime* for Serbian MWUs

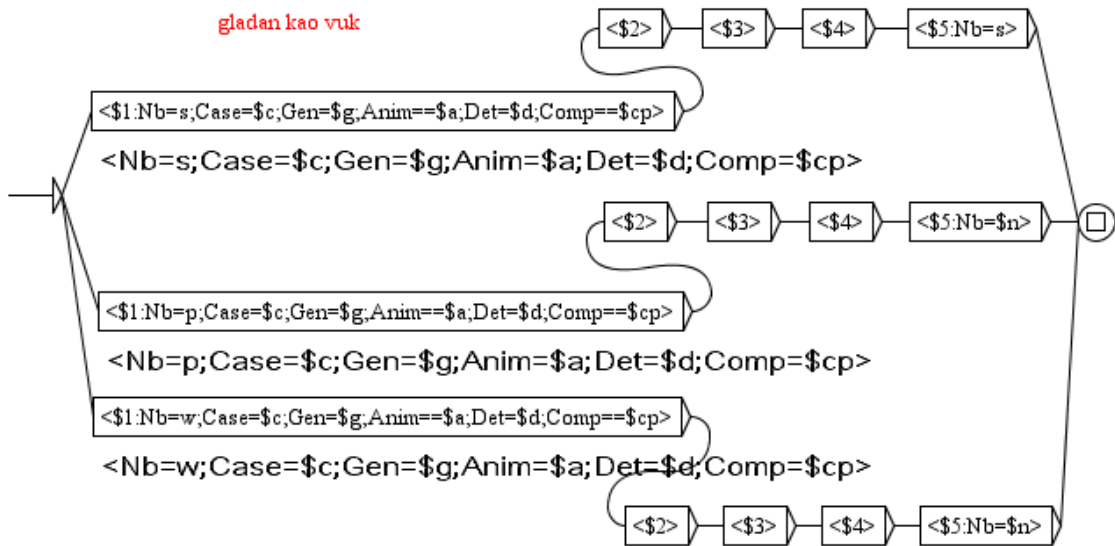


Figure 10.35: Inflection graph *AC_A3XN2* for Serbian MWUs

Chapter 11

Cascade of Transducers

The general principle of a *cascade of transducers* is to apply several transducers, one after the other, onto a text to modify this text. For the system Unitex, a transducer is a graph. Such a system can be used to analyze a text in different ways: syntactic analysis, chunking, information extraction or simply to transform and/or prepare a text further treatments.

The prototype of the *CasSys* system was created in 2002 at the LI (the laboratory of Computer science of the University of Tours) as part of Nathalie Friburger's PhD Thesis ([30]). Later on, it was fully dedicated to named entity extraction. CasSys was generalized to allow any sort of work needing a cascade: throughout the years, it was improved but never really integrated in Unitex, until the recent "Feder-Région Centre" project launched by Denis Maurel which resulted in the complete integration of CasSys in Unitex carried out by David Nott and Nathalie Friburger.

A priori, the use of CasSys is roughly a succession of locate patterns eventually with special options and behaviors. In this chapter, we will explain how to create cascades of transducers and how to apply them. After that, we deals with details on what is very specific in CasSys.

11.1 Applying a cascade of transducers with CasSys

In the text menu, you can select the submesnu "Apply CasSys cascade..." (11.1) which will open the CasSys window. This submenu "Apply CasSys cascade" is active only if a text has previously been opened.

11.1.1 Creating the list of transducers

The list of transducers of a cascade is saved in a text file with the extension .csc (ex: mycascade.csc).

In order to create the list of transducers, click on the "new" button in the CasSys window. If you wish to modify an existing cascade, you can choose the file name of the cascade, then click on the "edit" button. These two buttons (edit or new) will open the "CasSys transducer configuration" window.

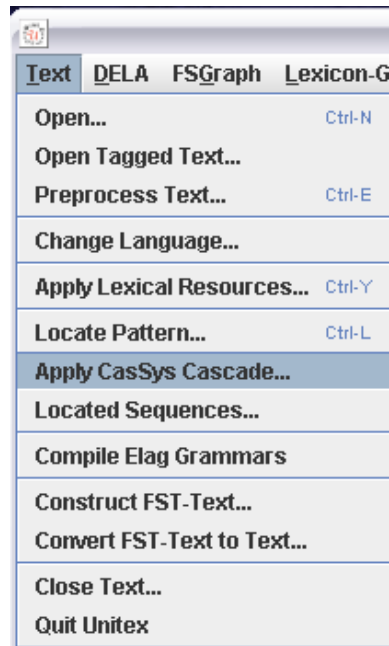


Figure 11.1: "Text" menu of Unitex and submenu "Apply CasSys Cascade"

11.1.2 Editing the list of transducers

The Cassys Transducer configuration window (11.2) comprises three parts :

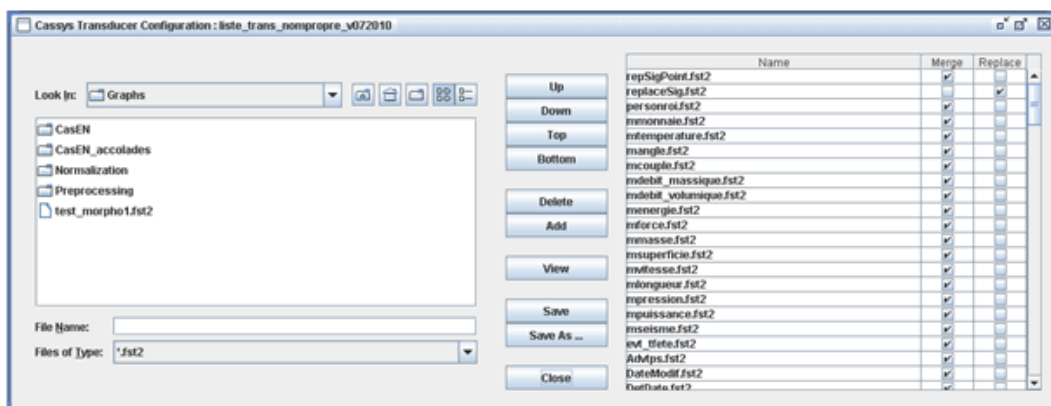


Figure 11.2: Cassys Transducer configuration window with a list of transducers on the right hand side

- The left frame of the Cassys Transducer configuration window contains a file explorer in which you can select the transducers to add to the cascade. The file explorer only displays fst2 files (all the graphs you want to place in the list of transducers must be compiled in fst2 format). To edit the cascade, you must select the graphs in the file explorer in the left frame and drag and drop them into the right frame of the window.

- In the right frame, there is the list of transducers which is empty when creating a new cascade. The list of transducers is ordered: the first graph on the list is applied first, and the last graph on the list is applied last. The list is composed of three columns: the first one contains the name of each graph, the second one the option "merge" and the third one the option "replace". You must choose between "merge" or "replace" which are the two different manners the outputs of the graph can be taken into account (like in the locate pattern program). "Merge" indicates that the outputs of a graph will be merged into the text whereas "replace" will replace the recognized text with the outputs of the graph. When using a cascade of transducers, it makes no sense to take into account the outputs of the graph: that is the reason why this option (no display of the outputs) is not available. When the CasSys transducer configuration window opens the list of transducers, it verifies whether all the files on the list of transducers exist or not: if a file does not exist, it appears in red on the list (which can mean you have forgotten to compile the graph into fst2 format). If you leave the mouse cursor for a few seconds on the file name, the complete path of the graph will appear.
- In the middle column, you will have buttons which are described below:
 - The "up"/"down"/"top"/"bottom" buttons are used to modify the order of the transducers on the list (it moves the selected transducer in the list); "up" and "down" to move the selected transducer one line up or down, and "Top" and "Bottom" to move the selection to the top or to the end of the list.
 - The "delete" button permits to remove a selected transducer from the list of transducers. The "Add" button adds a transducer (previously selected in the explorer) onto the list. The "Add" button replaces the drag and drop actions described above.
 - The "View" button opens the selected graph either in the file explorer or in the list of transducers of the window. It is very useful to get a quick access to any transducer either to take a quick look at its content or to modify it.
 - The "Save" button and "Save as" button permit to save the list of transducers. By default, the lists of transducers are stored in the CasSys folder of the current language (e.g. English/Cassys).
 - The "Close" button closes the current window.

11.1.3 Launching a cascade

The CasSys window (11.3) displays the content of the CasSys folder of the current language. It permits to choose the file containing the list of transducers to apply on the text. The displayed files are in a special file type "CaSCade configuration file" (.csc). When this list is chosen, you can click on the "Launch" button to apply the cascade.

11.1.4 Displaying the results of a cascade

The result of a cascade is an index file (concord.ind), just as for the locate pattern operation. This index file contains all the sequences recognized using the restrictions imposed by the

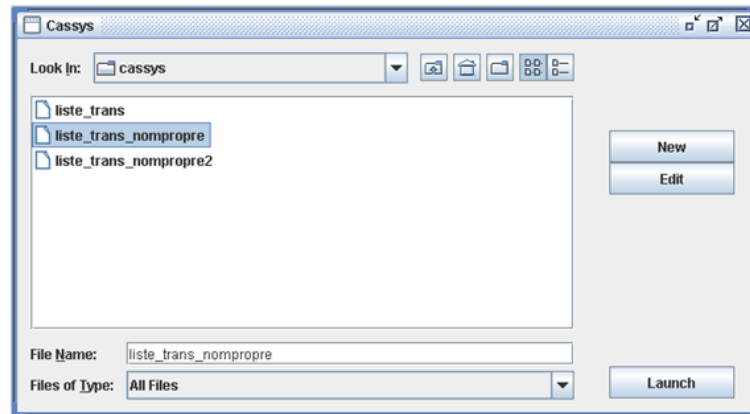


Figure 11.3: CasSys Window to launch a cascade of transducers

rules of unitex.

In order to display a concordance, you have to click on the "Build concordance" button (as described in Chapter 6) in the menu "Text / Located sequences" (11.4 presents a sample of concordance of the results of a cascade recognizing named entities). To create the file containing all the modifications of the cascade on the text, you have to click on "Modify text" in the "Located Sequences" window. The resulting file is a copy of the text in which the transducer outputs appear.

```

ieux sergent se mit à leur tête. " Merci, / capitaine..N+Entity+Function+Military/ ! dit Mr. Fog
nt : " Savez-vous une chose, ajouta-t-il, / capitaine..N+Entity+Function+Military/ ?... - Fogg.
que ainsi conçue : Suez à Londres. Rowan, / directeur..N+Entity+Function+Administration/ police,
able Batulcar, sorte de Barnum américain, / directeur..N+Entity+Function+Administration/ d'une t
eako, la grande cité qu'habite le mikado, / empereur..N+Entity+Function+Arismatic/ ecclésiast
eient quelques paroles, et, à ce moment, / le brigadier..N+Entity+Function+Military/ général, r
rche du steamer. Quand il était maniable, / le capitaine..N+Entity+Function+Military/ faisait ét
. Phileas Fogg voulait aller à Liverpool, / le capitaine..N+Entity+Function+Military/ ne voulait
étendant que j'avais tort de jouer pique, / le colonel..N+Entity+Function+Military/ m'a fait une
r. " Arrivé à Suez, mercredi 9 octobre, / 11 heures..N+Entity+Time+Hour/ matin. " Total des heur
e lendemain, c'était le 12 décembre. Du / 11, sept heures..N+Entity+Time+Hour/ du matin, au 21,
ut rapidement vers l'est. Le lendemain, / 13 décembre..N+Entity+Time+Date+Relative/, à midi, un
tion -, ne parlait que le surlendemain, / 14 décembre..N+Entity+Time+Date+Relative/. Et d'ailleu
saki et Yokohama. Arrivé le matin même, / 14 novembre..N+Entity+Time+Date+Relative/, à l'heure r
faux pont, tout y passa. Le lendemain, / 19 décembre..N+Entity+Time+Date+Relative/, on brûla la

```

Figure 11.4: Concordance of Cassys under Unitex

11.2 Details on Cassys

In this section , we present details concerning the functioning of Cassys.

11.2.1 Type of graphs used

Cassys uses the compiled version of the graphs (the fst2 files). Cassys can handle the local grammars (section 6.1) (syntactic graphs) presented in Chapter 6. These grammars can use subgraphs, morphological filters and mode, and allow to refer to information in dictionaries. The grammars used in the cascade must follow the constraints of the grammars used in Unitex.

11.2.2 The Unitex rules used for the cascade

In the cascade, each successive graph is applied following the unitex rules:

- Insertion to the left of the matched patterns : in the merge mode, the output is inserted to the left of the recognized sequence.
- Priority of the leftmost match : during the application of a local grammar, overlapping occurrences are all indexed. During the construction of a concordance, all these overlapping occurrences are present but CasSys modifies the text with each graph of the cascade : so it is necessary to choose among these occurrences the one that will be taken into account. To do that, the priority is given to the leftmost sequence.
- Priority of the longest match : in CasSys, during the application of a graph, it is the longest sequence that will be kept.
- Search limitation to a certain number of occurrences: in Cassys, this search is not limited : such a limitation has no sense in the use of CasSys. We always index all utterances in the text.

11.2.3 A special way to mark up patterns with CasSys

The output of the transducers can be used to insert special information into texts, particularly to mark up the recognized patterns: it is possible to use all the marks you want such as (), [], "", etc. or xml tags such as <xxx> </xxx>, but Cassys proposes a special way to mark up patterns, that offers some advantages and that we present here.

Unitex splits texts into different sorts of tokens like the sentence delimiter S; the stop marker STOP, contiguous sequences of letters, lexical tags aujourd'hui, ADV, etc. The lexical tag is used by CasSys in a special way. The lexical tag (between curly brackets) is normally used to avoid ambiguities (see explanation in section 2.5.4 and in section 7.5.1). For example, in a text, if you have the token {*curly brackets*, N}, neither "curly" nor "brackets" will be recognized but the whole sequence "curly brackets". A lexical tag can contain complex lexical information like *N+Pers+Hum:fs*. In a graph, you can look for a lexical token using the lexical information it contains: for example, you can write <.N> to search a noun, <.Pers+Hum> for a human person or <.Pers>. These lexical masks are explained in the Chapter Searching with Regular Expressions in section 4.3.1.

In Cassys, we use the lexical tag in a special way. A cascade of transducers is interesting to locate the island of certainty first. It is necessary for such a system to avoid that previously

recognized patterns be ambiguous with patterns recognized by the following graphs. To do that, you can tag the patterns of your graphs surrounding them by $\{$ and $\text{,tag1+tag2+tagn}\}$ in the outputs of the graph (where tag1 , tag2 , etc. are your own tags).

To explain this behavior, here is a very simple example. The text on which we work is :

bac a b c cc a b b ba ab a b bca a b c abaabc.

The graph grfAB (??) recognizes the sequence *ab* in the text and tags this sequence with the lexical tag $\{a b,\text{AB}\}$. This graph is merged with the text and adds its outputs $\{$ and $\text{,AB}\}$ to the text.

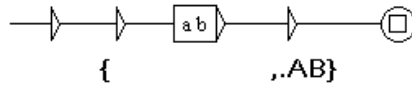


Figure 11.5: The graph grfAB

The resulting text is : *bac {a b,\text{AB}} c cc {a b,\text{AB}} b ba ab {a b,\text{AB}} bca {a b,\text{AB}} c abaabc.*

Now the pattern *a b* is tagged *AB*. A part (a or b alone) of this pattern cannot be recognized because of the tagging of *a b*.

After that graph, the cascade applies another graph named tagAB (11.6) containing the lexical masks $\langle AB \rangle$. It recognizes all the sequences lexically tagged by the previous graph.

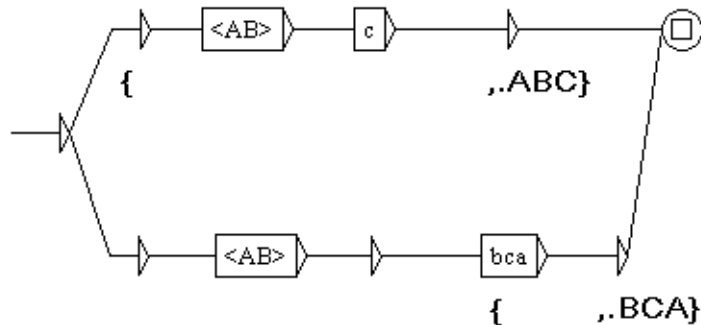


Figure 11.6: The graph tagAB

The resulting text is : *bac {{a b,\text{AB}} c,\text{ABC}} cc {a b,\text{AB}} b ba ab {a b,\text{AB}} {bca,\text{BCA}} {{a b,\text{AB}} c,\text{ABC}} abaabc.*

The concordance displayed by Unitex should be like in (11.7). For programming reasons (ambiguities between characters in the curly brackets of the lexical tags), we have no option but to place backslashes \backslash before all ambiguous characters ; that is why these symbols are protected with \backslash in the concordance to avoid problems in Unitex.

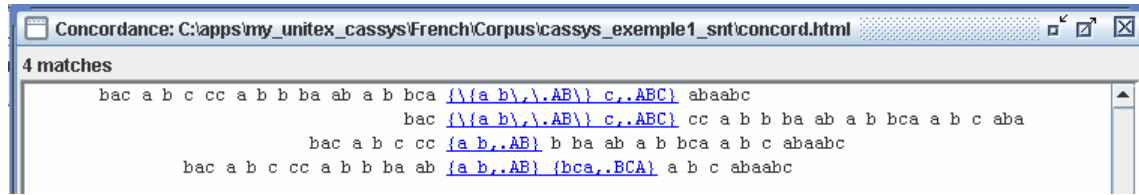


Figure 11.7: The concordance resulting from this cascade

This examples shows that the writing of graphs using the lexical tags created by preceding graphs is very simple. The tags can overlap each other.

11.2.4 Interest of a cascade of transducer

Unitex grammars are known as Context free grammars and contain the notion of transduction derived from the field of finite state automata. A grammar with transduction (a transducer) is enabled to produce some output. Cassys is dedicated to the application of transducers in the form of a cascade.

Transducers are interesting because they allow the association of a recognized sequence to informations found in the outputs of the graphs.

These outputs can:

- Be merged into the recognized sequence found in the text and appear in the resulting concordance or modified text.
- Replace the recognized sequence to modify the text.

These two operations transform the text, add information inside the text for different purposes or modify it. The cascade can then be used for syntactic analysis, chunking, information extraction, etc.

The advantage of a cascade is mainly that it is a good way to manage priority between patterns that you want to find in a text. If you know two ambiguous pattern, you can apply the less ambiguous pattern first.

11.2.5 The longest pattern

The heuristic of the longest pattern matching is applied to each transducer of the cascade. When a graph is applied to a text, several paths can be recognized by the graph.

If the graph arrives to its final state through several paths then it is the path that recognizes the longest pattern that is chosen. The longer the pattern is, the less ambiguous it is. If the transducer doesn't arrive to its final state then the recognizing step restarts on the next word of the text.

The longest pattern matching heuristic is interesting but if several paths of the same size are recognized there is still a problem; one of the paths will be chosen with no control on this choice for the user, meaning that the worst path might well be chosen. A solution to that problem can be the creation of a cascade of transducer giving priority to a transducer among the list of transducers.

If a graph contains two paths that are ambiguous, one can create two graphs containing one path each. The first graph will contain the safest path, the second graph the least safe path.

Cassys keeps all the text created by each graph of the cascade. This can be useful to test, debug or check the different results of the cascade. It is possible to correct the errors on the order of the graphs or to find the errors in the writing of the graphs. A good idea is to place the name of the transducer recognizing a pattern in the outputs of the graphs: thanks to that, you can see in the final results the name of the graph by which a pattern is recognized.

11.2.6 Files resulting from CasSys

If you apply a cascade on the text named `example.txt`, two folders are created: `example_snt` and `example_csc`. The most important files produced in `example_csc` are the results obtained by each graphs. These files are named according to the number of the graph which produced them (if the third graph finds a pattern, the result will be the file named `example_3.snt`).

Chapter 12

Use of external programs

This chapter presents the use of the different programs of which Unitex is composed. These programs, which can be found in the `Unitex/App` folder, are automatically called by the interface (in fact, `UnitexToolLogger` is actually called, in order to reduce significantly the size of the downloadable zip file). It is possible to see the commands that have been executed by clicking on "Info>Console". It is also possible to see the options of the different programs on "Info>Help on commands" (see Figure 12.1). Note that that all Unitex programs support the `-h/--help` option.

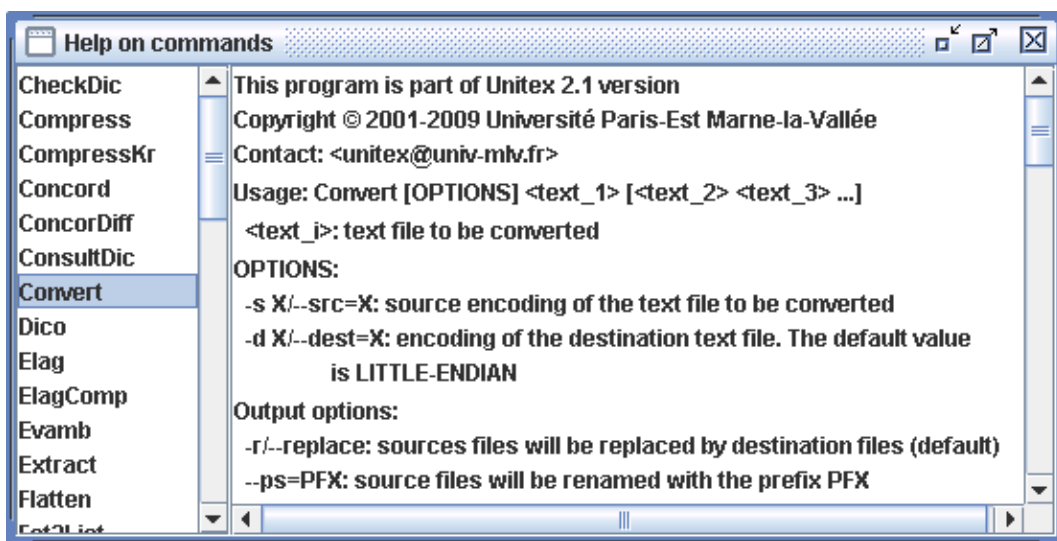


Figure 12.1: Help on commands

WARNING: many programs use the text directory (`my_text_snt`). This directory is created by the graphical interface after the normalization of the text. If you work with the command line, you have to create the directory manually before the execution of the program `Normalize`.

WARNING (2): whenever a parameter contains spaces, it needs to be enclosed in quotation marks so it will not be considered as multiple parameters.

WARNING (3): many programs need an `Alphabet.txt` file. For all those programs, this information can be omitted. In that case, a default definition of letters is used (see `u_is_letter` in `Unicode.cpp` source file).

12.1 Creating log files

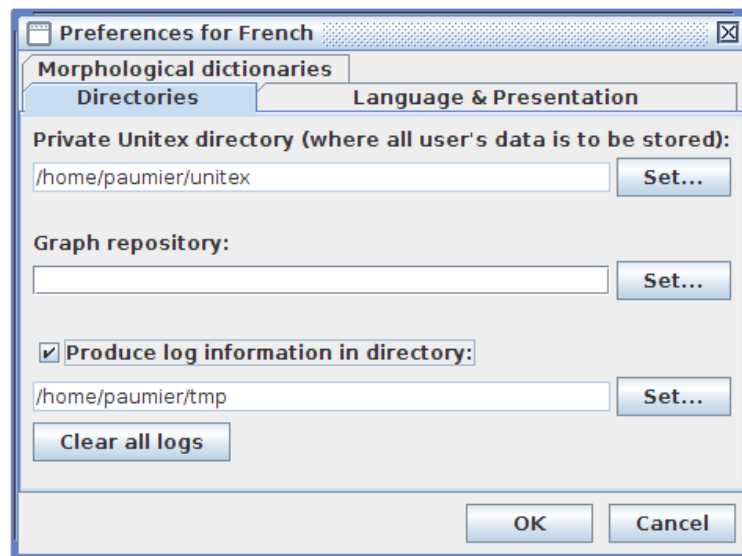


Figure 12.2: Logging configuration

You can create log files of external program launches. These log files can be useful for debugging or regression tests. You just need to enable this feature in the Preferences frame. You have to choose a log directory where all log files will be stored and to select the "Produce log" check box. Clicking on the "Clear all logs" button will remove all log files contained in this directory, if any. Then, any further program execution will produce a `unitex_log_XXX.ulp` file located in the log directory. XXX stands for the log number that can be found in the console (see next section).

12.2 The console

When Unitex launches an external program, the invoked command line is stored in the console. To see it, click on "Info>Console". When a command emits no error message, it is displayed with a green icon. Otherwise, the icon is a red triangle that you can click on to see the error messages, as shown on Figure 12.3. This is useful when an error message occurs

so fast that you cannot read it. If a command has been logged, its log number appears in the second column. Note that you can export all the commands displayed in the console to the clipboard with Ctrl+C.

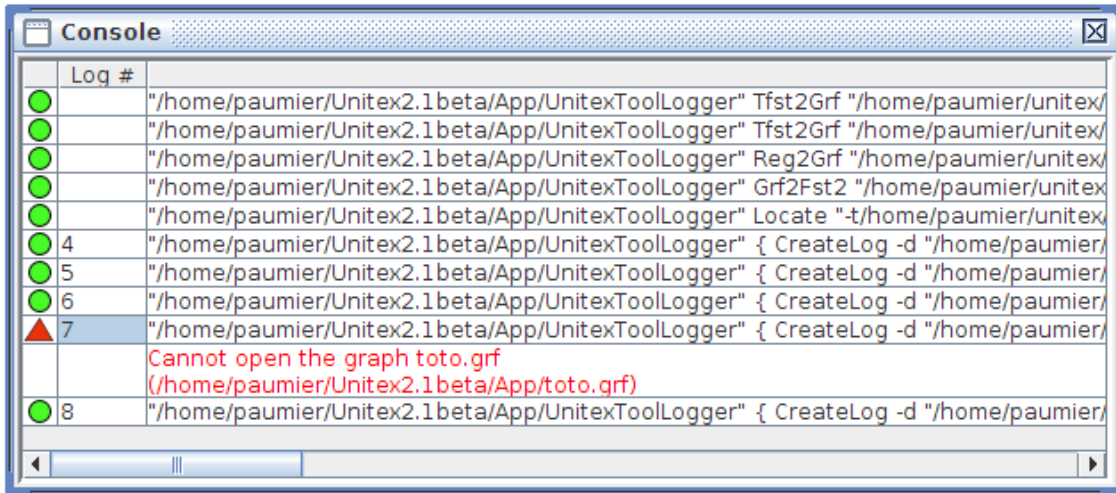


Figure 12.3: Console

12.3 Text file encoding parameters

Unitex uses Unicode for text file13.1. All program which read or write text file share same encoding parameters. Possible format are utf16le-bom, utf16le-no-bom, utf16be-bom, utf16be-no-bom, utf8-bom, utf8-no-bom, for Unicode Big-Endian, Little-Endian and UTF-8, with or without Unicode byte order mark at the beginning of the file. For the input format, you can specify several *-bom encoding separated by comma, but only one *-no-bom encoding.

OPTIONS:

- `-k=ENCODING/--input_encoding=ENCODING`: input text file format. Can contain several value, separated by a comma;
- `-q=ENCODING/--output_encoding=ENCODING`: output text file format.

By default, value are `--input_encoding=utf16le-bom, utf16be-bom, utf8-bom --output_e`

12.4 BuildKrmwuDic

```
BuildKrmwuDic [OPTIONS] dic
```

This program generates a MWU dictionary graph from a text table `dic` describing each component of each MWU.

OPTIONS:

- -o GRF/--output=GRF: .grf file to produce;
- -d DIR/--directory=DIR: inflection directory containing the inflection graphs required to produce morphological variants of roots;
- -a ALPH/--alphabet=ALPH: alphabet file to use;
- -b BIN/--binary=BIN: .bin simple word dictionary to use.

12.5 Cassys

Cassys [OPTIONS] <snt>

This program applies an ordered list of grammars to a text and constructs an index of the occurrences found.

OPTIONS:

- -a ALPH/--alphabet=ALPH: the language alphabet file
- -r X/--transducer_dir=X: take transducer on directory X (so you don't specify full path for each transducer; note that X must be (back)slash terminated
- -l TRANSDUCERS_LIST/--transducers_list=TRANSDUCERS_LIST: the transducers list file with their output policy
- -s transducer.fst2/--transducer_file=transducer.fst2: a transducer to apply
- -m output_policy/--transducer_policy=output_policy: the output policy of the transducer specified
- -t TXT/--text=TXT: the text file to be modified, with extension .snt;
- -i/--in_place: mean uses the same csc/snt directories for each transducer
- -d/--no_create_directory: mean the all snt/csc directories already exist and don't need to be created
- -g minus/--negation_operator=minus: uses minus as negation operator for Unitex 2.0 graphs
- -g tilde/--negation_operator=tilde: uses tilde as negation operator (default)
- -h/--help: display this help

Cassys applies a list of grammar to a text and saves the matching sequence index in a file named `concord.ind` stored in the text directory. The target text file has to be a preprocessed snt file with its `_snt/` directory. The transducer list file is a file in which each line contains the path to a transducer followed by the output policy to be applied to this transducer.

Instead a list file, you can specify each file and each output policy by a set of couple of `-s/--transducer_file` and `-m/--transducer_policy` argument to enumerate the list

The policy may be MERGE or REPLACE.

The file option, the alphabet option and the transducer list file option are mandatory

As the locate pattern program, this program saves the references to the found occurrences in a file called `concord.ind` stored in the `_snt` directory of the text. The file `concord.ind` produced is in the same format as described in the chapter 13 , but the cascade may be constituted of graphs applied in merge or replace mode so the `#M` or `#R` at the first line of the file `concord.ind` has no sense in this context.

12.6 CheckDic

CheckDic [OPTIONS] dic

This program carries out the verification of the format of a dictionary of DELAS or DELAF type. `dic` corresponds to the name of the dictionary that is to be verified.

OPTIONS:

- `-f/--delaf`: checks an inflected dictionary;
- `-s/--delas`: checks a non inflected dictionary;
- `-r/--strict`: strict syntax checking against unprotected dot and comma;
- `-t/--tolerate`: tolerates unprotected dot and comma (default);
- `-n/--no_space_warning`: tolerates spaces in grammatical/semantic/inflectional codes;
- `-p/--skip_path`: does not display the full path of the dictionary (useful for consistent log files across several systems);
- `-a ALPH/--alphabet=ALPH`: specifies the alphabet file to use.

The program checks the syntax of the lines of the dictionary. It also creates a list of all characters occurring in the inflected and canonical forms of words in the text, the list of grammatical codes and syntax, as well as the list of inflection codes used. The results of the verification are stored in a file called `CHECK_DIC.TXT`.

Selecting strict syntax checking detects using unprotected dot in inflected form, or unprotected comma in lemma. The `--tolerate` option acts like Unitex 2.0 and lower and does not detect them.

12.7 Compress

Compress [OPTIONS] dictionary

OPTIONS:

- `-o BIN/--output=BIN`: sets the output file. By default, a file `xxx.dic` will produce a file `xxx.bin`;
- `-f/--flip`: indicates that the inflected and canonical forms should be swapped in the compressed dictionary. This option is used to construct an inverse dictionary which is necessary for the program `Reconstruca0`;
- `-s/--semitic`: indicates that the semitic compression algorithm should be used. Setting this option with semitic languages like Arabic significantly reduces the size of the output dictionary.

This program takes a DELAF dictionary as a parameter and compresses it. The compression of a dictionary `dico.dic` produces two files:

- `dico.bin`: a binary file containing the minimum automaton of the inflected forms of the dictionary;
- `dico.inf`: a text file containing the compressed forms required for the reconstruction of the dictionary lines from the inflected forms contained in the automaton.

For more details on the format of these files, see chapter 13.

12.8 Concord

Concord [OPTIONS] <index>

This program takes a concordance index file produced by the program `Locate` and produces a concordance. It is also possible to produce a modified text version taking into account the transducer outputs associated to the occurrences. Here is the description of the parameters:

OPTIONS:

- `-f FONT/--font=FONT`: the name of the font to use if the output is an HTML file;
- `-s N/--fontsize=N`: the font size to use if the output is an HTML file. The font parameters are required if the output is an HTML file;
- `-l X/--left=X`: number of characters on the left of the occurrences (default=0). In Thai mode, this means the number of non-diacritic characters.

- `-r X/--right=X`: number of characters (non-diacritic ones in Thai mode) on the right of the occurrences (default=0). If the occurrence is shorter than this value, the concordance line is completed up to `right`. If the occurrence is longer than the length defined by `right`, it is nevertheless saved as whole.

NOTE: For both `--left` and `--right`, you can add the `s` character to stop at the first `{S}` tag. For instance, if you set `40s` for the left value, the left context will end at 40 characters at most, less if the `{S}` tag is found before.

Sort order options:

- `--TO`: order in which the occurrences appear in the text (default);
- `--LC`: left context for primary sort, then occurrence for secondary sort;
- `--LR`: left context, then right context;
- `--CL`: occurrence, then left context;
- `--CR`: occurrence, then right context;
- `--RL`: right context, then left context;
- `--RC`: left context, then occurrence.

For details on the sorting modes, see section [4.8.2](#).

Output options:

- `-H/--html`: produces a concordance in HTML format encoded in UTF-8 (default);
- `-t/--text`: produces a concordance in Unicode text format;
- `-g SCRIPT/--glossanet=SCRIPT`: produces a concordance for GlossaNet in HTML format. The HTML file is encoded in UTF-8;
- `-p SCRIPT/--script=SCRIPT`: produces a HTML concordance file where occurrences are links described by `SCRIPT`. For instance, if you use `-phttp://www.google.com/search?q=`, you will obtain a HTML concordance file where occurrences are hyperlinks to Google queries;
- `-i/--index`: produces an index of the concordance, made of the content of the occurrences (with the grammar outputs, if any), preceded by the positions of the occurrences in the text file given in characters;
- `-u/--uima`: the same as `--index`, but the ending position of each occurrence is also given;
- `-e/--xml`: produces xml index of the concordance;

- `-w/--xml-with-header`: produces xml index of the concordance with full xml header;
- `-A/--axis`: quite the same as `--index`, but the numbers represent the median character of each occurrence. For more information, see [31];
- `-x/--xalign`: another index file, used by the text alignment module. Each line is made of 3 integers *X Y Z* followed by the content of the occurrence. *X* is the sentence number, starting from 1. *Y* and *Z* are the starting and ending positions of the occurrence in the sentence, given in characters;
- `-m TXT/--merge=TXT`: indicates to the program that it is supposed to produce a modified version of the text and save it in a file named `TXT` (see section 6.10.4).

Other options:

- `-d DIR/--directory=DIR`: indicates to the program that it must not work in the same directory than `<index>` but in `DIR`;
- `-a ALPH/--alphabet=ALPH`: alphabet file used for sorting;
- `-T/--thai`: option to use for Thai concordances.

The result of the application of this program is a file called `concord.txt` if the concordance was constructed in text mode, a file called `concord.html` if the output mode was `--html`, `--glossnet` or `--script`, and a text file with the name defined by the user of the program if the program has constructed a modified version of the text.

In `--html` mode, the occurrence is coded as a hypertext link. The reference associated to this link is of the form ``. *X* et *Y* represent the beginning and ending positions of the occurrence in characters in the file `text_name.snt`. *Z* represents the number of the sentence in which the occurrence was found.

12.9 ConcorDiff

```
ConcorDiff [OPTIONS] <concor1> <concor2>
```

This program takes two concordance files and produces an HTML page that shows their differences (see section 6.10.6, page 144). `<concor1>` and `<concor2>` concordance index files must have absolute names, because Unitex uses these names to deduce on which text there were computed.

OPTIONS:

- `-o X/--out=X`: output HTML page;
- `-f FONT/--font=FONT`: name of the font to use in output HTML page;
- `-s N/--size=N`: font size to use in output HTML page.

12.10 Convert

Convert [OPTIONS] <text_1> [<text_2> <text_3> ...]

With this program you can transcode text files.

OPTIONS:

- `-s X/--src=X`: input encoding;
- `-d X/--dest=X`: output encoding (default=LITTLE-ENDIAN);

Transliteration options (only for Arabic):

- `-F/--delaf`: the input is a DELAF and we only want to transliterate the inflected form and the lemma;
- `-S/--delas`: the input is a DELAS and we only want to transliterate the lemma.

Output options:

- `-r/--replace`: input files are overwritten (default);
- `-o file/--output=file`: name of destination file (only one file to convert);
- `--ps=PREFIX`: input files are renamed with the PREFIX prefix (`toto.txt` ⇒ `PFXtoto.txt`);
- `--pd=PREFIX`: output files are renamed with the PREFIX prefix;
- `--ss=SUFFIX`: input files are named with the SUFFIX suffix; (`toto.txt` ⇒ `totoSFX.txt`);
- `--sd=SUFFIX`: output files are named with the SUFFIX suffix.

HTML options:

Convert offers some special options dedicated to HTML files. You can use a combination of the following options:

- `--dnc` (Decode Normal Chars): things like `é`, `x` and `ø` will be decoded as the single equivalent unicode character, except if it represents an HTML control character;
- `--dcc` (Decode Control Chars): `<`, `>`, `&` and `"` will be decoded as `<`, `>` and `&` and the quote (the same for their decimal and hexadecimal representations);
- `--eac` (Encode All Chars): every character that is not supported by the output encoding will be encoded as a string like `ǉ`;
- `--ecc` (Encode Control Chars): `<`, `>`, `&` and the quote will be encoded by `<`, `>`, `&` and `"`;

All HTML options are deactivated by default.

Other options:

- `-m/--main-names`: prints the list of the encoding main names;
- `-a/--aliases`: prints the list of the encoding aliases;
- `-A/--all-infos`: prints all the information about all the encodings;
- `-i X/--info=X`: prints all the information about the encoding X.

The encodings can take values in the following list (non exhaustive, see below):

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

12.11 Dico

Dico [OPTIONS] <dic_1> [<dic_2> <dic_3>...]

This program applies dictionaries to a text. The text must have been cut up into lexical units by the `Tokenize` program.

OPTIONS:

- `-t TXT/--text=TXT`: complete `.snt` text file name;
- `-a ALPH/--alphabet=ALPH`: the alphabet file to use;
- `-m DICS/--morpho=DICS`: this optional parameter indicates which morphological dictionaries are to be used, if needed by some `.fst2` dictionaries. `DICS` represents a list of `.bin` files (with full paths) separated with semi-colons;
- `-K/--korean`: tells `Dico` that it works on Korean;
- `-s/--semitic`: tells `Dico` that it works on a semitic language (needed if `Dico` has to compress a morphological dictionary);
- `-u X/--arabic_rules=X`: specifies the Arabic typographic rule configuration file.

<dic_i> represents the path and name of a dictionary. The dictionary must be a `.bin` dictionary (obtained with the `Compress` program) or a dictionary graph in the `.fst2` format (see section 3.6, page 63). It is possible to give priorities to the dictionaries. For details see section 3.6.1.

The program `Dico` produces the following files, and saves them in the directory of the text:

- `dlf`: dictionary of simple words in the text;
- `dlc`: dictionary of compound words in the text;
- `err`: list of unknown words in the text;
- `tags_err`: unrecognized simple words that are not matched by the `tags.ind` file;
- `tags.ind`: sequences to be inserted in the text automaton (see section 3.6.3, page 64);
- `stat_dic.n`: file containing the number of simple words, the number of compound words, and the number of unknown words in the text.

NOTE: Files `dlf`, `dlc`, `err` and `tags_err` are not sorted. Use the program `SortTxt` to sort them.

12.12 Elag

Elag [OPTIONS] <tfst>

This program takes a .tfst text automaton <tfst> and applies to it ambiguity removal rules.

OPTIONS:

- -l LANG/--language=LANG: ELAG configuration file for the language of the text;
- -r RULES/--rules=RULES: rule file compiled in the .rul format;
- -o OUT/--output=OUT: output text automaton.

12.13 ElagComp

ElagComp [OPTIONS]

This program compiles the ELAG grammar named GRAMMAR, or all the grammars specified in the RULES file. The result is stored in the OUT file that will be used by the Elag program.

OPTIONS:

- -r RULES/--rules=RULES: file listing ELAG grammars;
- -g GRAMMAR/--grammar=GRAMMAR: single ELAG grammars;
- -l LANG/--language=LANG: ELAG configuration file for the language of the grammar(s);
- -o OUT/--output=OUT: output file. By default, the output file name is the same as RULES, except for the extension that is .rul.

12.14 Evamb

Evamb [OPTIONS] <tfst>

This program computes an average lexical ambiguity rate on the text automaton <tfst>, or just on the sentence which number is specified by N. The results of the computation are displayed on the standard output. The text automaton is not modified.

OPTIONS:

- -o OUT/--output=OUT: optional output filename;
- -s N/--sentence=N: sentence number.

12.15 Extract

Extract [OPTIONS] <text>

This program extracts from the given text all sentences that contain at least one occurrence from the concordance. The parameter <text> represents the complete path of the text file, without omitting the extension `.snt`.

OPTIONS:

- `-y/--yes`: extracts all sentences containing matching units (default);
- `-n/--no`: extracts all sentences that don't contain matching units;
- `-o OUT/--output=OUT`: output text file;
- `-i X/--index=X`: the `.ind` file that describes the concordance. By default, `X` is the `concord.ind` file located in the text directory.

The result file is a text file that contains all extracted sentences, one sentence per line.

12.16 Flatten

Flatten [OPTIONS] <fst2>

This program takes a `.fst2` grammar as its parameter, and tries to transform it into a final state transducer.

OPTIONS:

- `-f/--fst`: the grammar is "unfolded" to the maximum depth and is truncated if there are calls to sub-graphs. Truncated calls are replaced by void transitions. The result is a `.fst2` grammar that only contains a single finite state transducer;
- `-r/--rtn`: calls to sub-graphs that remain after the transformation are left as they are. The result is therefore a finite state transducer in the favorable case, and an optimized grammar strictly equivalent to the original grammar if not (default);
- `-d N/--depth=N`: maximum depth to which graph calls should be unfolded. The default value is 10.

12.17 Fst2Check

Fst2Check [OPTIONS] <fst2>

This program checks if a `.fst2` file has no error for Locate.

OPTIONS:

- `-y/--loop_check`: enables error checking (loop detection);
- `-n/--no_loop_check`: disables error checking (default);
- `-t/--tfst_check`: checks whether the given graph can be considered as a valid sentence automaton or not;
- `-e/--no_empty_graph_warning`: no warning will be emitted when a graph matches the empty word. This option is used by `MultiFlex` in order not to scare users with meaningless error messages when they design an inflection grammar that matches the empty word.

Output options:

- `-o file/--output=file`: output file for error message;
- `-a/--append`: opens the message output file in append mode;
- `-s/--statistics`: displays statistics about `fst2` file.

12.18 Fst2List

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

This program takes a `.fst2` file and lists the sequences recognized by this grammar. The parameters are:

- `fname`: grammar name, including `.fst2`;
- `-o out`: specifies the output file, `lst.txt` by default;
- `-[a/t] s/m`: indicates if the program must take into account (`t`) or not (`a`) the outputs of the grammars if any. `s` indicates that there is only one initial state, whereas `m` indicates that there are several ones (this mode is useful in Korean). The default value is `-a s`;
- `-l line#`: maximum number of lines to be printed in the output file;
- `-i subname`: indicates that the recursive exploration must end when the program enters in graph `subname`. This parameter can be used several times in order to specify several stop graphs;
- `-p s/f/d`: `s` displays paths graph by graph; `f` (default) displays global paths; `d` displays global paths with information on nested graph calls;
- `-c SS=0XXXXX`: replaces symbol `SS` when it appears between angle brackets by the Unicode character whose hexadecimal number is `0XXXXX`;

- `-s "L[,R]"` : specifies the left (L) and right (R) delimiters that will enclose items. By default, no delimiters are specified;
- `-s0 "Str"` : if the program must take outputs into account, this parameter specifies the sequence `Str` that will be inserted between input and output. By default, there is no separator;
- `-f a/s` : if the program must take outputs into account, this parameter specifies the format of the lines that will be generated: `in0 in1 out0 out1(s)` or `in0 out0 in1 out1 (a)`. The default value is `s`;
- `-v` : prints information during the process (verbose mode);
- `-rx "L, [R]"` : specifies how cycles must be displayed. L and R are delimiters. If we consider the graph shown on Figure 12.4, here are the results for `L="["` and `R="]"` *":

```
il fait [très très]*
il fait très beau
```

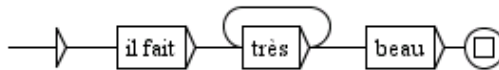


Figure 12.4: Graph with a cycle

12.19 Fst2Txt

```
Fst2Txt [OPTIONS] <fst2>
```

This program applies a transducer to a text at the preprocessing stage, when the text has not been cut into lexical units yet.

OPTIONS:

- `-t TXT/--text=TXT`: the text file to be modified, with extension `.snt`;
- `-a ALPH/--alphabet=ALPH`: the alphabet file of the language of the text;
- `-s/--start_on_space`: this parameter indicates that the search will start at any position in the text, even before a space. This parameter should only be used to carry out morphological searches;
- `-x/--dont_start_on_space`: forbids the program to match expressions that start with a space (default);
- `-c/--char_by_char`: works in character by character tokenization mode. This is useful for languages like Thai;

- `-w/--word_by_word`: works in word by word tokenization mode (default);

Output options:

- `-M/--merge`: merge transducer outputs with text inputs (default);
- `-R/--replace`: replace texts inputs with corresponding transducer outputs.

This program modifies the input text file.

12.20 Grf2Fst2

```
Grf2Fst2 [OPTIONS] <grf>
```

This program compiles a grammar into a `.fst2` file (for more details see section 6.2). The parameter `<grf>` denotes the complete path of the main graph of the grammar, without omitting the extension `.grf`.

OPTIONS:

- `-y/--loop_check`: enables error checking (loop detection);
- `-n/--no_loop_check`: disables error checking (default);
- `-a ALPH/--alphabet=ALPH`: specifies the alphabet file to be used for tokenizing the content of the grammar boxes into lexical units;
- `-c/--char_by_char`: tokenization will be done character by character. If neither `-c` nor `-a` option is used, lexical units will be sequences of any Unicode letters.
- `-d DIR/--pkgdir=DIR`: specifies the repository directory to use (see section 5.2.2, page 94).
- `-e/--no_empty_graph_warning`: no warning will be emitted when a graph matches the empty word. This option is used by `MultiFlex` in order not to scare users with meaningless error messages when they design an inflection grammar that matches the empty word.
- `-t/--tfst_check`: checks whether the given graph can be considered as a valid sentence automaton or not.
- `-s/--silent_grf_name`: does not print the graph names (needed for consistent log files across several systems).

The result is a file with the same name as the graph passed to the program as a parameter, but with extension `.fst2`. This file is saved in the same folder as `<grf>`.

12.21 ImplodeTfst

ImplodeTfst [OPTIONS] <tfst>

This program implodes the specified text automaton by merging together lexical entries which only differ in their inflectional features.

OPTIONS:

- `-o OUT/--output=OUT`: output file. By default, the input text automaton is modified.

12.22 Locate

Locate [OPTIONS] <fst2>

This program applies a grammar to a text and constructs an index of the occurrences found.

OPTIONS:

- `-t TXT/--text=TXT`: complete path of the text file, without omitting the `.snt` extension;
- `-a ALPH/--alphabet=ALPH`: complete path of the alphabet file;
- `-m DICS/--morpho=DICS`: this optional parameter indicates which morphological dictionaries are to be used, if needed by some `.fst2` dictionaries. `DICS` represents a list of `.bin` files (with full paths) separated with semi-colons;
- `-s/--start_on_space`: this parameter indicates that the search will start at any position in the text, even before a space. This parameter should only be used to carry out morphological searches;
- `-x/--dont_start_on_space`: forbids the program to match expressions that start with a space (default);
- `-c/--char_by_char`: works in character by character tokenization mode. This is useful for languages like Thai;
- `-w/--word_by_word`: works in word by word tokenization mode (default);
- `-d DIR/--sntdir=DIR`: puts produced files in `DIR` instead of the text directory. Note that `DIR` must end with a file separator (`\` or `/`);
- `-K/--korean`: tells `Locate` that it works on Korean;
- `-u X/--arabic_rules=X`: Arabic typographic rule configuration file;

- `-g X/--negation_operator=X`: specifies the negation operator to be used in `Locate` patterns. The two legal values for `X` are `minus` and `tilde` (default). Using `minus` provides backward compatibility with previous versions of `Unitex`.

Search limit options:

- `-l/--all`: looks for all matches (default);
- `-n N/--number_of_matches=N`: stops after the first `N` matches.

Maximum iterations per token options:

- `-o N/--stop_token_count=N`: stops after `N` iterations on a token;
- `-o N,M/--stop_token_count=N,M`: emits a warning after `N` iterations on a token and stops after `M` iterations.

Matching mode options:

- `-S/--shortest_matches`;
- `-L/--longest_matches` (default);
- `-A/--all_matches`.

Output options:

- `-I/--ignore`: ignore transducer outputs (default);
- `-M/--merge`: merge transducer outputs with text inputs;
- `-R/--replace`: replace texts inputs with corresponding transducer outputs;
- `-p/--protect_dic_chars`: when `-M` or `-R` mode is used, `-p` protects some input characters with a backslash. This is useful when `Locate` is called by `Dico` in order to avoid producing bad lines like:

```
3,14,.PI.NUM
```

Ambiguous output options:

- `-b/--ambiguous_outputs`: allows the production of several matches with same input but different outputs (default);
- `-z/--no_ambiguous_outputs`: forbids ambiguous outputs. In case of ambiguous outputs, one will be arbitrarily kept, depending on the internal state of the program.

Variable error options

These options have no effect if the output mode is set with `--ignore`; otherwise, they rule the behavior of the `Locate` program when an output is found that contains a reference to a variable that is not correctly defined.

- `-X/--exit_on_variable_error`: kills the program;
- `-Y/--ignore_variable_errors`: acts as if the variable has an empty content (default);
- `-Z/--backtrack_on_variable_errors`: stop exploring the current path of the grammar.

This program saves the references to the found occurrences in a file called `concord.ind`. The number of occurrences, the number of units belonging to those occurrences, as well as the percentage of recognized units within the text are saved in a file called `concord.n`. These two files are stored in the directory of the text.

12.23 LocateTfst

```
LocateTfst [OPTIONS] <fst2>
```

Applies a grammar to a text automaton, and saves the matching sequence index in a file named `concord.ind`, just as `Locate` does.

OPTIONS:

- `-t TFST/--text=TFST`: complete path of the text automaton, without omitting the `.tfst` extension;
- `-a ALPH/--alphabet=ALPH`: complete path of the alphabet file;
- `-K/--korean`: tells `LocateTfst` that it works on Korean;
- `-g X/--negation_operator=X`: specifies the negation operator to be used in `Locate` patterns. The two legal values for `X` are `minus` and `tilde` (default). Using `minus` provides backward compatibility with previous versions of `Unitex`.

Search limit options:

- `-l/--all`: looks for all matches (default);
- `-n N/--number_of_matches=N`: stops after the first `N` matches.

Matching mode options:

- `-S/--shortest_matches`;

- `-L/--longest_matches` (default);
- `-A/--all_matches`.

Output options:

- `-I/--ignore`: ignore transducer outputs (default);
- `-M/--merge`: merge transducer outputs with text inputs;
- `-R/--replace`: replace texts inputs with corresponding transducer outputs.

Ambiguous output options:

- `-b/--ambiguous_outputs`: allows the production of several matches with same input but different outputs (default);
- `-z/--no_ambiguous_outputs`: forbids ambiguous outputs. In case of ambiguous outputs, one will be arbitrarily kept, depending on the internal state of the program.

Variable error options

These options have no effect if the output mode is set with `--ignore`; otherwise, they rule the behavior of the `Locate` program when an output is found that contains a reference to a variable that is not correctly defined.

- `-X/--exit_on_variable_error`: kills the program;
- `-Y/--ignore_variable_errors`: acts as if the variable has an empty content (default);
- `-Z/--backtrack_on_variable_errors`: stop exploring the current path of the grammar.

This program saves the references to the found occurrences in a file called `concord.ind`. The number of occurrences and the number of produced outputs are saved in a file called `concord_tfst.n`. These two files are stored in the directory of the text.

12.24 MultiFlex

```
MultiFlex [OPTIONS] <delaf>
```

This program carries out the automatic inflection of a DELA dictionary containing simple (see section 3.1.2) or compound word lemmas (see chapter 10).

OPTIONS:

- `-o DELAF/--output=DELAF`: output DELAF file;

- `-a ALPH/--alphabet=ALPH`: alphabet file;
- `-d DIR/--directory=DIR`: the directory containing Morphology and Equivalences files and inflection graphs for single and compound words;
- `-K/--korean`: tells MultiFlex that it works on Korean;
- `-s/--only-simple-words`: the program will consider compound words as errors;
- `-c/--only-compound-words`: the program will consider simple words as errors;
- `-p DIR/--pkgdir=DIR`: specifies the graph repository.

Note that `.fst2` inflection transducers will automatically be built from corresponding `.grf` files if absent or older than `.grf` files.

12.25 Normalize

Normalize [OPTIONS] <text>

This program carries out a normalization of text separators. The separators are space, tab, and newline. Every sequence of separators that contains at least one newline is replaced by a unique newline. All other sequences of separators are replaced by a single space.

This program also checks the syntax of lexical tags found in the text. All sequences in curly brackets should be either the sentence delimiter {S}, the stop marker {STOP}, or valid entries in the DELAF format ({aujourd'hui, .ADV}).

Parameter <text> represents the complete path of the text file. The program creates a modified version of the text that is saved in a file with extension `.snt`.

OPTIONS:

- `-n/--no_carriage_return`: every separator sequence will be turned into a single space;
- `-r XXX/--replacement_rules=XXX`: specifies the normalization rule file to be used. See section 13.13.6 for details about the format of this file. By default, the program only replaces { and } by [and].

WARNING: if you specify a normalization rule file, its rules will be applied prior to anything else. So, you have to be very careful if you manipulate separators in such rules.

12.26 PolyLex

PolyLex [OPTIONS] <list>

This program takes a file containing unknown words <list> and tries to analyse each of the words as a compound obtained by concatenating simple words. The words that have at least one analysis are removed from the file of unknown words and the dictionary lines that correspond to the analysis are appended to file OUT.

OPTIONS:

- -a ALPH/--alphabet=ALPH: the alphabet file to use;
- -d BIN/--dictionary=BIN: .bin dictionary to use;
- -o OUT/--output=OUT: designates the file in which the produced dictionary lines are to be printed; if that file already exists, the produced lines are appended at the end of the file;
- -i INFO/--info=INFO: designates a text file in which the information about the analysis has been produced.

Language options:

- -D/--dutch
- -G/--german
- -N/--norwegian
- -R/--russian

NOTE: for Dutch or Norwegian words, the program tries to read a text file containing a list of forbidden words. This file is supposed to be named `ForbiddenWords.txt` (see section [13.13.7](#)) and stored in the same directory than BIN.

12.27 RebuildTfst

RebuildTfst <tfst>

This program reconstructs text automaton <tfst> taking into account the manual modifications. If the program finds a file `sentenceN.grf` in the same directory as <tfst>, it replaces the automaton of sentence N with the one represented by `sentenceN.grf`. The input text automaton is modified.

12.28 Reconstrucao

Reconstrucao [OPTIONS] <index>

This program generates a normalization grammar designed to be applied before the construction of an automaton for a Portuguese text. The <index> file represents a concordance which has to be produced by applying in MERGE mode to the considered text a grammar that extracts all forms to be normalized. This grammar is called V-Pro-Suf, and is stored in the /Portuguese/Graphs/Normalization directory.

OPTIONS:

- -a ALPH/--alphabet=ALPH: the alphabet file to use;
- -r ROOT/--root=ROOT: the inverse .bin dictionary to use to find forms in the future and conditional given their canonical forms. It has to be obtained by compressing the dictionary of verbs in the future and conditional with the parameter --flip (see section 12.7);
- -d BIN/--dictionary=BIN: the .bin dictionary to use;
- -p PRO/--pronoun_rules=PRO: the .fst2 grammar describing pronoun rewriting rules;
- -n PRO/--nasal_pronoun_rules=PRO: the .fst2 grammar describing nasal pronoun rewriting rules;
- -o OUT/--output=OUT: the name of the .grf graph to be generated.

12.29 Reg2Grf

Reg2Grf <txt>

This program constructs a .grf file corresponding to the regular expression written in file <txt>. The parameter <txt> represents the complete path to the file containing the regular expression. This file needs to be a Unicode text file. The program takes into account all characters up to the first newline. The result file is called `regexp.grf` and is saved in the same directory as <txt>.

12.30 SortTxt

SortTxt [OPTIONS] <txt>

This program carries out a lexicographical sorting of the lines of file <txt>. <txt> represents the complete path of the file to be sorted.

OPTIONS:

- `-n/--no_duplicates`: remove duplicate lines (default);
- `-d/--duplicates`: remove duplicate lines;
- `-r/--reverse`: sort in descending order;
- `-o XXX/--sort_order=XXX`: sorts using the alphabet of the order defined by file XXX. If this parameter is missing, the sorting is done according to the order of Unicode characters;
- `-l XXX/--line_info=XXX`: backup the number of lines of the result file in file XXX;
- `-t/--thai`: option for sorting Thai text.

The input text file is modified. By default, the sorting is performed in the order of Unicode characters, removing duplicate lines.

12.31 Stats

`Stats [OPTIONS] <ind>`

This program computes some statistics from the `<ind>` concordance index file.

OPTIONS:

- `-m MODE/--mode=MODE`: specifies the output to be produced:
 - 0 = matches with left and right contexts + number of occurrences;
 - 1 = collocates + number of occurrences;
 - 2 = collocates + number of occurrences + z-score.
- `-a ALPH/--alphabet=ALPH`: alphabet file to use;
- `-o OUT/--output=OUT`: output file;
- `-l N/--left=N`: length of left contexts in tokens;
- `-r N/--right=N`: length of right contexts in tokens;
- `-c N/--case=N`: case policy: 0 = case insensitive, 1 = case sensitive (default).

12.32 Table2Grf

Table2Grf [OPTIONS] <table>

This program automatically generates graphs from a lexicon-grammar <table> and a template graph.

OPTIONS:

- -r GRF/--reference_graph=GRF: name of the template graph;
- -o OUT/--output=OUT: name of the result main graph;
- -s XXX/--subgraph_pattern=XXX: if this optional parameter is specified, all the produced subgraphs will be named according to this pattern. In order to have unambiguous names, we recommend to include @% in the parameter (remind that @% will be replaced by the line number of the entry in the table). For instance, if you set the pattern parameter to 'subgraph-@%.grf', subgraph names will be such as 'subgraph-0013.grf'. By default, subgraph names look like 'result_0013.grf', where 'result.grf' designates the result main graph.

12.33 Tagger

Tagger [OPTIONS] <tfst>

The input of this program is the text automaton in the specified .tfst. The program applies the Viterbi-Path algorithm to it and produces a linear automaton. The automaton is pruned in a probabilistic way based on a second-order hidden Markov model. If the specified tagger data file contains tuples of "cat" tags, the tagger prunes transitions on the basis of grammatical, syntactic and semantic codes (for example, that.DET+Ddem versus that.PRO+Pdem). Else if it contains tuples of "morph" tags, so the tagger prunes transitions on grammatical, semantic, syntactic and inflectional codes (the.DET+Ddef:s versus the.DET+Ddef:p). In that case, the automaton needs to be exploded before applying the tagging process and a tagset file must be specified by the -t option below.

OPTIONS:

- -a ALPH/--alphabet=ALPH: alphabet file.
- -o OUT/--output=OUT: output text automaton.
- -t TAGSET/--tagset=TAGSET: name of the tagset description file.
- -d DATA/--data=DATA: a .bin tagger data file that contains occurrence counts for unigrams, bigrams and trigrams in order to compute probabilities. This file is obtained with the TrainingTagger program (see section [13.10.2](#)).

12.34 TagsetNormTfst

TagsetNormTfst [OPTIONS] <tfst>

This program normalizes the specified `.tfst` text automaton according to a tagset description file, discarding undeclared dictionary codes and incoherent lexical entries. Inflectional features are unfactorized so that `{rouge, .A:fs:ms}` will be divided into the 2 tags `{rouge, .A:fs}` and `{rouge, .A:ms}`.

OPTIONS:

- `-o OUT/--output=OUT`: output text automaton. By default, the input text automaton is modified;
- `-t TAGSET/--tagset=TAGSET`: name of the tagset description file.

12.35 TEI2Txt

TEI2Txt [OPTIONS] <xml>

Produces a raw text file from the given <xml> TEI file.

OPTIONS:

- `-o TXT/--output=TXT`: name of the output text file. By default, the output file has the same name than the input one, replacing `.xml` by `.txt`.

12.36 Tfst2Grf

Tfst2Grf [OPTIONS] <tfst>

This program extracts a sentence automaton in `.grf` format from the given text automaton.

OPTIONS:

- `-s N/--sentence=N`: the number of the sentence to be extracted;
- `-o XXX/--output=XXX`: pattern used to name output files `XXX.grf`, `XXX.txt` and `XXX.tok` (default=`cursor`);
- `-f FONT/--font=FONT`: sets the font to be used in the output `.grf` (default=`Times new Roman`);
- `-z N/--fontsize=N`: sets the font size (default=10).

The program produces the following files and saves them in the directory of the text:

- `cursentence.grf`: graph representing the automaton of the sentence;
- `cursentence.txt`: text file containing the sentence;
- `cursentence.tok`: text file containing the numbers of the tokens that compose the sentence.

12.37 Tfst2Unambig

`Tfst2Unambig [OPTIONS] <tfst>`

This program takes a `.tfst` text automaton and produces an equivalent text file if the automaton is linear (i.e. with no ambiguity). See section 7.6, page 175.

OPTIONS:

- `-o TXT/--out=TXT`: the output text file.

12.38 Tokenize

`Tokenize [OPTIONS] <txt>`

This program tokenizes a text file into lexical units. `<txt>` the complete path of the text file, without omitting the `.snt` extension.

OPTIONS:

- `-a ALPH/--alphabet=ALPH`: alphabet file;
- `-c/--char_by_char`: indicates whether the program is applied character by character, with the exceptions of the sentence delimiter `{S}`, the stop marker `{STOP}` and lexical tags like `{today, .ADV}` which are considered to be single units;
- `-w/--word_by_word`: with this option, the program considers a unit to be either a sequence of letters (the letters are defined by file `alphabet`), or a character which is not a letter, or the sentence separator `{S}`, or a lexical label like `{aujourd'hui, .ADV}`. This is the default mode.
- `-t TOKENS/--tokens=TOKENS`: specifies a `tokens.txt` file to load and modify, instead of creating a new one from scratch.

The program codes each unit as a whole. The list of units is saved in a text file called `tokens.txt`. The sequence of codes representing the units now allows the coding of the text. This sequence is saved in a binary file named `text.cod`. The program also produces the following four files:

- `tok_by_freq.txt`: text file containing the units sorted by frequency;

- `tok_by_alph.txt`: text file containing the units sorted alphabetically;
- `stats.n`: text file containing information on the number of sentence separators, the number of units, the number of simple words and the number of numbers;
- `enter.pos`: binary file containing the list of newline positions in the text. The coded representation of the text does not contain newlines, but spaces. Since a newline counts as two characters and a space as a single one, it is necessary to know where newlines occur in the text when the positions of occurrences located by the `Locate` program are to be synchronized with the text file. File `enter.pos` is used for this by the `Concord` program. Thanks to this, when clicking on an occurrence in a concordance, it is correctly selected in the text. File `enter.pos` is a binary file containing the list of the positions of newlines in the text.

All produced files are saved in the text directory.

12.39 TrainingTagger

```
TrainingTagger [OPTIONS] <txt>
```

This program automatically generates two tagger data files from a tagged corpus text file. They are used by the `Tagger` program in order to compute probabilities and linearize the text automaton. The tagged corpus file must follow the format described in section 13.10.1. Those files contain tuples (unigrams, bigrams and trigrams), formed by tags and words. In the first data file, tags are "cat" tags (i.e. grammatical, syntactic and semantic codes). In the second data file, tags are "morph" tags (i.e. grammatical, syntactic, semantic and inflectional codes).

OPTIONS:

- `-a/--all`: indicates whether the program should produce all data files (default);
- `-c/--cat`: indicates whether the program should produce only data file with "cat" tags;
- `-m/--morph`: indicates whether the program should produce only data file with "morph" tags;
- `-n/--no_binaries`: indicates whether the program should not compress data files into `.bin` files, in this case only `.dic` data files are generated;
- `-b/--binaries`: indicates whether the program should compress data files into `.bin` files (default);
- `-o XXX/--output=XXX`: pattern used to name output tagger data files `XXX_data_cat.bin` and `XXX_data_morph.bin` (default=filename of text corpus without extension);
- `-s/--semitic`: indicates that the semitic compression algorithm should be used.

12.40 Txt2Tfst

`Txt2Tfst [OPTIONS] <txt>`

This program constructs an automaton of a text. `<txt>` represents the complete path of a text file without omitting the `.snt` extension.

OPTIONS:

- `-a ALPH/--alphabet=ALPH`: alphabet file;
- `-c/--clean`: indicates whether the rule of conservation of the best paths (see section 7.2.4) should be applied;
- `-n XXX/--normalization_grammar=XXX`: name of a normalization grammar that is to be applied to the text automaton;
- `-t TAGSET/--tagset=TAGSET`: Elag tagset file to use to normalize dictionary entries;
- `-K/--korean`: tells `Txt2Tfst` that it works on Korean.

If the text is separated into sentences, the program constructs an automaton for each sentence. If this is not the case, the program arbitrarily cuts the text into sequences of 2000 lexical units and produces an automaton for each of these sequences.

The result is a file called `text.tfst` which is saved in the directory of the text. Another file named `text.tind` is also produced.

NOTE: The program will also try to use the `tags.ind` file, if any (see section 13.7.4).

12.41 Uncompress

`Uncompress [OPTIONS] <bin>`

This program uncompresses a `.bin` dictionary into a text file `.dic` one.

OPTIONS:

- `-o OUT/--output=OUT`: optional output file name (default: `file.bin>file.dic`).

12.42 Untokenize

```
Untokenize [OPTIONS] <txt>
```

Untokenizes and rebuild the original text. The token list is stored into `tokens.txt` and the coded text is stored into `text.cod`. The file `enter.pos` contains the position in tokens of all the carriage return sequences. These files are located in the `XXX_snt` directory where `XXX` is `<txt>` without its extension.

OPTIONS:

- `-d X/--sntdir=X`: uses directory `X` instead of the text directory; note that `X` must be (back)slash terminated
- `-n N/--number_token=N`: adds tokens number each `N` token;
- `-r N/--range=N`: emits only token from number `N` to end;
- `-r N,M/--range=N,M`: emits only token from number `N` to `M`.

12.43 UnitexTool

```
UnitexTool <utilities>
```

This program is a super-program that allows you to invoke all Unitex external programs. With it, you can chain commands so that they will be invoked within a same system process, in order to speed up processing. This can be done by invoking commands nested in round brackets as this:

```
UnitexTool { cmd #1+args } { cmd #2+args } etc.
```

For instance, if you want to join a locate operation and the construction of the concordance, you can use the following command:

```
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\del-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" }
```


12.44 UnitexToolLogger

UnitexToolLogger <utilities>

This program is a superset of UnitexTool that allows execute again an .ulp logfile. It can also record a running session of an UnitexTool and create an ulp logfile. If UnitexToolLogger is used like UnitexTool (with just parameter with command line for Unitex external program), and a file named `unitex_logging_parameters_count.txt` (in current directory) contain a path, an .ulp logfile will be created for the running session. The .ulp file in an uncompressed zipfile (compatible with unzip), and can be useful for debugging.

```
UnitexToolLogger RunLog [OPTIONS] <ulp>
```

OPTIONS after RunLog:

- `-m/--quiet`: do not emit message when running;
- `-v/--verbose`: emit message when running;
- `-d DIR/--rundir=DIR`: path where log is executed;
- `-r newfile.ulp/--result=newfile.ulp`: name of result ulp created;
- `-c/--clean`: remove work file after execution;
- `-k/--keep`: keep work file after execution;
- `-s file.txt/--summary=file.txt`: summary file with log compare result to be created;
- `-e file.txt/--summary-error=file.txt`: summary file with error compare result to be created;
- `-b/--no-benchmark`: do not store time execution in result log;
- `-n/--cleanlog`: remove result ulp after execution;
- `-l/--keeplog`: keep result ulp after execution;
- `-o NameTool/--tool=NameTool`: run only log for NameTool;
- `-i N/--increment=N`: increment filename <ulp> by 0 to N;
- `-t N/--thread=N`: create N thread;
- `-a N/--random=N`: select N time a random log in the list (in each thread);
- `-f N/--break-after=N`: user cancel after N run (with one thread only);
- `-u PATH/--unfound-location=PATH`: take dictionary and FST2 from PATH if not found on the logfile;

Another usage of `UnitexToolLogger` is using the `MzRepairUlp` option to repair a corrupted ulp file (often, a crashing log):

```
UnitexToolLogger MzRepairUlp [OPTIONS] <ulpfile>
```

OPTIONS after MzRepairUlp:

- `-t X/--temp=X`: uses `X` as filename for temporary file (`<ulpfile>.build` by default);
- `-o X/--output=X`: uses `X` as filename for fixed `.ulp` file (`<ulpfile>.repair` by default);
- `-m/--quiet`: do not emit message when running;
- `-v/--verbose`: emit message when running;

Another usage of `UnitexToolLogger` is using the `CreateLog` option (with round bracket) to create logfile of running Unitex program, like:

```
UnitexToolLogger { CreateLog [OPTIONS] } cmd args
```

```
UnitexToolLogger { CreateLog [OPTIONS] } { cmd #1+args } { cmd #2+args } etc.
```

By example,

```
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 after CreateLog:

- `-g/--no_create_log`: do not create any log file. Incompatible with all others options;
- `-p XXX/--param_file=XXX`: load a parameters file like `unitex_logging_parameters.txt`. Incompatible with all others options;
- `-d XXX/--directory=XXX`: location directory where log file to create;
- `-l XXX/--log_file=XXX`: filename of log file to create;
- `-i/--store_input_file`: store input file in log (default);
- `-n/--no_store_input_file`: don't store input file in log (prevent rerun the log-file);
- `-o/--store_output_file`: store output file in log;

- `-u/--no_store_output_file`: don't store output file in log (default);
- `-s/--store_list_input_file`: store list of input file in log (default);
- `-t/--no_store_list_input_file`: don't store list of input file in log;
- `-r/--store_list_output_file`: store list of output file in log (default);
- `-f/--no_store_list_output_file`: don't store list of output file in log.

12.45 XMLizer

XMLizer [OPTIONS] <txt>

This program takes the raw text file <txt> and produces a corresponding basic TEI or XML file. The difference between TEI and XML is that TEI files will contain a TEI header.

OPTIONS:

- `-x/--xml`: produces a XML file;
- `-t/--tei`: produces a TEI file (default);
- `-n XXX/--normalization=XXX`: specify the normalization rule file to be used (see section 13.13.6);
- `-o OUT/--output=OUT`: optional output file name (default: `file.txt > file.xml`);
- `-a ALPH/--alphabet=ALPH`: alphabet file;
- `-s SEG/--segmentation_grammar=SEG`: sentence delimitation grammar to be used. This grammar should be like the `Sentence.grf` one used during the preprocessing of a corpus, but it can include the special tag `{P}` to indicate paragraph bounds.

Chapter 13

File formats

This chapter presents the formats of files read or generated by Unitex. The formats of the DELAS and DELAF dictionaries have already been presented in sections 3.1.1 and 3.1.2.

NOTE: In this chapter the symbol ¶ represents the newline symbol. Unless otherwise indicated, all text files described in this chapter are encoded in Unicode Little-Endian.

13.1 Unicode encoding

By default, text files processed by Unitex have to be encoded in Unicode Little-Endian. Unitex accepts also Unicode Big-Endian or UTF-8 files. This encoding allows the representation of 65536 characters by coding each of them in 2 bytes. In Little-Endian, the bytes are in lo-byte hi-byte order. If this order is reversed, we speak of Big-Endian. A text file encoded in Unicode Little-Endian, Big-Endian or UTF-8 starts with the special character (Unicode Byte Order Mark - BOM) with the hexadecimal value FF FE (Little-Endian), FE FF (Big-Endian) or EF BB BF (UTF-8). The newline symbols have to be encoded by the two characters 0D 00 and 0A 00 (Little-Endian), 00 0D and 00 0A (Big-Endian), or 0D and 0A (UTF-8).

Consider the following text:

```
Unitex¶  
β-version¶
```

Here is its representation in Unicode Little-Endian:

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 13.1: Hexadecimal representation of a Unicode Little-Endian text

Here is its representation in 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 13.2: Hexadecimal representation of a Unicode Big-Endian text

Here is its representation in Unicode 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 13.3: Hexadecimal representation of a Unicode UTF-8 text

On Unicode Little-Endian, the hi-bytes and lo-bytes have been reversed, which explains why the start character is encoded as FF FE in stead of FE FF, and 00 0D and 00 0A are 0D 00 and 0A 00 respectively.

13.2 Alphabet files

There are two kinds of alphabet files: a file which defines the characters of a language, and a file that indicates the sorting preferences. The first is designed under the name *alphabet*, the second under the name *sorted alphabet*.

13.2.1 Alphabet

The alphabet file is a text file that describes all characters of a language, as well as the correspondances between capitalized and non-capitalized letters. This file is called `Alphabet.txt` and is found in the root of the directory of a language. Its presence is obligatory for Unitex to function.

Example: the English alphabet file has to be in the directory `.../English/`

Each line of the alphabet file must have one of the following three forms, followed by a newline symbol:

- #가힐 : a hash symbol followed by two characters X and Y which indicate that all characters between X and Y are letters. All these characters are considered to be in non-capitalized and capitalized form at the same time. This method is used to define the alphabets of Asian languages like Korean, Chinese or Japanese where there is no distinction between upper- and lower-case, and where the number of characters makes a complete enumeration tedious;
- Aa : two characters X and Y indicate that X and Y are letters and that X is a capitalized equivalent of the non-capitalized Y form.
- ㄱ: a unique character X defines X as a letter in capitalized and non-capitalized form. This form is used to define a single Asian character.

For certain languages like French, it is possible that a lower-case letter corresponds to multiple upper-case letters. For example, \acute{e} , in practice, can have the upper-case form E or \acute{E} . To express this, it suffices to use multiple lines. The reverse is equally true: a capitalized letter can correspond to multiple lower-case letters. Thus, E can be the capitalization of e , \acute{e} , \grave{e} , $\grave{\text{e}}$ or \hat{e} . Here is an excerpt of the French alphabet file which defines different properties of letter e :

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

13.2.2 Sorted alphabet

The sorted alphabet file defines the sorting priorities of the letters of a language. It is used by the `SortTxt` program. Each line of that file defines a group of letters. If a group of letters A is defined before a group of letters B , every letter of group A is inferior to every letter in group B .

The letters of a group are only distinguished if necessary. For example if the group of letters $e\acute{e}\grave{e}\grave{\text{e}}\hat{e}$ has been defined, the word `ébahi` should be considered 'smaller' than `estuaire`, and also 'smaller' than `été`. Since the letters that follow e and \acute{e} determine the order of the words, it is not necessary to compare letters e and \acute{e} since they are of the same group. On the other hand, if the words `chantés` and `chantes` are to be sorted, `chantes` should be considered as 'smaller'. It is therefore necessary to compare the letters e and \acute{e} to distinguish these words. Since the letter e appears first in the group $e\acute{e}\grave{e}\grave{\text{e}}\hat{e}$, it is considered to be

'smaller' than `chantés`. The word `chantes` should therefore be considered to be 'smaller' than the word `chantés`.

The sorted alphabet file allows the definition of equivalent characters. It is therefore possible to ignore the different accents as well as capitalization. For example, if the letters `b`, `c`, and `d` are to be ordered without considering capitalization and the cedilla, it is possible to write the following lines:

```
Bb
CcÇç
Dd
```

This file is optional. If no sorted alphabet file is specified, the `SortTxt` program sorts in the order of the Unicode encoding.

13.3 Graphs

This section presents the two graph formats: the graphic format `.grf` and the compiled format `.fst2`.

13.3.1 Format `.grf`

A `.grf` file is a text file that contains presentation information in addition to information representing the contents of the boxes and the transitions of the graph. A `.grf` file begins with the following lines:

```
#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
#
```


The first line `#Unigraph` is a comment line. The following lines define the parameter values of the graph presentation:

- `SIZE x y` : defines the width x and the height y of a graph in pixels;
- `FONT name:xyz` : defines the font used for displaying the contents of the boxes. `name` represents the name of the mode. x indicates if the text should be in bold face or not. If x is `B`, it indicates that it should be bold. For non-bold face, x should be a space. In the same way, y has value `I` if the text should be italic, a space if not. z represents the size of the text;
- `OFONT name:xyz` : defines the mode used for displaying transducer outputs. Parameters `name`, x , y , and z are defined in the same way as `FONT`;
- `BCOLOR x` : defines the background color of the graph. ' x ' represents the color in RGB format;
- `FCOLOR x` : defines the foreground color of the graph. ' x ' represents the color in RGB format;
- `ACOLOR x` : defines the color inside the boxes that correspond to the calls of sub-graphs. x represents the color in RGB format;
- `SCOLOR x` : defines the color used for writing in comment boxes (boxes that are not linked up with any others). x represents the color in RGB format;
- `CCOLOR x` : defines the color used for designing selected boxes. x represents the color in RGB format;
- `DBOXES x` : this line is ignored by Unitex. It is conserved to ensure compatibility with Intex graphs;
- `DFRAME x` : there will be a frame around the graph if x is `y`, not if it is `n`;
- `DDATE x` : puts the date at the bottom of the graph if x is `y`, not if it is `n`;
- `DFILE x` : puts the name of the file at the bottom of the graph depending on whether x is `y` or `n`;
- `DDIR x` : prints the complete path of the graph whether x is `y` or `n`. This option has no effect if the `DFILE` option is set to `n`;
- `DRIG x` : displays the graph from right to left or left to right depending on whether x is `y` or `n`;
- `DRST x` : this line is ignored by Unitex. It is conserved to ensure compatibility with Intex graphs;
- `FITS x` : this line is ignored by Unitex. It is conserved to ensure compatibility with Intex graphs;

- `PORIENT x` : this line is ignored by Unitex. It is conserved to ensure compatibility with Intex graphs;
- `#` : this line is ignored by Unitex. It serves to indicate the end of the header information.

The lines after the header give the contents and the position of the boxes in the graph. The following example corresponds to a graph recognizing a number:

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

The first line after the header indicates the number of boxes in the graph, immediately followed by a newline. This number can not be lower than 2, since a graph always has an initial and a final state.

The following lines define the boxes of the graph. The boxes are numbered starting at 0. By convention, state 0 is the initial state and state 1 is the final state. The contents of the final state is always empty.

Each box in the graph is defined by a line that has the following format:

contents *X Y N transitions*

contents is a sequence of characters enclosed in quotation marks that represents the contents of the box. This sequence can sometimes be preceded by an `s` if the graph is imported from Intex; this character is then ignored by Unitex. The contents of the sequence is the text that has been entered in the editing line of the graph editor. Table 13.4 shows the encoding of two special sequences that are not encoded in the same way as they are entered into the `.grf` files:

Sequence in the graph editor	Sequence in the <code>.grf</code> file
"	\ "
\ "	\\ \"

Table 13.4: Encoding of special sequences

NOTE: The characters between `<` and `>` or between `{` and `}` are not interpreted. Thus the `+` character in sequence `"le <A+Conc>"` is not interpreted as a line separator, since the pattern `<A+Conc>` is interpreted with priority.

`X` and `Y` represent the coordinates of the box in pixels. Figure 13.1 shows how these coordinates are interpreted by Unitex.

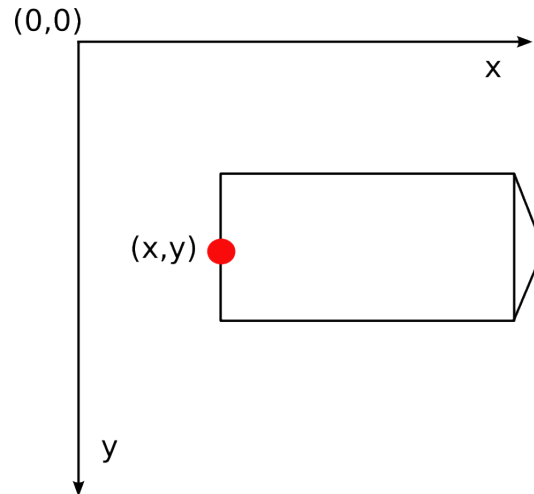


Figure 13.1: Interpretation of the coordinates of boxes

N represents the number of outgoing transitions of the box. This number is always 0 for the final state.

The transitions are defined by the number of their target box.

Every line of the box definition ends with a newline.

13.3.2 Format .fst2

An `.fst2` file is a text file that describes a set of graphs. Here is an example of an `.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
```

The first line represents the number of graphs that are encoded in the file. The beginning of each graph is identified by a line that indicates the number and the name of the graph (-1 NP and -2 Adj in the file above).

The following lines describe the states of the graph. If the state is final, the line starts with the `t` character and with the `:` character if not. For each state, the list of transitions is a possibly empty sequence of pairs of integers:

- the first integer indicates the number of the label or sub-graph that corresponds to the transition. Labels are numbered starting at 0. Sub-graphs are represented by negative integers, which explains why the numbers preceding the names of the graphs are negative;
- the second integer represents the number of the result state after the transition. In each graph, the states are numbered starting at 0. By convention state 0 is the initial state.

Each state definition line terminates with a space. The end of each graph is marked by a line containing an `f` followed by a space and a newline.

Labels are defined after the last graph. If the line begins with the `@` character, the contents of the label is to be searched without allowing case variations. This information is not used if the label is not a word. If the line starts with a `%`, capitalization variants are authorized. If a label carries a transducer output sequence, the input and output sequences are separated by the `/` character (example: `the/DET`). By convention, the first label is always the empty word (`<E>`), even if that label is never used for any transition.

The end of the file is indicated by a line containing the `f` character followed by a newline.

13.4 Texts

This section presents the different files used to represent texts.

13.4.1 .txt files

`.txt` files are text files encoded in Unicode Little-Endian. These files should not contain any opening or closing braces, except for those used to mark a sentence separator (`{S}`) or a valid lexical tag (`{aujourd'hui, .ADV}`). The newline needs to be encoded with the two special characters with hexadecimal values `000D` and `000A`.

13.4.2 .snt files

.snt files are .txt files that have been processed by Unitex. These files should not contain any tabs. They should also not contain multiple consecutive spaces or newlines. The only allowed braces in .snt files are those of the sentence delimiter {S} and those of lexical labels ({aujourd'hui, .ADV}).

13.4.3 File text.cod

The text.cod file is a binary file containing a sequence of integers that represent the text. Each integer *i* reflects the token with index *i* in the tokens.txt file. These integers are encoded in four bytes.

NOTE: Tokens are numbered starting at 0.

13.4.4 The tokens.txt file

The tokens.txt file is a text file that contains the list of all lexical units of the text. The first line of this file indicates the number of units found in the file. Units are separated by a newline. Whenever a sequence is found in the text with capitalization variants, each variant is encoded as a distinct unit.

NOTE: Newlines that might be in the .snt file are encoded like spaces. Therefore there is no unit encoding the newline.

13.4.5 The tok_by_alph.txt and tok_by_freq.txt files

These two files are text files that contain the list of lexical units sorted alphabetically or by frequency.

In the tok_by_alph.txt file, each line is composed by a unit, followed by a tab and the number of occurrences of the unit within the text.

The lines of the tok_by_freq.txt file are formed after the same principle, but the number of occurrences is placed after the tab and the unit.

13.4.6 The enter.pos file

This file is a binary file containing the list of positions of the newline symbol in the .snt file. Each position is the index in the text.cod file where a newline has been replaced by a space. These positions are integers that are encoded in 4 bytes.

13.5 Text Automaton

13.5.1 The `text.tfst` file

The `text.tfst` file represents the text automaton. It is a text file that starts with a ten digit line indicating the number of sentence automata it contains. Then, for each sentence automaton, you have the following header lines:

- `$XXX¶`: `XXX` = number of the sentence
- `foo foo foo...¶`: text of the sentence
- `a/b c/d e/f g/h...¶`: for each token of the sentence, we have a pair `x/y`: `x` is the token index in file `tokens.txt`, `y` is the length of the token in characters
- `X_Y¶`: `X` is the offset of the first token of the sentence, in tokens from the beginning of the text; `Y` is the same, but the offset is in characters from the beginning of the text.

Then, all states of the automaton are encoded, one per line. If the state is final, the line starts with `t`. Otherwise, the line starts with `:`. All transitions are written as pairs `x y`, `x` being the number of the tag, `y` being the number of the destination state. Note that, at the opposite of `.fst2` format, lines have not to end with a space. The end of state lines is marked by a line containing `f`.

Finally, all tags are encoded. By convention, the first tag is always the epsilon one:

```
@<E>¶
.¶
```

Other labels have to be either lexical units or entries in the DELAF format in braces. They are encoded as follows:

```
@STD¶
@content¶
@a.b.c-x.y.z¶
.¶
```

`content` is the tag content. The `a.b.c-x.y.z` information describe the zone in text covered by the tag:

- `a`: start offset in tokens from the beginning of the sentence;
- `b`: start offset in characters from the beginning of the first token of the tag;
- `c`: start offset in logical letters from the first character of the tag. This information is useful for Korean, because a tag can represent a Jamo sequence that occurs inside a Hangul character. Thus, the character offset is not precise enough;

- x : end offset in tokens from the beginning of the sentence;
- y : end offset in characters from the beginning of the last token of the tag;
- z : end offset in logical letters from the last character of the tag. In Korean sentence automata, empty surface forms can occur that correspond to the empty word in the text. In such cases, z has special value -1 .

The end of tag definitions is marked by a line containing f .

Example: Here is the file that corresponds to the text *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¶

```

13.5.2 The text.tind file

The `text.tind` file is an index file used to jump at correct byte offset in the `text.tfst` file when we want to load a given sentence. It is a binary file that contains $4 \times N$ bytes, where N is the number of sentences. It gives the start offset of each sentence as a 4-byte little-endian sequence.

13.5.3 The cursentence.grf file

The `cursentence.grf` file is generated by Unitex during the display of a sentence automaton. The `Fst2Grf` program constructs a `.grf` file from the `text.fst2` file that represents a sentence automaton.

NOTE: outputs of graph boxes are used to encode offsets, as defined in `.tfst` tags. Offsets are separated with spaces. For instance, here are some lines of the graph representing the first sentence of *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 ¶
```

13.5.4 The sentenceN.grf file

Whenever the user modifies a sentence automaton, that automaton is saved under the name `sentenceN.grf`, where `N` represents the number of the sentence. Such a graph contains offsets in graph box outputs (see note in section 13.5.3).

13.5.5 The cursentence.txt file

During the extraction of the sentence automaton, the text of the sentence is saved in the file called `cursentence.txt`. That file is used by Unitex to display the text of the sentence under the automaton. That file contains the text of the sentence, followed by a newline.

13.5.6 The cursentence.tok file

During the extraction of the sentence automaton, the numbers of the tokens that compose the sentence are stored in a file named `cursentence.tok`. This file contains one line per token, each line being made of 2 integers `x y`: `x` is the token number, `y` is the length of the token in characters.

Here is the content of this file for the first sentence of *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¶      _
```

13.5.7 The `tfst_tags_by_freq.txt` and `tfst_tags_by_alph.txt` files

Those files contain all the tags that appear in the text automaton sorted by frequency and alphabetical order.

13.6 Concordances

13.6.1 The `concord.ind` file

The `concord.ind` file is the index of the occurrences found by either `Locate` or `LocateTfst` during the application of a grammar. It is a text file that contains the starting and ending positions of each occurrence, possibly accompanied by a sequence of letters if the construction of the concordance took into account the possible transducer outputs of the grammar. Here is an example of such a file:

```
#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
```

The first line indicates in which transduction mode the concordance has been constructed. The three possible values are:

- `#I` : transducer outputs have been ignored;
- `#M` : transducer outputs have been inserted before the corresponding inputs (MERGE mode);
- `#R` : transducer outputs have replaced the recognized sequences (REPLACE mode).

Each occurrence is described in one line. The lines start with the start and end positions of the occurrence. These positions corresponds to the offsets defined in `.tfst` tags (see 13.5.1).

If the file has the heading line #1, the end position of each occurrence is immediately followed by a newline. Otherwise, it is followed by a space and a sequence of characters. In REPLACE mode, that sequence corresponds to the output produced for the recognized sequence. In MERGE mode, it represents the recognized sequences into which the outputs have been inserted. In MERGE or REPLACE mode, this sequence is displayed in the concordance. If the outputs have been ignored, the contents of the occurrence is extracted from the text file.

13.6.2 The concord.txt file

The `concord.txt` file is a text file that represents a concordance. Each occurrence is encoded in a line that is composed of three character sequences separated by a tab, representing the left context, the occurrence (possibly modified by transducer outputs) and the right context.

13.6.3 The concord.html file

The `concord.html` file is an HTML file that represents a concordance. This file is encoded in UTF-8.

The title of the page is the number of occurrences it describes. The lines of the concordance are encoded as lines where the occurrences are considered to be hypertext lines. The reference associated to each of these lines has the following form: ``. X and Y represent the start and end position of the occurrence in characters in the file `name_of_text.snt`. Z represents the number of the phrase in which this occurrence appears.

All spaces that are at the left and right edges of lines are encoded by a non breaking space (` ` in HTML), which allows the preservation of the alignment of the utterances even if one of them has a left context with spaces.

NOTE: If the concordance has been constructed with the `glossanet` parameter, the HTML file has the same structure, except for the links. In these concordances, the occurrences are real links pointing at the web server of the GlossaNet application. For more information on GlossaNet, consult the link on the Unitex web site.

Here is an example of a file:

```
<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>&nbsp;i&nbsp;<br>
&nbsp;extended <a href="125 127 2">in</a>&nbsp;ancient&nbsp;<br>
&nbsp;Scott {S}<a href="32 34 2">IN</a>&nbsp;THAT PL&nbsp;<br>
STRICT of <a href="61 66 2">merry</a>&nbsp;Engl&nbsp;<br>
S}IN THAT <a href="40 48 2">PLEASANT</a>&nbsp;D&nbsp;<br>
&nbsp;which is <a href="84 91 2">watered</a>&nbsp;by&nbsp;<br>
</font>
</td></table></body>
</html>

```

Figure 13.2 shows the page that corresponds to the file below.

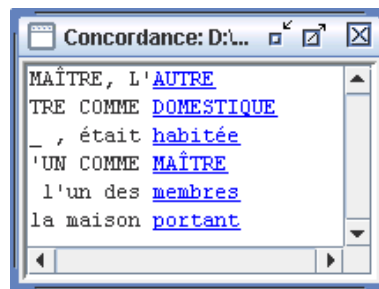


Figure 13.2: Example of a concordance

13.6.4 The diff.html file

The `diff.html` file is an HTML file that presents the differences between two concordances. This file is encoded in UTF-8. Here is an example of file (new lines have been introduced for presentation convenience):

```

<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>&nbsp;  of the beautiful hills </font>
</td>
<td width="450"><font color="green"></font></td>
</tr>
</table>
</body>
</html>

```

13.7 Text dictionaries

The `Dico` program produces several files that represent text dictionaries.

13.7.1 `dlf` and `dlc`

`dlf` and `dlc` are simple and compound word dictionaries in the DELAF format (see section 3.1.1).

13.7.2 `err`

This file is made of unknown words, one per line.

13.7.3 `tags_err`

This file is made of unknown words, one per line. The difference with the `err` file is that in this one do not appear simple words that have been matched in the `tags.ind` file.

13.7.4 `tags.ind`

This file has the same format than a `concord.ind` one obtained in MERGE or REPLACE mode, but its header is `#T`. Note that the outputs DO NOT BEGIN with a slash.

13.8 Dictionaries

The compression of the DELAF dictionaries by the `Compress` program produces two files: a `.bin` file that represents the minimal automaton of the inflected forms of the dictionaries, and a `.inf` file that contains the compressed forms required for the construction of the dictionaries from the inflected forms. This section describes the format of these two file types, as well as the format of the `CHECK_DIC.TXT` file, which contains the result of the verification of a dictionary.

13.8.1 The `.bin` files

A `.bin` file is a binary file that represents an automaton. The first 4 bytes of the file represent an integer that indicates the size of the file in bytes. The states of the automaton are encoded in the following way:

- the first two bytes indicate if the state is final as well as the number of its outgoing transitions. The highest bit is 0 if the state is final, 1 if not. The other 15 bits encode the number of transitions.

Example: a non-final state with 17 transitions is encoded by the hexadecimal sequence `8011`

- if the state is final, the three following bytes encode the index in the `.inf` file of the compressed form to be used to reconstruct the dictionary lines for this inflected form.

Example: if the state refers to the compressed form with index 25133, the corresponding hexadecimal sequence is `00622D`

- each leaving transition is then encoded in 5 bytes. The first 2 bytes encode the character that labels the transition, and the three following encode the byte position of the result state in the `.bin` file. The transitions of a state are encoded next to each other.

Example: a transition that is labeled with the `A` letter and goes to the state of which the description starts at byte 50106, is represented by the hexadecimal sequence `004100C3BA`.

By convention, the first state of the automaton is the initial state.

13.8.2 The `.inf` files

A `.inf` file is a text file that describes the compressed files that are associated to a `.bin` file. Here an example of a `.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¶
```

The first line of the file indicates the number of compressed forms that it contains. Each line can contain one or more compressed forms. If there are multiple forms, they are separated by commas. Each compressed form is made up of a sequence required to reconstruct a canonical knowing an inflected form, followed by a sequence of grammatical, semantic and inflection codes that are associated to the entry.

The mode of compression of the canonical form varies in function of the inflected form. If the two forms are identical, the compressed form contains only the grammatical, semantic and inflectional information as in:

```
.N+Hum:ms
```

If the forms are different, the compression program cuts up the two forms in units. These units can be a space, a hyphen, or a sequence of characters that contains neither a space nor a hyphen. This way of cutting up units allows the program to efficiently take into account the inflected forms of the compound words.

If the inflected and the canonical form do not have the same number of units, the program encodes the canonical form by the number of characters to be removed from the inflected form followed by the characters to append. For instance, the line below is a line in the initial dictionary:

```
James Bond,007.N
```

Since the sequence `James Bond` contains three units and `007` only one, the canonical form is encoded with `_10\0\0\7`. The `_` character indicates that the two forms do not have the same number of units. The following number (here 10) indicates the number of characters to be removed. The sequence `\0\0\7` indicates that the sequence `007` should be appended. The digits are preceded by the `\` character so they will not be confused with the number of characters to be removed.

Whenever the two forms have the same number of units, the units are compressed two by two. Each pair consists of a unit the inflected form and the corresponding unit in the canonical form. If each of the two units is a space or a hyphen, the compressed form of the unit is the unit itself, as in the following line:

```
0-1.N:p
```

which is the output for `battle-axes,battle-axe.N:p`

This maintains a certain readability of the `.inf` file when the dictionary contains compound words.

Whenever one or both of the units in a pair is neither a space nor a hyphen, the compressed form is composed of the number of characters to be removed followed by the sequence of characters to be appended. Thus, the dictionary line:

```
première partie,premier parti.N+AN+Hum:fs
```

is encoded by the line:

```
3er 1.N+AN+Hum:fs
```

The `3er` code indicates that 3 characters are to be removed from the sequence `première` and the characters `er` are to be appended to obtain `premier`. The `1` indicates that only one character needs to be removed from `partie` to obtain `parti`. The number `0` is used whenever it needs to be indicated that no letter should be removed.

13.8.3 Dictionary information file

In the "Apply lexical resources" frame, it is possible for some dictionaries to get some information with a right click. Such information is attached to a `biniou.bin` or `biniou.fst2` dictionary by the mean of a raw text file named `biniou.txt`, located in the same directory.

13.8.4 The CHECK_DIC.TXT file

This file is produced by the dictionary verification program `CheckDic`. It is a text file that contains information about the analysed dictionary and has four parts.

The first part is the possibly empty list of all syntax errors found in the dictionary: absence of the inflected or the canonical form, the grammatical code, empty lines, etc. Each error is described by the number of the line, a message describing the error, and the contents of the line. Here is an example of a message:

```
Line 12451: unexpected end of line
garden,N:s
```

The second and third parts display the list of grammatical codes and/or semantic and inflectional codes respectively. In order to prevent coding errors, the program reports encodings that contain spaces, tabs, or non-ASCII characters. For instance, if a Greek dictionary contains the `ADV` code where the Greek `Α` character is used instead of the Latin `A` character, the program reports the following warning:

```
ADV warning: 1 suspect char (1 non ASCII char): (0391 D V)
```


Non-ASCII characters are indicated by their hexadecimal character number. In the example below, the code 0391 represents Greek Α. Spaces are indicated by the SPACE sequence:

```
Km s warning: 1 suspect char (1 space): (K m SPACE s)
```

When the following dictionary is checked:

```
1,2 et 3!,.INTJ
abracadabra,INTJ
supercalifragilisticexpialidocious,.INTJ
damned,. INTJ
Paul,.N+Hum+Hum
eat,.V:W:P1s:Ps:P1p:P2p:P3p
```

the following CHECK_DIC.TXT file is obtained:

```
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  ----¶
-----¶

```

Note that the inflectional codes of `eat` are not reported, since an error occurred in this line.

13.9 ELAG files

13.9.1 tagset.def file

See section 7.3.6, page 163.

13.9.2 .lst files

.LST FILES ARE NOT UNICODE FILES.

A `.lst` file contains a list of `.grf` file names. If a file's path is not absolute, it is relative to the location of the `elag.lst` file. Here is the `elag.lst` file used for French:

```

PPVs/PpvIL.grf¶
PPVs/PpvLE.grf¶
PPVs/PpvLUI.grf¶
PPVs/PpvPR.grf¶
PPVs/PpvSeq.grf¶
PPVs/SE.grf¶
PPVs/postpos.grf¶

```

13.9.3 .elg files

`.elg` files contain compiled ELAG rules. These files are in the `.fst2` format.

13.9.4 .rul files

.RUL FILES ARE NOT UNICODE FILES.

A .rul file contains the different .elg files that compose an ELAG rule set. It contains one part per .elg file. Each part lists the ELAG grammars that correspond to a given .elg file. .elg file names are surrounded with angles brackets. The lines that start with a tabulation are considered as comments by the Elag program. Here is the elag.rul file used for French:

```

    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>¶

```

13.10 Tagger files

This section presents files produced and used by TrainingTagger and Tagger programs.

13.10.1 The corpus.txt file

This file is used by the TrainingTagger program in order to compute statistics for the Tagger program. It contains sentences where each word is represented in a separate line. Each line representing a word is composed of a word, simple or compound, followed by a slash and the tag of the word. This tag is composed of a grammatical code, sometimes followed by a '+' and syntactic or semantic codes. Inflectional codes are specified after a ':'. If the word is a compound, simple words contained in it must be separated by a '_'. Here is an example of a corpus.txt file :

```

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¶
¶

```

NOTE: Sentences must be delimited by empty lines.

13.10.2 The tagger data file

The TrainingTagger program generates two data files (by default) used by the Tagger program in order to compute a second-order hidden Markov model. These files contain unigram, bigram and trigram tuples extracted from the tagged corpus.txt file. Tuples are composed of either a sequence of 2 or 3 tags (to compute transition probability) or a word preceded by 0 or 1 tag (to compute emit probability). Units in a tuple must be separated by a tabulation. These tuples are followed by the sequence of delimiters "," and then an integer representing the number of occurrences of this tuple in the corpus file.

Filenames are suffixed by "cat" or "morph". In the first one, tuples are composed of tags formed of grammatical, syntactic and semantic codes. In the second one, tuples consist in tags formed of grammatical, syntactic and semantic codes and sometimes followed by a ' : ' and inflectional codes. Here is an example of a data file with "cat" tags :

```

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¶
¶
```

Here is an example of a data file with "morph" tags :

```
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¶
¶
```

A special line is added to data files in order to identify whether the file contains "cat" or "morph" tags. This line contains CODE FEATURES followed by either the integer 0 for "cat" tags or 1 for "morph" tags.

NOTE: At the final stage, TrainingTagger compresses these two data files into the ".bin" format.

13.11 Configuration files

13.11.1 The Config file

Whenever the user modifies his preferences for a given languages, these modifications are saved in a text file named 'Config' which can be found in the directory of the current language. The file has the following syntax (the order of lines can vary):

```
#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¶

```

The first two lines are comment lines. The following three lines indicate the name, the style and the size of the font used to display texts, dictionaries, lexical units, sentences in text automata, etc.

The `CONCORDANCE FONT NAME` and `CONCORDANCE FONT HTML SIZE` parameters define the name, the size and the font to be used when displaying concordances in HTML. The size of the font has a value between 1 and 7.

The `INPUT FONT . . .` and `OUTPUT FONT . . .` parameters define the name, the style and the size of the fonts used for displaying the paths and the transducer outputs of the graphs.

The following 10 parameters correspond to the parameters given in the headings of the graphs. Table 13.5 describes the correspondances.

The `PACKAGE NODES` parameter defines the color to be used for displaying calls to sub-graphs located in the repository.

The `CONTEXT NODES` parameter defines the color to be used for displaying boxes that correspond to context bounds.

The `CONTEXT NODES` indicates if the current language must be tokenized character by character or not.

The `ANTIALIASING` parameter indicates whether graphs as well as sentence automata are displayed by default with the antialiasing effect.

Parameters in the Config file	Parameters in the .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 13.5: Meaning of the parameters

The `HTML VIEWER` parameter indicates the name of the navigator to be used for displaying concordances. If no navigator name is defined, concordances are displayed in a Unitex window.

The `MAX TEXT FILE SIZE` parameter is deprecated.

The `ICON BAR POSITION` parameter indicates the default position of icon bars in graph frames.

The `PACKAGE PATH` parameter specifies the location of the repository.

The `MORPHOLOGICAL DICTIONARY` parameter specifies the list of morphological dictionaries to use, separated with semi-colons.

The `MORPHOLOGICAL NODES COLOR` parameter specifies the color to use to render the `$<` and `$>` tags.

The `MORPHOLOGICAL USE OF SPACE` parameter indicates if the `Locate` program is allowed to start matching on spaces. Default is false.

13.11.2 The `system_dic.def` file

The `system_dic.def` file is a text file that describes the list of system dictionaries that are applied by default. This file can be found in the directory of the current language. Each line corresponds to a name of a `.bin` file. The system dictionaries are in the system directory, and in that directory in the `(current language)/Dela` sub-directory. Here is an example of this file:

```
delacf.bin¶
delaf.bin¶
```

13.11.3 The `user_dic.def` file

The `user_dic.def` file is a text file that describes the list of dictionaries the user has defined to be applied by default. This file is in the directory of the current language and has the same format as the `system_dic.def` file. The dictionaries need to be in the `(current language)/DeL` sub-directory of the personal directory of the user.

13.11.4 The `user.cfg` and `.unitex.cfg` files

Under Linux, Unitex expects the personal directory of the user to be called `unitex` and expects it to be in his root directory (`$HOME`). If you want to change this default location, a `.unitex.cfg` file is created in your home directory, and it contains the path to your private Unitex directory. This file is a UTF8 one.

Under Windows, it is not always possible to associate a directory to a user per default. To compensate for that, Unitex creates a `.cfg` file for each user that contains the path to his personal directory. This file is saved under the name `(user login).cfg` in the `Unitex/Users` system sub-directory.

WARNING: THIS FILE IS NOT IN UNICODE

WARNING (2): THE PATH OF THE PERSONAL DIRECTORY IS NOT FOLLOWED BY A NEWLINE.

13.12 Cassys files

13.12.1 Cassys configuration files `csc`

To memorize the list of transducer of a CasSys cascade, we use a text file (`csc` file) in which each line contains the path to a transducer followed by the output policy (`merge/replace`) to be applied to this transducer. The generic format of a line of `csc` file is : `Name_and_path_of_transducer`
Merge Here is an example of cascade file `csc`:

```
"C:\apps\my_unitex\French\Graphs\grf1.fst2" Merge  
"C:\apps\my_unitex\French\Graphs\grf2.fst2" Replace
```

13.13 Various other files

For each text, Unitex creates multiple files that contain information that are designed to be displayed in the graphical interface. This section describes these files and some others.

13.13.1 The `dlf.n`, `dlc.n`, `err.n` et `tags_err.n` files

These files are text files that are stored in the text directory. They contain the number of lines of the `dlf`, `dlc`, `err` and `tags_err` files respectively. These numbers are followed by a newline.

13.13.2 The `stat_dic.n` file

This file is a text file in the directory of the text. It has three lines that contain the number of lines of the `dlf`, `dlc` and `err` files.

13.13.3 The `stats.n` file

This file is in the text directory and contains a line with the following form:

```
3949 sentence delimiters, 169394 (9428 diff) tokens, 73788 (9399)
simple forms, 438 (10) digits¶
```

The numbers indicated are interpreted in the following way:

- `sentence delimiters`: number of sentence separators (`{S}`);
- `tokens`: total number of lexical units in the text. The number preceding `diff` indicates the number of different units;
- `simple forms`: the total number of lexical units in the text that are composed of letters. The number in parentheses represents the number of different lexical units that are composed of letters;
- `digits`: the total number of digits used in the text. The number in parentheses indicates the number of different digits used (10 at most).

13.13.4 The `concord.n` file

The `concord.n` file is a text file in the directory of the text. It contains information on the latest search of the text and looks like the following:

```
6 matches¶
6 recognized units¶
(0.004% of the text is covered)¶
```

The first line gives the number of found occurrences, and the second the name of units covered by these occurrences. The third line indicates the ratio between the covered units and the total number of units in the text.

13.13.5 The `concord_tfst.n` file

The `concord_tfst.n` file is a text file in the directory of the text. It contains information on the latest search on the text automaton and looks like the following:

```
23 matches (45 outputs)¶
```

13.13.6 Normalization rule file

This file is used by the `Normalization` and `XMLizer` programs. It represents replacement rules. Each line stands for a rule, according to the following format (\mapsto stands for the tabulation character):

```
input sequence  $\mapsto$  output sequence
```

If you want to use the tabulation or the new line, you must protect them with a backslash like this:

```
123\  
 $\mapsto$  ONE_TWO_THREE_NEW_LINE
```

13.13.7 Forbidden word file

The `PolyLex` programs requires a forbidden word file for Dutch and Norwegian. This raw text file is supposed to be named `ForbiddenWords.txt`. It must be in the user's `Dela` directory corresponding to the language to work on. Each line is supposed to contain one forbidden word.

13.13.8 Log file

The `UnitexToolLogger` programs, when a `unitex_logging_parameters.txt` file is found with a path (to store logfile) creates `.ulp` file with a log of the running Unitex tool selected. It creates a `unitex_logging_parameters_count.txt` file which contain only the number of latest log file created.

Log file (with `.ulp` extension) are uncompressed zipfile, compatible with `unzip` and all standard `unzip` tools. It contain these files:

- `test_info/command_line.txt`: a list of parameter of the command line used to run the tool. There is one parameter on each line. The first line contain the return value, the second line the number of parameters;
- `test_info/command_line_synth.txt`: a simple line with a summary of the command line used to run the tool;
- `test_info/list_file_in.txt`: a list of file read by the tool. The first column is file size, second column is `crc32`, third is filename;
- `test_info/list_file_out.txt`: a list of file created by the tool. The first column is file size, second column is `crc32`, third is filename;
- `test_info/std_out.txt`: content of standard console output;
- `test_info/std_err.txt`: content of error console output;
- `src/xxx`: a copy of file read by the tool (needed to run the log again);

- `dest/xxx`: a copy of file created by the tool.

If the second line of `unitex_logging_parameters.txt` contains 0, these file are not recorded; if this line contains 1, they are recorded;

13.13.9 Arabic typographic rules: `arabic_typo_rules.txt`

For Arabic, dictionary lookups can be parameterized with a file that describes whether some typographic variations are allowed or not. This file is made of lines like the following:

```
fatha omission=YES
```

where `fatha omission` is the name of the rule. For a complete description of all the available rules, you have to consult the `Arabic.h` file in the program sources.

Appendix A - GNU Lesser General Public License

This license can also be found in [\[35\]](#).

GNU LESSER GENERAL PUBLIC LICENSE Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages—typically libraries—of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive

or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses

the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) The modified work must itself be a software library.
- b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore

falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do

not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND,

EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.> Copyright (C)
<year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990 Ty Coon, President of Vice
That's all there is to it!

Appendix B - TRE's 2-clause BSD License

This is the license, copyright notice, and disclaimer for TRE, a regex matching package (library and tools) with support for approximate matching.

Copyright (c) 2001-2009 Ville Laurikari <vl@iki.fi> All rights reserved.

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.

Appendix C - Lesser General Public License For Linguistic Resources

This license was designed by the University of Marne-la-Vallée, and it has received the approval of the Free Software Foundation ([1]).

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 utility 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

Bibliography

- [1] Free Software Foundation. <http://www.fsf.org>. 13.13.9
- [2] Anna ANASTASSIADIS-SYMEONIDIS, Tita KYRIACOPOULOU, Elsa SKLAVOUNOU, Iason 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. 3.7
- [3] Jean-Claude ANSCOMBRE. Pourquoi un moulin à vent n'est pas un ventilateur. *Langue Française*, 86, 1990. 10.1
- [4] Laurie BAUER. *English Word-Formation*. Cambridge University Press, 1983. 10.1
- [5] Emile BENVENISTE. *Fondements syntaxiques de la composition nominale. Formes nouvelles de la composition nominale*, pages 145–176. Gallimard, Paris, 1974. 10.1
- [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. *Linguisticæ Investigationes*, 21(1), 1997. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.7
- [8] Xavier BLANCO. Les dictionnaires électroniques de l'espagnol (DELASs et DELACs). *Linguisticæ Investigationes*, 23(2), 2000. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.7
- [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. 8.1
- [10] Jean-Paul BOONS, Alain GUILLET, and Christian LECLÈRE. *La structure des phrases simples en français : constructions intransitives*. Droz, Genève, 1976. 8.1
- [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. 10.1

- [14] Folker CAROLI. Les verbes transitifs à complément de lieu en allemand. *Linguisticae Investigationes*, 8(2):225–267, 1984. Amsterdam-Philadelphia : John Benjamins Publishing Company. 8.1
- [15] A. CHROBOT, B. COURTOIS, M. HAMMANI-MC CARTHY, M. GROSS, and K. ZELLAGUI. Dictionnaire électronique DELAC anglais : noms composés. Technical Report 59, LADL, Université Paris 7, 1999. 3.7
- [16] Unicode Consortium. <http://www.unicode.org>. 2.2
- [17] Matthieu CONSTANT and Anastasia YANNAKOPOULOU. Le dictionnaire électronique du grec moderne: Conception et développement d'outils pour son enrichissement et sa validation. In *Studies in Greek Linguistics, Proceedings of the 23rd annual meeting of the Department of Linguistics*. Faculty of Philosophy, Aristotle University of Thessaloniki, 2002. 3.7
- [18] Danielle CORBIN. Hypothèses sur les frontières de la composition nominale. *Cahiers de grammaire*, 17:26–55, 1992. Université de Toulouse Le Mirail. 10.1
- [19] Blandine COURTOIS. Formes ambiguës de la langue française. *Linguisticae Investigationes*, 20(1):167–202, 1996. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.7
- [20] Blandine Courtois and Max Silberztein, editors. *Les dictionnaires électroniques du français*. Larousse, Langue française, vol. 87, 1990. 3.7, 10.2.1, 10.2.2
- [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. 10.1
- [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. 9
- [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. *Linguisticae Investigationes*, 19(1):57–82, 1995. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.7
- [26] Anibale ELIA. *Le verbe italien. Les complétives dans les phrases à un complément*. Schena/Nizet, Fasano/Paris, 1984. 8.1
- [27] Anibale ELIA. *Lessico-grammatica dei verbi italiani a completiva. Tavole e indice generale*. Liguori, Napoli, 1984. 8.1

- [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. 3.7
- [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. 3.7
- [30] Nathalie FRIBURGER. *Reconnaissance automatique des noms propres - Application à la classification automatique de textes journalistiques*. PhD thesis, Université François-Rabelais, Tours, 2002. 11
- [31] A Simple English Axis Generator. <http://nlp.cs.nyu.edu/GMA/docs/HOWTO-axis>. 12.8
- [32] Jacqueline GIRY-SCHNEIDER. Syntax and lexicon: Blessure (wound), noeud (knot), caresse (caress)... *SMIL, Journal of Linguistic Calculus*, 3-4:55–72, 1978. 8.1
- [33] Jacqueline GIRY-SCHNEIDER. *Les nominalisations en français. L'opérateur faire dans le lexique*. Droz, Genève-Paris, 1978. 8.1
- [34] Jacqueline GIRY-SCHNEIDER. *Les prédicats nominaux en français. Les phrases simples à verbe support*. Droz, Genève-Paris, 1987. 8.1
- [35] GNU. Lesser General Public License. <http://www.gnu.org/licenses/lgpl.html>. 1.1, 13.13.9
- [36] Gaston GROSS. Définition des noms composés dans un lexique-grammaire. *Langue Française*, 87, 1990. 10.1
- [37] Gaston GROSS. *Les expressions figées en français. Noms composés et autres locutions*. Ophrys, Paris, 1996. 3.7, 10.1
- [38] Maurice GROSS. *Méthodes en syntaxe*. Hermann, Paris, 1975. 8.1
- [39] 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. 8.1
- [40] Maurice GROSS. Taxonomy in syntax. *SMIL, Journal of Linguistic Calculus*, 3-4:73–96, 1978. 8.1
- [41] 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. 8.1
- [42] Maurice GROSS. On structuring the lexicon. *Quaderni di Semantica*, 4(1):107–120, 1983. 8.1

- [43] 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. 8.1
- [44] 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. 8.1
- [45] Maurice GROSS. *Grammaire transformationnelle du français. 3 - Syntaxe de l'adverbe*. AS-STRIL, Paris, 1986. 3.7, 8.1
- [46] Maurice GROSS. Lexicon-grammar. the representation of compound words. In *COLING-1986 Proceedings*, pages 1–6. Bonn, 1986. 8.1
- [47] 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. 8.1
- [48] Maurice GROSS. Linguistic representations and text analysis. In *Linguistic Unity and Linguistic Diversity in Europe*, pages 31–61. London: Academia Europaea, 1991. 8.1
- [49] Maurice GROSS. Constructing lexicon-grammars. In Atkins and Zampolli, editors, *Computational Approaches to the Lexicon*, pages 213–263. Oxford Univ. Press, 1994. 8.1
- [50] 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. 8.1
- [51] Alain GUILLET and Christian LECLÈRE. *La structure des phrases simples en français : les constructions transitives locatives*. Droz, Genève, 1992. 8.1
- [52] Benoît HABERT and Christian JACQUEMIN. Noms composés, termes, dénominations complexes: problématiques linguistiques et traitements automatiques. *Traitement Automatique des Langues*, 2:5–41, 1993. 10.1
- [53] IGM. Lesser General Public License for Linguistic Resources. <http://igm.univ-mlv/~unitex/lgp11r.html>. 1.1
- [54] Text Encoding Initiative. <http://www.tei-c.org>. 9.1
- [55] Christian JACQUEMIN. *Spotting and Discovering Terms through Natural Language Processing*. MIT Press, 2001. 10.2.3
- [56] 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. 3.7
- [57] Cvetana KRSTEV, Duško VITAS, and Agata SAVARY. Prerequisites for a Comprehensive Dictionary of Serbian Compounds. *LNCS*, 4139:552–563, 2006. 10.2

- [58] Tita KYRIACOPOULOU. *Les dictionnaires électroniques: la flexion verbale en grec moderne*, 1990. Thèse de doctorat. Université Paris 8. 3.7
- [59] 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. 3.7
- [60] Tita KYRIACOPOULOU, Safia MRABTI, and Anastasia YANNAKOPOULOU. Le dictionnaire électronique des noms composés en grec moderne. *Linguisticae Investigationes*, 25(1):7–28, 2002. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.7
- [61] Jacques LABELLE. Le traitement automatique des variantes linguistiques en français: l'exemple des concrets. *Linguisticae Investigationes*, 19(1):137–152, 1995. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.7
- [62] Eric LAPORTE and Anne MONCEAUX. Elimination of lexical ambiguities by grammars : The ELAG system. *Linguisticae Investigationes*, 22:341–367, 1998. Amsterdam-Philadelphia : John Benjamins Publishing Company. 7.3
- [63] Ville LAURIKARI. TRE home page. <http://laurikari.net/tre/>. 1.1, 4.7
- [64] 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. 8.1
- [65] Judith N. LEVI. *The Syntax and Semantics of Complex Nominals*. Academic Press, New York-London, 1978. 10.1
- [66] XAlign(Alignement multilingue) LORIA (2006). <http://led.loria.fr/outils/ALIGN/align.html>. 9
- [67] Annie MEUNIER. *Nominalisation d'adjectifs par verbes supports*, 1981. Thèse de doctorat. Université Paris 7. 8.1
- [68] Sun Microsystems. Java. <http://java.sun.com>. 1.2
- [69] Christian MOLINIER and Françoise LEVRIER. *Grammaire des adverbes: description des formes en -ment*. Droz, Genève, 2000. 8.1
- [70] Anne MONCEAUX. Le dictionnaire des mots simples anglais : mots nouveaux et variantes orthographiques. Technical Report 15, IGM, Université de Marne-la-Vallée, 1995. 3.7
- [71] 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. 3.7
- [72] OpenOffice.org. <http://www.openoffice.org>. 2.2, 8.2.2

- [73] Dong-Ho PAK. *Lexique-grammaire comparé français-coréen. Syntaxe des constructions complétives*. PhD thesis, UQAM, Montréal, 1996. [8.1](#)
- [74] Soun-Nam PARK. *La construction des verbes neutres en coréen*, 1996. Thèse de doctorat. Université Paris 7. [8.1](#)
- [75] 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. [9](#), [9.2](#)
- [76] 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, EAACL 2003*, 2003. [10.1.1](#), [10.2.2](#)
- [77] Roger-Bruno RABENILAINA. *Le verbe malgache*. AUPELF-UREF et Université Paris 13, Paris, 1991. [8.1](#)
- [78] Elisabete RANCHHOD. Frozen adverbs. comparative forms como c in portuguese. *Linguisticae Investigationes*, 15(1):141–170, 1991. Amsterdam-Philadelphia : John Benjamins Publishing Company. [3.7](#), [8.1](#)
- [79] Elisabete RANCHHOD. Ressources linguistiques du portugais implémentées sous intex. In C. Fairon, editor, *Analyse Lexicale et Syntaxique: Le système INTEX*, *Linguisticae Investigationes*, pages 263–277. Amsterdam-Philadelphia : John Benjamins Publishing Company, 1998. [3.7](#)
- [80] Elisabete RANCHHOD. Problèmes de traduction automatique des constructions à verbes supports. *Linguisticae Investigationes*, 23(2):253–267, 2001. Amsterdam-Philadelphia : John Benjamins Publishing Company. [8.1](#)
- [81] Elisabete RANCHHOD and Michele DE GIOIA. Comparative romance syntax. frozen adverbs in italian and in portuguese. *Linguisticae Investigationes*, 20(1):33–85, 1996. Amsterdam-Philadelphia : John Benjamins Publishing Company. [8.1](#)
- [82] 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. [3.7](#)
- [83] Morris SALKOFF. Verbs of mental states. In *Lexique, syntaxe et lexique-grammaire. Papers in honour of Maurice Gross*, volume 24 of *Linguisticae Investigationes Supplementa*, pages 561–571. Amsterdam/Philadelphia: Benjamins, 2004. [8.1](#)
- [84] Agata SAVARY. *Recensement et description des mots composés - méthodes et applications*, 2000. Thèse de doctorat. Université de Marne-la-Vallée. [3.7](#), [10.1.1](#), [10.1.2](#)
- [85] Agata SAVARY. A formalism for the computational morphology of multi-word units. *Archives of Control Sciences*, 15(3):437–449, 2005. [10](#), [10.1.2](#), [10.2](#)

- [86] Max SILBERZTEIN. Les groupes nominaux productifs et les noms composés lexicalisés. *Linguisticae Investigationes*, 27(2):405–426, 1999. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.7, 10.1
- [87] Carlos SUBIRATS-RÜGGERBERG. *Sentential complementation in Spanish. A lexicogrammatical study of three classes of verbs*. John Benjamins, Amsterdam/Philadelphia, 1987. 8.1
- [88] Thomas TREIG. Complétives en allemand. classification. Technical Report 7, LADL, 1977. 8.1
- [89] 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. 8.1
- [90] 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. 3.7
- [91] 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. 9

Index

+, 48, 77, 90
_, 166
cat, 165
complete, 166
discr, 165
flex, 165
t, 34
{STOP}, 41
!, 74
#, 37, 72, 74, 112, 125
\$*, 120
\$, 98
\$<, 124
\$>, 124
\$[, 118
\$], 118
*, 77
,, 48, 50
-, 63
., 48, 57, 76
/, 48, 97
1, 52
2, 52
3, 52
:, 48, 93
<CDIC>, 72, 125
<DIC>, 72, 75, 125
<E>, 37, 72, 74, 77, 90, 110, 112
<I=?>, 57
<L>, 110
<MAJ>, 37, 72, 75, 125
<MIN>, 37, 72, 75, 125
<MOT>, 37, 72, 125
<NB>, 37, 72, 74, 125
<PNC>, 37
<PRE>, 37, 72, 75, 125
<R=?>, 57
<SDIC>, 72, 125
<TDIC>, 72
<TOKEN>, 125, 177
<X=n>, 57
<^>, 37, 110
=, 49
@%, 185
@, 185
A, 51
ADV, 51
Abst, 51
Anl, 51
AnlColl, 51
BuildKrMwuDic, 237
C, 52, 57, 109
CONJC, 51
CONJS, 51
Cassys, 238
CheckDic, 53, 239, 288
Compress, 49, 61, 240, 286
Conc, 51
ConcColl, 51
ConcorDiff, 144, 242
Concord, 240
Convert, 243
D, 57, 109
DET, 51
Dico, 43, 64, 245
Elag, 246, 291
ElagComp, 246
Equivalences.txt, 203
Evamb, 246
Extract, 247
F, 52
Flatten, 113, 247
Fst2Check, 247
Fst2Grf, 173

- Fst2List, 248
- Fst2Txt, 39, 249
- G, 52
- Grf2Fst2, 113, 250
- Hum, 51
- HumColl, 51
- I, 52
- INTJ, 51
- ImplodeTfst, 251
- J, 52, 57
- K, 52
- L, 57, 109
- Locate, 64, 196, 251
- LocateTfst, 253
- Morphology.txt, 202, 203
- MultiFlex, 254
- N, 51
- Normalize, 236, 255
- P, 52, 57, 109
- PREP, 51
- PRO, 51
- PolyLex, 44, 256
- R, 57, 109
- RebuildTfst, 256
- Reconstrucao, 153, 257
- Reg2Grf, 257
- S, 52
- SortTxt, 54, 257, 271
- Stats, 258
- T, 52
- TEI2Txt, 260
- Table2Grf, 259
- Tagger, 259
- TagsetNormTfst, 260
- Tfst2Grf, 260
- Tfst2Unambig, 175, 261
- Tokenize, 41, 261
- TrainingTagger, 262
- Txt2Tfst, 263
- U, 57, 109
- Uncompress, 263
- UnitexTool, 264
- UnitexToolLogger, 265
- Untokenize, 264
- V, 51
- W, 52, 57, 109
- XMLizer, 267
- Y, 52
- \, 48, 71
- \,, 48
- \., 48
- \=, 49
- _, 98
- en, 51
- f, 52
- i, 51
- m, 52
- n, 52
- ne, 51
- norm.rul, 168
- p, 52
- s, 52
- se, 51
- t, 51
- tags.ind, 66
- tokens.txt, 177
- z1, 51
- z2, 51
- z3, 51
- {STOP}, 72, 76
- {S}, 37, 76, 255, 261, 276, 277, 297
- ~, 73
- +, 63
- Acyclic automaton, 147
- Adding languages, 27
- Advanced search options, 139
- Algebraic languages, 90
- All matches, 79, 137
- Alphabet, 38, 242, 249, 251, 253, 261, 263, 270
 - Korean, 180
 - sort, 54
 - sorted, 271
- Ambiguity rate, 163
- Ambiguity removal, 157
- Ambiguous outputs, 139
- Analysis of compound words in German, 44
- Analysis of compound words in Norwegian, 44
- Analysis of compound words in Russian, 44

- Analysis of free compounds in Dutch, 256
- Analysis of free compounds in German, 256
- Analysis of free compounds in Norwegian, 256
- Analysis of free compounds in Russian, 256
- Antialiasing, 102, 294
- Approximation of a grammar through a final state transducer, 247
- Approximation of a grammar with a finite state transducer, 113
- Arabic typographic rules, 299
- Automata
 - finite state, 90
 - text, 260
- Automatic inflection, 56, 109, 254
- Automaton
 - acyclic, 147
 - minimal, 62
 - of the text, 73, 147, 263
 - text, 111
- Axiom, 89
- Box alignment, 104
- Boxes
 - alignment, 104
 - connecting, 93
 - creating, 90
 - deleting, 97
 - selection, 96
 - sorting lines, 101
- BSD, 311
- cascade of transducer, 227
- Cascade of transducers, 238
- Case
 - see Respect
 - of lowercase/uppercase, 112
- Case respect, 111
- Case sensitivity, 72, 79
- CasSys, 227
- Checking dictionary format, 53
- Chinese characters, 180
- Clitics
 - normalization, 151, 257
- Cognates, 193
- Collections of graphs, 129
- Colors
 - configuration, 105
- Comment
 - in a dictionary, 48
- Comments
 - in a graph, 90
- Comparing concordances, 144
- Comparing variables, 137
- Compilation of a graph, 113
- Compilation of graphs, 250
- Compiling
 - ELAG grammars, 158
- Compound words, 199
- Compressing dictionaries, 240
- Compression of dictionaries, 257
- Concatenation of regular expressions, 71, 76
- Concordance, 81, 142, 240
 - comparison, 144
- Concordance frame, 82
- Conservation of better paths, 263
- Console, 236
- Consonant skeleton, 61
- Constraints on grammars, 114
- Context, 64
- Context-free languages, 90
- Contexts, 117
 - concordance, 81, 142, 240
 - copy of a list, 99
- Copy, 97, 99, 101
- Copying lists, 99
- Corpus, *see* Text
- Creating a Box, 90
- Creating log files, 236
- Cut, 101
- Degree of ambiguity, 149
- DELA, 36, 47
- DELAC, 47
- DELACF, 47
- DELAF, 47–50, 64, 286
- DELAS, 47, 50
- Derivation, 89
- Dictionaries
 - application of, 245

- applying, 42, 63
- automatic inflection, 56, 254
- checking, 53
- codes used within, 51
- comments in, 48
- compressing, 240
- compression, 61, 257
- contents, 51
- default selection, 43
- DELAC, 47
- DELACE, 47
- DELAF, 47–50, 64, 240, 286
- DELAS, 47, 50
- filters, 63
- format, 47
- granularity, 149
- of the text, 73, 147
- priority, 63
- refer to, 73
- reference to information in the, 112
- sorting, 54
- text, 43
- verification, 239
- Dictionary entry variables, 126
- Dictionary graphs, 64
- Dictionary information file, 288
- Directory
 - personal working, 27
 - text, 37, 235
- ELAG, 112, 157
- ELAG tag sets, 163
- Epsilon, *see* <E>
- Equivalent characters, 54
- Error detection in graphs, 116, 248, 250
- Errors in graphs, 116, 248, 250
- Evaluation of the ambiguity rate, 163
- Exclusion of grammatical and semantic codes, 73
- Exploring the paths of a grammar, 127
- External programs
 - BuildKrMwuDic, 237
 - Cassys, 238
 - CheckDic, 53, 239, 288
 - Compress, 49, 61, 240, 286
 - ConcorDiff, 144, 242
 - Concord, 240
 - Convert, 243
 - Dico, 43, 64, 245
 - Elag, 160, 162, 163, 246, 291
 - ElagComp, 160, 163, 169, 246
 - Evamb, 246
 - Extract, 247
 - Flatten, 113, 247
 - Fst2Check, 247
 - Fst2Grf, 173
 - Fst2List, 248
 - Fst2Txt, 39, 249
 - Grf2Fst2, 113, 250
 - ImplodeTfst, 251
 - Locate, 64, 251
 - LocateTfst, 253
 - MultiFlex, 254
 - Normalize, 236, 255
 - PolyLex, 44, 256
 - RebuildTfst, 256
 - Reconstrucaao, 153, 257
 - Reg2Grf, 257
 - SortTxt, 54, 257, 271
 - Stats, 258
 - TEI2Txt, 260
 - Table2Grf, 259
 - Tagger, 259
 - TagsetNormTfst, 260
 - Tfst2Grf, 260
 - Tfst2Unambig, 175, 261
 - Tokenize, 41, 261
 - TrainingTagger, 262
 - Txt2Tfst, 263
 - Uncompress, 263
 - UnitexTool, 264
 - UnitexToolLogger, 265
 - Untokenize, 264
 - XMLizer, 267
- Extracting occurrences, 144
- Factorized lexical entries, 162
- File
 - tagset.def, 163, 166, 168, 169
 - conc.fst2, 160

- .bin, 61, 240, 245, 286, 295
- .cfg, 296
- .dic, 53, 61, 240
- .elg, 290
- .fst2, 80, 113, 173, 250, 275
- .grf, 80, 117, 173, 250, 257, 272
- .html, 242
- .inf, 61, 240, 286
- .lst, 162, 290
- .rul, 158, 160, 162, 246, 291
- .snt, 37, 255, 261, 263, 269, 277
- .tfst, 246
- .txt, 143, 242, 269, 276
- Alphabet.txt, 270
- Alphabet_sort.txt, 54
- CHECK_DIC.TXT, 53, 239, 288
- Config, 293
- Equivalences.txt, 203
- ForbiddenWords.txt, 298
- Morphology.txt, 202, 203
- Sentence.fst2, 38
- Unitex.jar, 18, 28
- alphabet, 64
- arabic_typo_rules.txt, 299
- concord.html, 283
- concord.ind, 253, 254, 282
- concord.n, 253, 297
- concord.txt, 283
- concord_tfst.n, 254, 297
- corpus.txt, 291
- cursor_sentence.grf, 261, 281
- cursor_sentence.tok, 261, 281
- cursor_sentence.txt, 261, 281
- diff.html, 284
- dlc, 43, 56, 245, 285, 296
- dlc.n, 296
- dlf, 43, 56, 245, 285, 296
- dlf.n, 296
- enter.pos, 262, 277
- err, 43, 56, 245, 285, 296
- err.n, 296
- norm.rul, 168
- regexp.grf, 257
- stat_dic.n, 245, 297
- stats.n, 41, 262, 297
- system_dic.def, 295
- tags.ind, 285
- tags_err, 285, 296
- tags_err.n, 296
- tagset.def, 290
- text.cod, 41, 261, 277
- text.tfst, 263, 278
- text.tind, 263, 280
- tfst_tags_by_alph.txt, 282
- tfst_tags_by_freq.txt, 282
- tok_by_alph.txt, 41, 262, 277
- tok_by_freq.txt, 41, 262, 277
- tokens.txt, 41, 261, 277
- train_dict, 292
- unitex_2.1.zip, 18
- user_dic.def, 296
- alphabet, 31, 38, 41, 53, 242, 249, 251, 253, 261, 263
- formats, 269
- HTML, 81, 142
- text, 34, 269
- transcoding, 32
- Forbidden word file, 298
- Form
 - canonical, 48
 - inflected, 47
- Generation of Korean MWU dictionary, 237
- German
 - compound words, 44
- GlossaNet, 241, 283
- Grammars
 - ambiguity removal, 157
 - collection, 162
 - constraints, 114
 - context-free, 89
 - ELAG, 112
 - extended algebraic, 90
 - for phrase boundary recognitions, 110
 - formalism, 89
 - inflectional, 56
 - local, 112
 - normalisation
 - of non-ambiguous forms, 110
 - of the text automaton, 111

- normalization
 - of non-ambiguous forms, 39
 - splitting into sentences, 37
- Granularity of dictionaries, 149
- Graph
 - antialiasing, 102
 - approximation through a final state transducer, 247
 - approximation with a finite state transducer, 113
 - box alignment, 104
 - calling a sub-graph, 93
 - comments in, 90
 - compilation, 113, 250
 - connecting boxes, 93
 - creating a box, 90
 - deleting boxes, 97
 - detection of errors, 116
 - display, 101
 - display options, fonts and colors, 105
 - error detection, 248, 250
 - format, 272
 - including into a document, 107
 - main, 259, 262
 - parameterized, 113, 184
 - printing, 108
 - saving, 93
 - syntactic, 112
 - types of, 109
 - variables in a, 98
 - zoom, 102
- Graph repository, 95
- Graphical units, 199
- Grid, 104
- Hangul, 58, 255, 278
- Including a graph into a document, 107
- Inflectional codes, 166
- Inflectional constraints, 74
- Information
 - grammatical, 48
 - inflectional, 48
 - semantic, 48
- Installation
 - on Linux, 18
 - on MacOS X, 19
 - on Windows, 18
- Integrated text editor, 34
- Jamo, 58, 278
- Java Runtime Environment, 17
- Java virtual machine, 17
- JRE, 17
- Keeping the best paths, 153
- Kleene star, 71, 77
- Korean MWU dictionary, 237
- LADL, 11, 47, 183
- Language selection, 31
- Lemma, 48
- Lexical
 - entries, 47
 - labels, 73, 149, 255, 261, 276, 277
 - mask, 72
 - resources, *see* Dictionaries
 - symbols, 170
 - units, 263
- Lexical units, 261
- Lexicon-grammar, 183
 - tables, 183, 259, 262
- LGPL, 17, 301
- LGPLLR, 17, 313
- License
 - BSD, 311
 - LGPL, 17, 301
 - LGPLLR, 313
- Log file, 298
- Log Unitex programs, 265
- Longest matches, 79, 137
- Lowercase
 - see* Respect
 - of lowercase/uppercase, 112
- Matrices, 183
- MERGE, 39, 64, 130, 137, 282
- Meta-characters, 100
- Meta-symbols, 37, 72
- Modification of the text, 143, 240
- Morphological dictionaries, 125

- Morphological dictionary graphs, 66
- Morphological filters, 64, 77
- Morphological mode, 64, 124
- Moving word groups, 132
- Multi-word units, 199
- Multiple selection, 96
 - copy-paste, 97
- MWU, 199

- Negation, 74
- Negation of a lexical mask, 74
- Non-terminal symbols, 89
- Normalization
 - clitics in Portuguese, 257
 - of ambiguous forms, 263
 - of ambiguous forms, 111, 150
 - of clitics in Portuguese, 151
 - of non-ambiguous forms, 39
 - of separators, 36, 255
 - of the text automaton, 111, 150, 263
- Normalization rule file, 298
- Norwegian
 - compound words, 44

- Occurrences
 - extraction, 144
 - number of, 80, 137
- Operator
 - ., 57
 - <I=?>, 57
 - <R=?>, 57
 - <X=n>, 57
 - C, 57, 109
 - D, 57, 109
 - J, 57
 - L, 57, 109
 - P, 57, 109
 - R, 57, 109
 - U, 57, 109
 - w, 57, 109
 - concatenation, 76
 - disjunction, 77
 - Kleene star, 77
- Optimizing ELAG Grammars, 169
- Options
 - configuration, 105
- Output associated to a subgraph call, 115
- Output variables, 135
- Overlapping occurrences, 131

- Parameterized graphs, 184
- Parenthesis, 76
- Paste, 97, 99, 101
- Pattern search, 251, 253
- Pixellisation, 102
- PNG graph export, 107
- Portuguese
 - normalization of clitics, 151, 257
- POSIX, 77
- Preferences, 106
- Printing
 - a graph, 108
 - a sentence automaton, 175
- Priority
 - of dictionaries, 63
 - of the leftmost match, 131
 - of the longest match, 132

- Reconstruction of the text automaton, 256
- Recursive Transition Networks, 90
- Reentrant alignment, 193
- Reference to information in the dictionaries, 73, 112
- Regular expressions, 71, 77, 90, 257
- REPLACE, 130, 137, 282
- Resolving ambiguity, 160
- Respect
 - of lowercase/uppercase, 110, 112
 - of spaces, 112
- RTN, 90
- Rules
 - for transducer application, 130
 - rewriting, 89
 - upper case and lower case letters, 64
 - white space, 64
- Russian
 - compound words, 44

- Scripting Unitex programs, 264
- Search for patterns, 79, 137
- Selecting a language, 31

- Semitic languages, 61
- Sentence delimiter, 255
- Sentence separator, 37, 76, 261, 276, 277, 297
- Separators, 36
- Shortest matches, 79, 137
- Sorting, 257
 - a dictionary, 54
 - concordances, 241
 - lines of a box, 101
 - of concordances, 81, 142
- SoyLatte, 20
- Space
 - obligatory, 72
 - prohibited, 72
- Splitting into sentences, 37
- State
 - final, 90
 - initial, 90
- Statistics, 258
- SVG graph export, 108
- Symbols
 - non-terminal, 89
 - special, 100
 - terminal, 89
- Synchronization point, 157
- Syntactical properties, 183
- Syntax diagrams, 90
- Testing variables, 136
- Text
 - automata, 260
 - automaton, 256
 - automaton of the, 73, 263
 - directory, 37
 - directory of, 236
 - formats, 31
 - modification, 143, 240
 - normalisation of the automaton, 111
 - normalization, 36, 255
 - normalization of the automaton, 150
 - preprocessing, 34, 110
 - splitting into sentences, 37
 - splitting into tokens, 41
 - tokenization, 41, 261
- Text alignment, 191
- Text automaton
 - conversion into linear text, 175, 261
- Text file encoding parameters, 237
- Token, 41, 71
- Tokenization, 41
- Toolbar, 100
- Transducer, 90
 - inflection, 109
 - rules for application, 130
 - with variables, 98
- Transducer output, 105
 - with variables, 132
- Transducers, 97
- Transduction, 90
- Types of graphs, 109
- Underscore, 98, 132
- Unicode, 31, 101, 243, 269
- Unification variables, 207
- Union of regular expressions, 71
- Union of regular expression, 77
- Uppercase
 - see Respect
 - of lowercase/uppercase, 112
- Using the Apple Java 1.6, 19
- UTF-8, 241, 283, 284
- Variable error policy, 140
- Variable names, 98
- Variables
 - comparison, 137
 - Dictionary entry, 126
 - in graphs, 132
 - in parameterized graphs, 185
 - test, 136
 - within graphs, 98
- Verification of the dictionary format, 239
- Void loops, 115
- Web browser, 81, 142
- Window for ELAG Processing, 162
- Words
 - compound, 43, 72
 - free in Dutch, 256
 - free in German, 256
 - free in Norwegian, 256

- free in Russian, 256
 - in Dutch, 44
 - in German, 44
 - in Norwegian, 44
 - in Russian, 44
 - with space or dash, 49
 - simple, 42, 72
 - unknown, 43, 75
- X11.app, 22
- Xcode, 24
- Zoom, 102