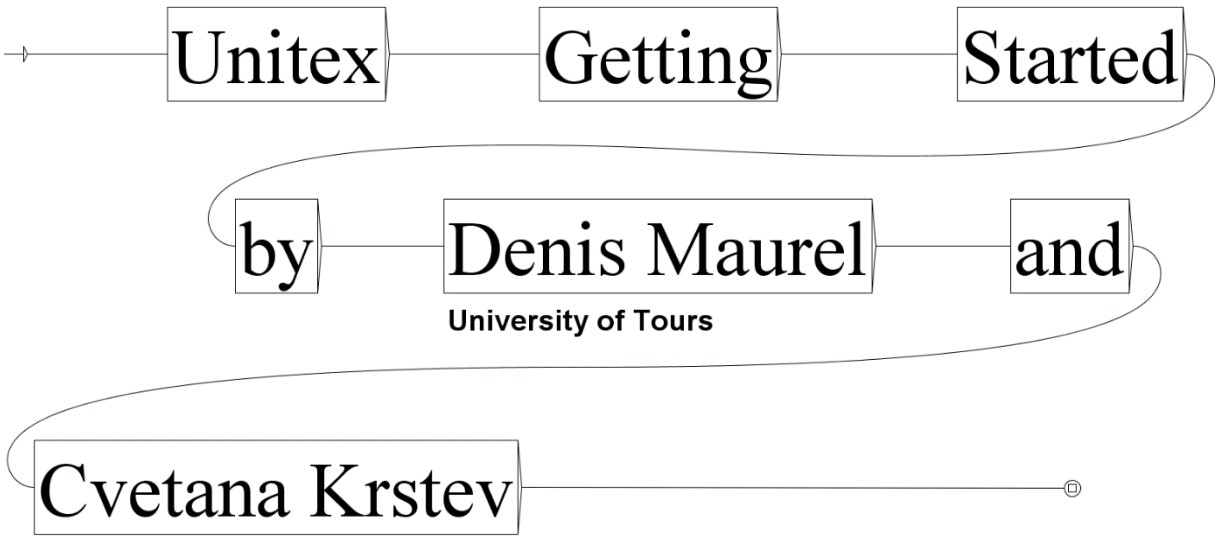


University of Tours

Language Resources and Technologies Society JeRTeh and University of Belgrade

Summary

Chapter 1: introduction

1 A short history

Before introducing the Unitex software, we must go back in time and talk about the *LADL*, *Laboratoire d'automatique documentaire et linguistique*, established in 1968 by the late Professor Maurice Gross.[1]

Maurice Gross introduced the notion of *lexicon-grammar*: he asserted that grammar could not be formalized without its dependence on the lexicon.[2] In collaboration with Professor Dominique Perrin,[3] he proposed the use of the theory of finite state automata (FSA) for some aspects of linguistic description.[4] At the same time, he had entrusted one of his doctoral students, Max Silberztein, with the task of creating the computer tool allowing such use of FSA. As part of his thesis, this student designed software called Intex.[5]

Another doctoral student of Maurice Gross,[6] Sébastien Paumier, wanted to test in his thesis[7] a different algorithm[8] from the one implemented in Intex. As was customary in the nineties, the license to use Intex was not free and its code was not open. Therefore Sébastien Paumier had the idea of creating the free and open software, Unitex, based on this new algorithm, in addition to the features already developed by Max Silberztein.

Since then, Unitex has been enhanced by additions and improvements made by many different developers and users. The main contributors are listed in the Unitex user manual.[9]

2 A quick overview of the book

The aim of this book is to help its readers to get started with Unitex. In this, it differs from the Unitex user manual whose objective is to list the different possibilities of use that Unitex

[1] https://en.wikipedia.org/wiki/Maurice_Gross

[2] Gross M. (1975), *Méthodes en syntaxe*, Hermann, Paris.

[3] https://en.wikipedia.org/wiki/Dominique_Perrin

[4] Gross M., Perrin D. (1989), *Electronic Dictionaries and Automata in Computational Linguistics*, LNCS 377.

[5] Silberztein M. (1989), *Dictionnaires électroniques et reconnaissance lexicale automatique*, Thèse de doctorat (Université Paris VII).

[6] Unfortunately, Maurice Gross passed away before the defense of the Sébastien Paumier's thesis.

[7] Paumier S. (2003), *De la Reconnaissance de Formes Linguistiques à l'Analyse Syntaxique*, Thèse de doctorat (Université de Marne-la-Vallée).

[8] Paumier S. (2003), A time-efficient token representation for parsers, *FSMNLP 2003*, Budapest.

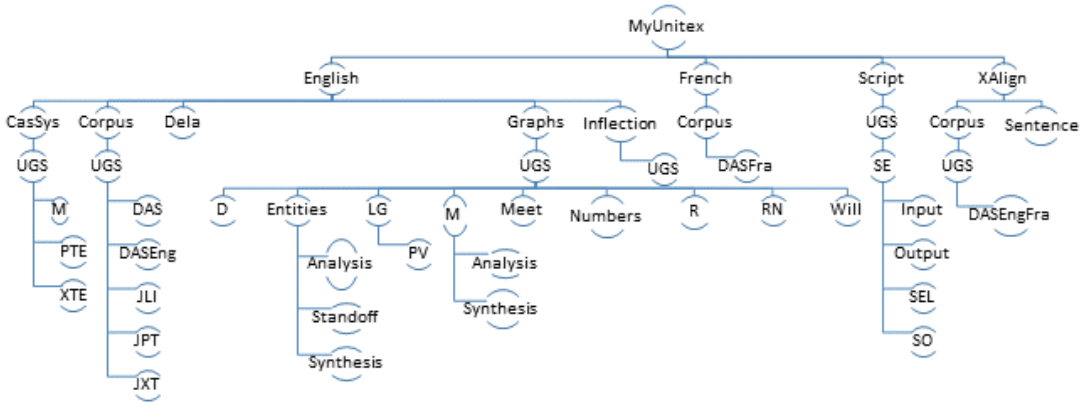
[9] The manual is automatically integrated when downloading Unitex (see Chapter 2, Section 1.1.2, page 7). It is also available at the address: https://unitexgramlab.org/releases/3.3/man/Unitex-GramLab-3.3-usermanual-en.pdf

offers. The objective of this work is an educational approach to Unitex. It covers most of its possibilities, although not all of them. Namely:

This book proposes many exercises and micro-projects that can be implemented step by step by following the text.[10] The different chapters are relatively independent. Screenshots are taken from the Unitex Windows interface. Some minor differences are possible with Mac or Linux interfaces.

3 Organization of correct files

The book comes with a zipped file (*MyUnitex.zip*) including all correct files (graphs, cascades, dictionaries, scripts and so on) answering the exercises. If the reader does not wish to construct her/his own graphs or wishes to verify them, she/he may consult them. The early chapters also include videos detailing the step-by-step construction of the graphs.



D: Dictionaries	JXT: JubileeXmlText	PV: ParticleVerbs	SEL: ScriptEntitiesLingpkg
DAS: DombeyAndSon	LG: LexiconGrammar	R: Repository	SO: StandoffOutput
JLI: JubileeLoremIpsum	M: Measures	RN: RomanNumerals	UGS: UnitexGettingStarted
JPT: JubileePlainText	PTE: PlainTextEntities	SE: ScriptEntities	XTE: XmlTextEntities

[10] Work with Elag Grammars and text automata (or FST) is not covered.

Since these files often have the same name in one section and another, they are stored in folders named by the number of the section. Also, before consulting them, the reader must copy an entire section folder and paste it into the folder hierarchy just above.

For example, the first graph built is in the folder named:

MyUnitex\English\Graphs\UnitexGettingStarted\Will\3.1.1

with an illustrative video. The reader must copy this graph and paste it in the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Will

before opening it with Unitex.

4 A workbook?

For a user already familiar with Unitex, this book can be used differently, as a book of correct examples.

4.1 Graph exercises

Before starting graph exercises
Open the text:*MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon.txt*

Exercise 1

Create a graph that recognizes all forms of the verb *meet*, including verbs with auxiliaries and modal auxiliaries, also including negative and interrogative forms.

The correct solution is in Chapter 2, Section 2.3.6, page 28.

Exercise 2

Create a graph that recognizes and annotates compound verbs using the verbs *will* and *shall*. Use *Verb*, *Adverb* and *Pronoun* tags. *Verb* tags can have an attribute *type* for annotation.[11]

The correct solution is in Chapter 3, Section 1.9.3, page 58.

Exercise 3

Create a graph that recognizes and annotates numbers written with words, from 2 to 999 999. Use non-nested *Number* tags.

The correct solution is in Chapter 3, Section 2.4.2, page 62.

Exercise 4

Create a graph that recognizes and annotates Roman numerals from 1 to 3 999. Use non-nested *RomanNumeral* tags.

The correct solution is in Chapter 3, Section 3.5.2, page 69.

4.2 Cascade exercises

Exercise 5

Before starting exercise 5
Open the text:*MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon.txt*

[11] Affirmative, negative, interrogative and interro-negative forms.

Create a cascade of graphs that recognizes and annotates measures with six different types: *currencies* (pence, sixpence,[12] shilling, pound), *duration* (minute, hour, day, week, month, year), *lengths* (mile, foot, yard), *speed* (mile an hour), *temperature* (degree) and *weight* (ton). Use *Number* and *Measure* tags.

Modify the synthesis cascade to have the *Number* tags only inside the *Measure* tags.

The correct solution is in Chapter 4, Section 3, page 88.

Exercise 6

Before starting exercise 6
Open the text:*MyUnitex\English\Corpus\UnitexGettingStarted\jubileePlainText\jubileePlainText.txt*

Create a cascade of graphs that recognizes and annotates absolute dates (the year is specified), relative dates (the year is not specified), names of towns or cities and names of streets or avenues. Use *AbsoluteDate*, *RelativeDate*, *Town* and *Address* tags.

Modify the synthesis cascade to have non-nested *Address* tags.

The correct solution is in Chapter 4, Section 1, page 71.

Exercise 7

Before starting exercise 7
Open the text:*MyUnitex\English\Corpus\UnitexGettingStarted\jubileePlainText\jubileeXmlText.xml*

Modify the previous cascades so that the annotation is only in the *<body></body>* part.

The correct solution is in Chapter 4, Section 2, page 84.

4.3 Dictionary exercises

Before starting Dictionary exercises
Open the text:*MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon.txt*

Create a dictionary that contains all forms of the following nouns and verbs. Use it to parse the text.

Exercise 8

The beginning of the list:

- *advantage*, *agent*, *firm*: nouns with suffix *s*.
- *branch*, *bus*, *box:* nouns with suffix *es*.
- *accept*, *connect*, *work*: verbs with suffixes *s*, *ed* and *ing*.

The correct solution is in Chapter 5, Section 2.1, page 107.

Continuation of the list:

- *company*, *complexity*, *family*: nouns with suffix *ies*.
- *leaf, thief, calf*: nouns with suffix *ves.*
- *wife, knife, life*: nouns with suffix *ves.*
- *mouse, louse*: nouns with suffix *ice.*
- *narcissus, terminus, tumulus*: nouns with suffix *i.*

[12] Sixpence is the name of an old coin.

- *phenomenon, criterion, datum*: nouns with suffix *a.*
- *love, live, give*: verbs with suffixes *s*, *d* and *ing*.
- *accompany, try, worry*: verbs with suffixes *ies*, *ied* and *ing*.

The correct solution is in Chapter 5, Section 2.2, page 113.

Continuation of the list:

- *goose, tooth*: nouns with suffix *eese or eeth.*
- *foot*: noun with suffix *eet.*
- *man, woman, hypothesis*: nouns in which the penultimate letter changes to *e*.
- *jog*, *plan*, *stop*, *prefer*: verbs with double consonant.
- *slit, split*: verbs with suffixes *s* and *ting*.
- *quiz, gas*: nouns with suffix *zzes* or *sses*.
- *shoot, speed, meet*: verbs for which the double letter (*oo*, *ee*) disappears in the simple past and the past participle.
- *kneel, sleep, creep*: verbs with suffixes *s, elt* or *ept* and *ing.*

The correct solution is in Chapter 5, Section 2.3, page 116.

Exercise 9

End of the list:

- *air of mystery*, *sort of thing*, *tree of knowledge*: MWUs in which only the first component is inflected.
- *crow's nest, death's-head, hair's breadth*: MWUs in which only the last component is inflected.
- *boa constrictor*, *prince regent*, *secretary general*: MWUs in which only the first component is inflected.
- *altar rail*, *Anglo-Norman*, *bad boy*: MWUs in which only the last component is inflected.
- *after-life*, *eye-cup*, *sky-blue*: MWUs in which only the last component is inflected; word tokens can be written together.
- *three quarter*, *blood red*, *self glorification*: MWUs in which only the last component is inflected; hyphen can be added.
- *carry out, take off*, *wash out*: MWUs in which only the last component is inflected; hyphen can be added, space can be omitted.
- *attorney general*, *notary public*: MWUs in which either the first or the second component (but not both!) are inflected.

The correct solution is in Chapter 5, Section 3, page 118.

Exercise 10

Create a dictionary graph that recognizes all Roman numbers from 1 to 3 999. Use it to create a *Word List*.

The correct solution is in Chapter 5, Section 5.1, page 128.

4.4 Lexicon grammar exercise

Exercise 11

Before starting Lexicon exercise 10
Open the text:*MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon.txt*
Open the *particleVerbTable.xlsx* Lexicon-grammar in the folder:
MyUnitex\English\Graphs\UnitexGettingStarted\LexiconGrammar\ParticleVerbs

Create a graph that recognizes all positive forms from the lexicon-grammar *particleVerbTable.xlsx*.

The correct solution is in Chapter 6, Section 1, page 137.

4.5 XAlign exercise

Exercise 12

Align the text:

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSonEng\DombeyAndSonEng.txt

with its translation:

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSonFra\DombeyAndSonFra.txt

The correct solution is in Chapter 6, Section 2, page 141.

4.6 Script exercises

Exercise 13

Create a script to perform the cascade from Exercise 7, page 4.

The correct solution is in Chapter 7, Section 2, page 149.

Exercise 14

Complete the previous cascade to get an inventory of tag occurrences with the name of the file and the information inside *title* tags.

The correct solution is in Chapter 7, Section 4, page 158.

5 Acknowledgement

The authors sincerely thank all those who helped and advised them while preparing this work: Laurence Danlos, Bérengère David, Anne Dister, Nathalie Friburger, Cristian Martínez, Alexis Neme, Benoît Sagot and Duško Vitas. Authors are especially grateful to Éric Laporte and Anne-Lyse Minard who carefully read the entire manuscript.

Chapter 2: first steps

1 Discovering Unitex

1.1 Unitex installation

The Unitex installation file can be found at the URL:

https://unitexgramlab.org

Unitex is a cross-platform software that can be installed on all widely used operating systems (Windows, Mac, Unix and so on). In the following text, we will describe the installation under Windows. Some details shown here may be slightly different on a non-Windows system. For other systems, refer to Section 1.3 of the Unitex user manual downloadable at the URL:[13]

https://unitexgramlab.org/releases/3.3/man/Unitex-GramLab-3.3-usermanual-en.pdf

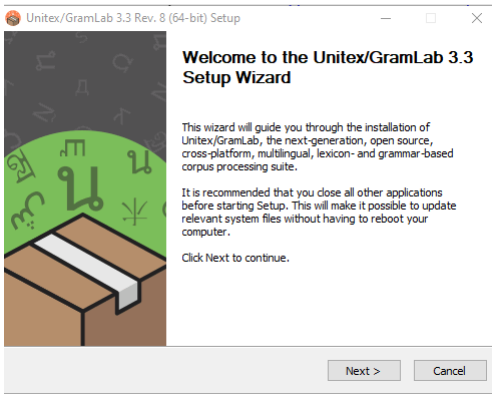
1.1.1 Creating a private working folder

Before launching Unitex for the first time, we must create a private working folder in which we will work. For example, let us go to the *Documents* folder and create a new folder, called for example *MyUnitex* (No spaces and case sensitive):[14]

C:\Documents\MyUnitex

1.1.2 Installation process

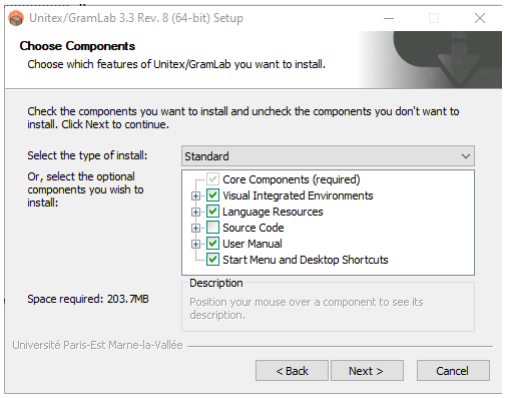
We start by launching the installation file:



[13] There is also a French version of the Unitex user manual, downloadable at the URL:
https://unitexgramlab.org/releases/3.3/man/Unitex-GramLab-3.3-usermanual-fr.pdf

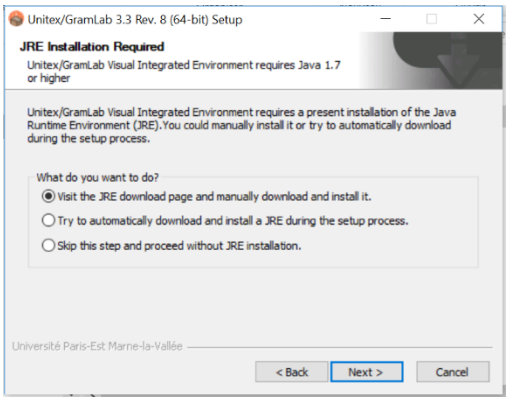
[14] Unitex is a cross-platform software, so filenames should not contain spaces and diacritics, which only Windows accepts. Also, calls to these filenames may be case-sensitive.

We will click the *Next* button, and then twice the *I Agree* button.[15] In the *Choose Components* window, we will get this screen and then we will follow the steps:



1. Click on the check box next to the *Visual Integrated Environments* line to uncheck it, then on the + sign placed on the left of this line to show possibilities and select the *Unitex Java IDE* line. Click on the – sign to the left of the *Visual Integrated Environments* line to hide the possibilities.
2. Click on the check box next to the *Languages Resources* line to uncheck it, then on the + sign placed to the left of this line to show possibilities and select *English* and *French.*[16] Click on the – sign to the left of the *Languages Resources* line to hide.
3. Click on the check box next to the *User Manual* line to uncheck it, then on the + sign placed to the left of this line to show possibilities and select the line *English.*[17] Click on the – sign to the left of the *User Manual* line to hide.

Unitex programs are written in C language, while its interface is written in Java language. Java must therefore be installed on the computer, which is frequently the case. In fact, Unitex favors a certain version of Java, JRE, and tests if it is installed on the computer. However, if it is not, the following window appears:



We have three choices: install Java ourselves, let Unitex install JRE or continue without installing it. If we know that Java is installed on the computer and we do not want to

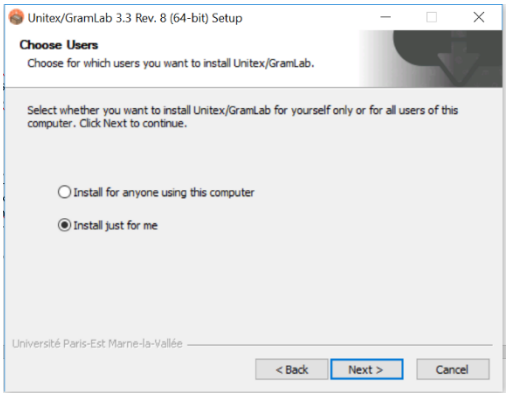
[15] The two licenses (for the programs and for the resources) are LGPL licenses that authorize their free use including the use in commercial products.

[16] In addition, some other languages, of course, if you want. We will use French in the chapter dealing with aligned texts (Chapter 6, Section 2, page 141).

[17] You can select the *French* manual as well if you prefer or check only *French*.

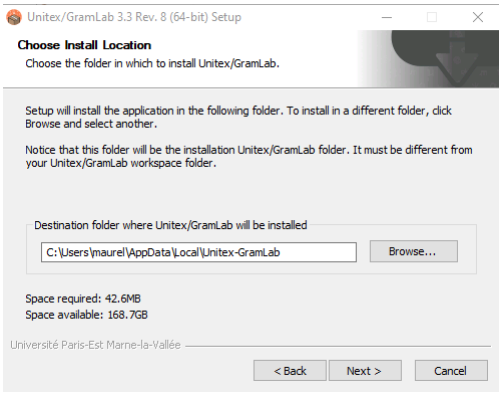
additionally install the JRE version, we should choose the third option. Otherwise, we advise you to choose the second option and let Unitex try to install it during the setup process.

After selecting the second option click on the *Next* button. If you are the administrator of your computer, Unitex offers you an installation only for you or an installation for all users of the computer:



If you choose the *just for me* option, Unitex offers the installation in the User folder:

C:\Users\userName\AppData\Local\Unitex-GramLab[18]

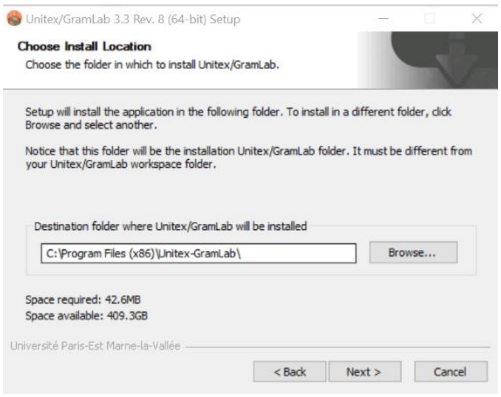


If you choose the *for anyone* option, Unitex offers the installation in the Program Files (x86) folder:[19]

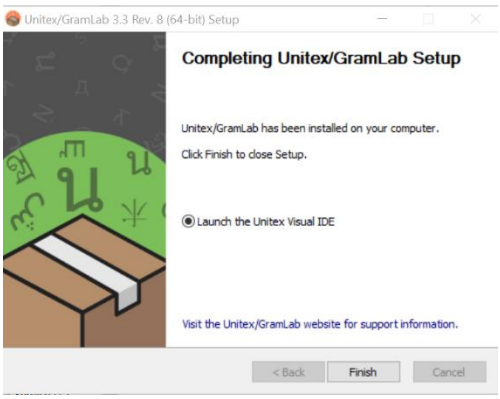
C:\Program Files (x86)\Unitex-GramLab\App

[18] *userName* is your name under the Windows system.

[19] To choose this option, you must be an IT administrator.



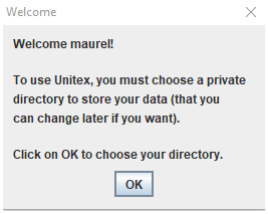
Both options are fine. If you want to change this setting, don't install Unitex in the *MyUnitex* folder that you have created: this folder must remain separate from the installation folder. After choosing the folder for installation that is appropriate for your computer, click on the *Next* button, and after that on the *Finish* button:



Unitex starts automatically; if that does not happen, see Section 1.2.1 that follows.

1.1.3 Initial choices

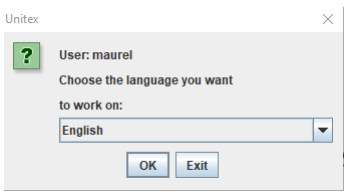
Unitex will then ask you to choose your private working directory (or working folder):



Select then the *MyUnitex* folder that you have already created (see Section 1.1.1, page 7):

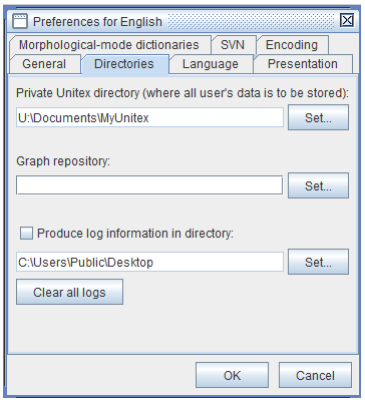
C:\Documents\MyUnitex

And then choose the language (in this case, *English*) and click on the *OK* button:

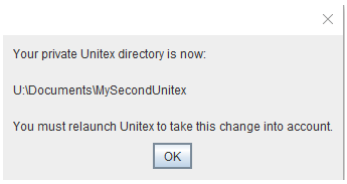


1.1.4 Check or modify your private working directory

If you want to check your private working directory, you can open the *Info/Preferences…/Directories* menu.



To modify your private working directory, click the *Set* button to choose a new one, and click on the *OK* button. Unitex must then be closed and reopened to take the changes into account.



It is thus possible to have several private working directories, one per project.

The name of your private working directory is saved in a specific file which depends on where Unitex is installed.

If you chose the *just for me* option, Unitex save this name in the file:

C:\Users\userName\AppData\Local\Unitex-GramLab\Users\userName.cfg

Else, if you chose the *for anyone* option, Unitex save this name in the file:

C:\Users\userName\.unitex.cfg

To avoid opening and closing Unitex to change the private working directory, it is possible to directly modify this file. You can also create a shortcut to it on the desktop.

1.2 Using Unitex

1.2.1 Starting Unitex

It can happen that Unitex is not launched at the end of the setup or that you have accidentally closed Unitex. It is also possible that you are using a computer on which Unitex is already installed. In these three cases, you must launch Unitex yourself, as you will always do when

wanting to work with it. The easiest way is to use the shortcut placed on the desktop during installation and double click it:[20]

If you do not have this shortcut or it does not work, it is possible to launch Unitex by going to the folder where the Unitex program has been installed:

C:\Program Files (x86)\Unitex-GramLab\App
or
C:\Users\userName\AppData\Local\Unitex-GramLab\App

Double-click on the file named *Unitex* having the type *Executable Jar File*. A permanent solution should be to send this file to the desktop as a shortcut by right clicking on this file and choosing the *Send to/Desktop* option. This will create a shortcut of the *Unitex executable jar file*.

1.2.2 The Unitex inteface

Unitex processes different types of files: texts, dictionaries, graphs and so on. In the Unitex interface, each menu is specific to one of them and therefore has, for instance, an *Open* sub-menu (or option).

Menus	types of files
Text	text to analyze
DELA	dictionary
FSGraph	graph
Lexicon-Grammar	lexicon-grammar
XAlign	parallel texts
File Edition	textual file to edit

In addition, there are three more menus: *Windows*, *Help* and *Info*. The Unitex user manual can be opened directly from Unitex, via the *Help* menu.

[20] Whenever you update Java, you will need to redo this shortcut by restarting the installation or changing its target.

1.2.3 Opening a text

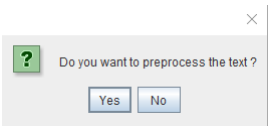
Before starting this section

Go to the folder: *MyUnitex\English\Corpus\UnitexGettingStarted*
Create a new folder named *DombeyAndSon* (without space).
Go to the folder: *MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon*[21]
Download the *DombeyAndSon.txt* file.

We will click the *Text/Open…* menu to open the file named:

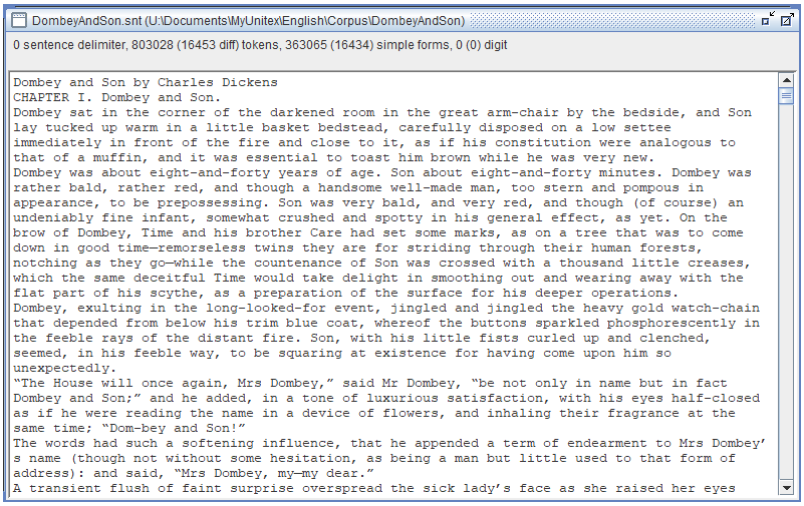
MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon.txt

and answer *No* to the question *Do you want to preprocess the text?*



Unitex will create a new file: [22]

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon.snt



The top bar displays some numbers: *0 sentence delimiter, 803 028 (16 453 diff) tokens, 363 065 (16 434) simple forms, 0 (0) digits.*[23] Their meaning is:

sentence delimiter: Unitex can insert the special symbol *{S}* at the end of a sentence during the preprocessing (see Section 3.3.1, page 37). Since we skipped the preprocessing there are no such symbols.

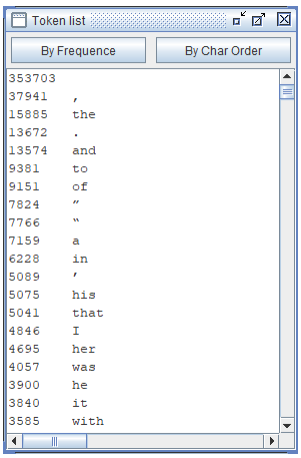
[21] This file is the novel *Dombey and Son*, by English author Charles Dickens. According to Wikipedia: "It follows the fortunes of a shipping firm owner, who is frustrated at the lack of a son to follow him in his footsteps; he initially rejects his daughter's love before eventually becoming reconciled with her before his death".

[22] Unitex never works on an original text file.

[23] These numbers are recorded in the file named:
MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon_snt\stats.n

- *token*: it is either a string of letters[24] or any other single character (including spaces). There are 803 028 tokens in this novel, but only 16 453 different.
- *simple form*: it is a string of letters. There are 363 065 of them in this novel, but only 16 434 different.

At the bottom left is a reduced window, the *Token list*, that lists the tokens of the text and their number of appearances. By double-clicking on it, we obtain this list in decreasing order of frequency. You can also have it in the alphabetical order by clicking the *Char Order* button.[25]



We can notice that this *Token list* begins with space, punctuation characters and functional words, which is normal, because they are the most frequently used tokens.

1.2.4 Applying dictionaries

We will open the *Text/Apply Lexical Resources…* menu. There we will find available dictionaries, on the left those in the private working folder, on the right those that come with the Unitex distribution.[26] The default dictionaries are preselected (*dela-en-public.bin* and *Dnum.fst2*).

---

[24] Letters of a language are defined in the file:

MyUnitex\English\Alphabet.txt

A sorting order, for arranging concordances and dictionaries, is specified in the file:

MyUnitex\English\Alphabet_sort.txt

If these files are not present, Unitex will not work. In this case you have to open the folder where Unitex is installed:

C:\Users\maurel\AppData\Local\Unitex-GramLab\English

or

C:\Program Files (x86)\Unitex-GramLab\English

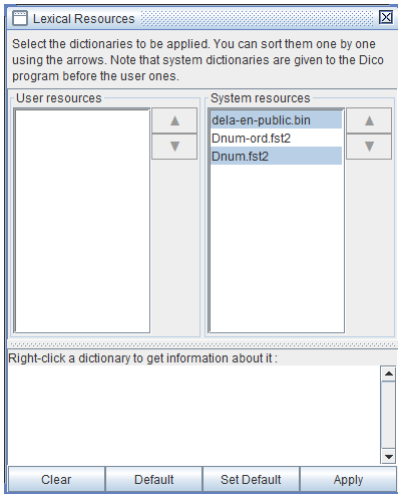
and copy the three files *Alphabet.txt*, *Alphabet_sort.txt* and *Norm.txt*, and then paste them in the private folder *English*:

MyUnitex\English

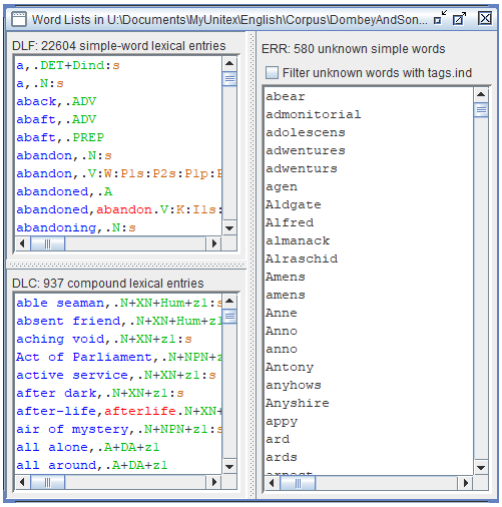
[25] These lists are recorded in the three files:

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon_snt\tokens.txt
MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon_snt\tok_by_freq.txt
MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon_snt\tok_by_alph.txt.

[26] We have not yet dictionaries in the private working folder, see Chapter 5, page 107.



By clicking the *Apply* button, we apply the dictionaries to the text and we obtain the *WordList*.



The *WordList* is displayed in three windows:[27] *DLF* (22 604 *simple forms*), *DLC* (937 *compounds* or *multiword units*) and *ERR* (580 words that are not in applied dictionaries).

1.2.5 The DLF window

We will first look at the format of the eleventh line of the *DLF* window (the first line not displayed in the above screenshot):

abandoning,abandon.V:G

At the beginning of the line is the word from the text, followed by its heading form (or lemma), after the comma and before the period. After the period comes a word's part of speech (*V* for verb), and it is followed, after the colon, by the inflection code (*G* for gerund). This word from the text is recognized as possibly being the gerund of the verb *to abandon*. If the lemma form

[27] These windows correspond to the following files:

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon_snt\dlf
MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon_snt\dlc
MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon_snt\err

They come with three additional files:

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon_snt\dlf.n
MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon_snt\dlc.n
MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon_snt\err.n

These contain just the number of entries in the corresponding files.

does not appear, which is the case on the line just above the analyzed one where the period comes right after the comma, it means that the lemma is identical to the text word itself; in this case the lemma of the noun *abandoning* is simply *abandoning*.

It may seem strange to see two lines beginning with *abandoning*. We must keep in mind that words out of context are ambiguous, and *abandoning* can be the noun *abandoning* or the gerund of the verb *to abandon*.

It is possible to force one analysis rather than another by modifying the text. For instance, by replacing *after abandoning herself* with *after {abandoning,abandon.V:G} herself*, we explicitly assign the verb category to the word *abandoning*. This expression between curly braces is counted as a single token and is called a *lexical tag* (However, if you want to do that manually, you shall have to close the text with Unitex, edit the *.txt* file, save it and reopen it with Unitex).

A line can also contain one or more features for the entries, which are preceded by a plus sign. One example can be seen in the first line of the *DLF* window where the *Dind* features is used to indicate that the determiner is indefinite:

a,. DET+Dind:s

1.2.6 The DLC window

The *DLC* window has the same format as the *DLF* window. If the multiword form contains a character used as a delimiter (comma, period, plus, colon and slash),[28] it must be preceded by a backslash. Such a case does not occur in our *DLC* window.

1.2.7 The ERR window

The words that were not recognized by the applied dictionary are mostly proper names (*Aldgate*, *Alfred*, *Alraschid* …); but also foreign language words: *Anno* (Latin), Roman numbers, unusual orthography (*adwentures*), parts of contracted verbal forms (*didn*, *hasn…*) or simply words that were not recorded in the applied dictionaries (*nevyless*,[29] *unbusiness*, *undauntable…*).

2 A detailed example

Before starting this section

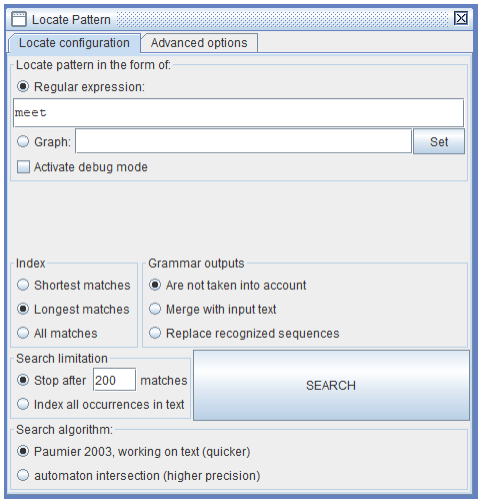
Go to the folder: *MyUnitex\English\Graphs*
Create a new folder named *UnitexGettingStarted* (without space).
Go to the folder: *MyUnitex\English\Graphs\UnitexGettingStarted*
Create a new folder named *Meet*.

2.1 Concordances with one word query

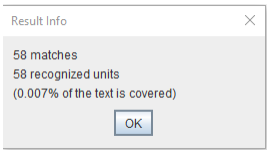
Suppose, for example, that we want to study the use of the verb *to meet* in our novel. In order to do that we will open the *Text/Locate Pattern* menu, select the *Regular expression* option and type *meet* in the corresponding field.

[28] The slash enables comments, see Chapter 5, Section 4.3.1, page 125.

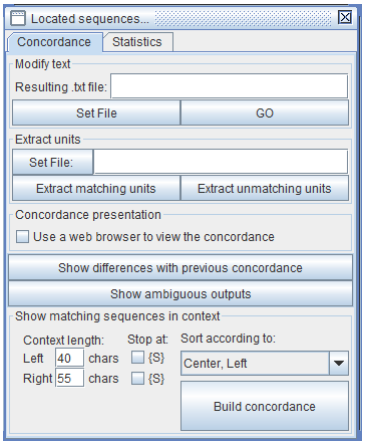
[29] The word *nevy* is an old form of *nephew*.



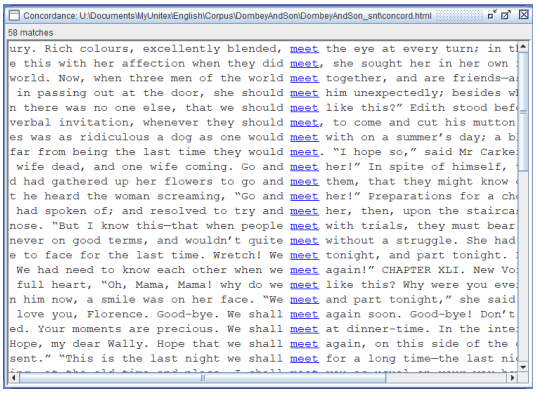
We will launch the search by clicking the *Search* button, which will yield 58 matches.



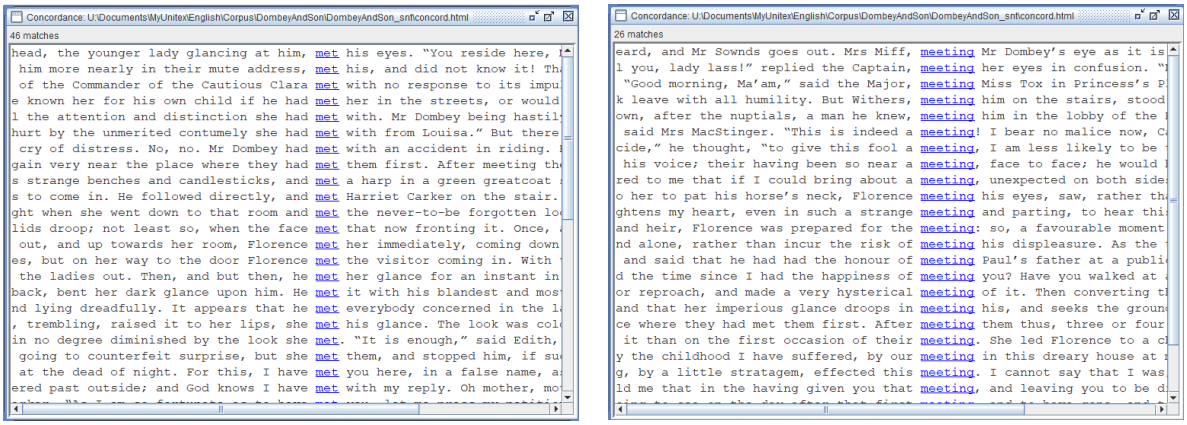
Clicking the *OK* button will open the *Located sequences* dialog box. In order to obtain concordances, we need to click the *Build concordance* button (situated at the bottom right of the *Located sequences* dialog box).



The produced list is a hypertext file in which each recognized sequence leads to the corresponding location in the text.[30] For instance, if we click on the first line, this opens the following window (right):



As the verb *to meet* has conjugated forms *meets*, *met* and *meeting*, we would have to repeat what we have just done three more times in order to obtain a concordance of these three forms as well.



If we look closely at the concordance of the word *meeting*, we can see that some lines represent a form of the verb *to meet (Mrs Miff, meeting Mr Dombey's eye…)*, while others represent the noun *meeting* (*This is indeed a meeting!*). You will remember that in Section 1.2.5, page 15, we explained that some words can have several interpretations.

2.2 Concordances with a lemma

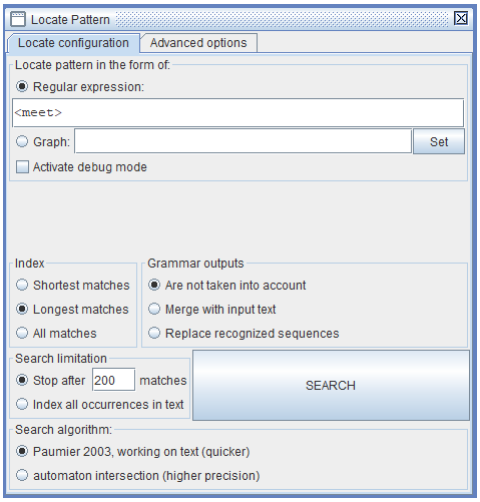
Instead of launching four different searches, we could have obtained the same result with one query in the form of the regular expression *meet+meets+meeting+met*. With this query, we

[30] These concordances are registered in the file:
MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon_snt\concord.html
With each new search and concordance produced, this file is replaced, but you can save it using a different name if you want to keep it. The saved concordance can be reopened by the *Text/Open Concordance* menu.

would obtain a list of 131 concordance lines, containing all the lines previously obtained in four subsequent runs (58+1+46+26). However, all these queries ignore the fact that Unitex uses dictionaries, containing among other entries, all forms of the verb *to meet*.[31]

```
…
meet,.A
meet,.N:s
meet,.V:W:P1s:P2s:P1p:P2p:P3p
meeting,.N:s
meeting,meet.V:G
meets,meet.N:p
meets,meet.V:P3s
…
met,meet.V:K:I1s:I2s:I3s:I1p:I2p:I3p
…
```

Instead of launching four searches, or listing all forms of the verb, we can use one query with the lemma form of the verb, *<meet>*.[32]



With this query, we will obtain the same result, a list of 131 concordance lines.

2.3 Concordances with a graph query

Until now, only simple forms of the verb *to meet*, one of the four possible forms, appeared in the concordances. But we might like to highlight longer sequences in concordance lines: the sequences containing, beside forms of the verb *to meet*, also words that we see in the left context of concordance lines: *did meet*, *should meet*, *would meet*, *shall meet*, *will meet*, *would seldom meet*, *may ever meet*, *had met*, *have met*, *being met*, *had been met*, *have been met*, *had ever met*, *had never met* and *has met*. The reason to switch from regular expressions to using Unitex graphs will become obvious with these examples.[33]

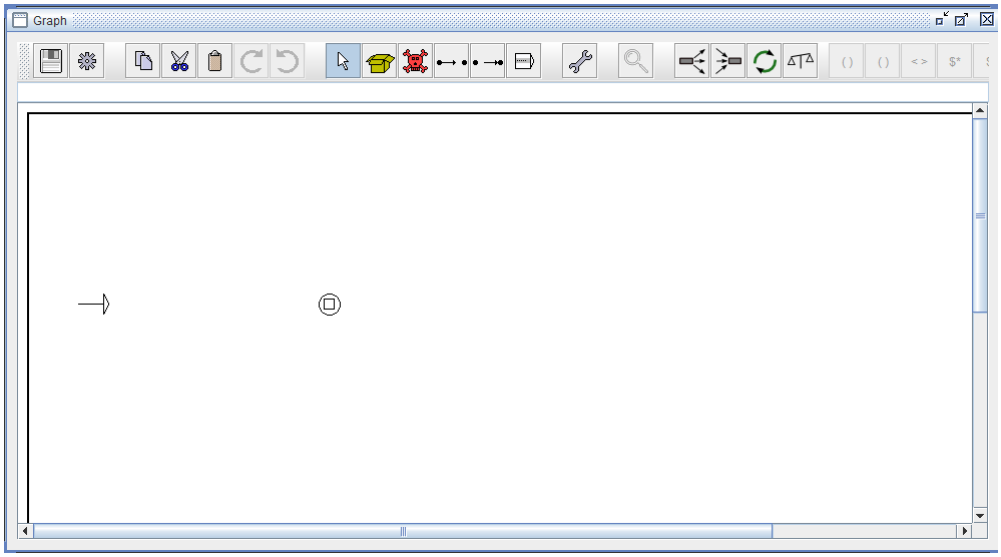
[31] Note that the noun *meetings* (plural form of *meeting*) is not present in this text.

[32] Note that this does not remove the ambiguity, because *<meet>* is also the lemma of a noun and an adjective.

[33] For the sake of simplicity, we will omit the modal form (*may ever meet*) and the emphatic past (*did meet*).

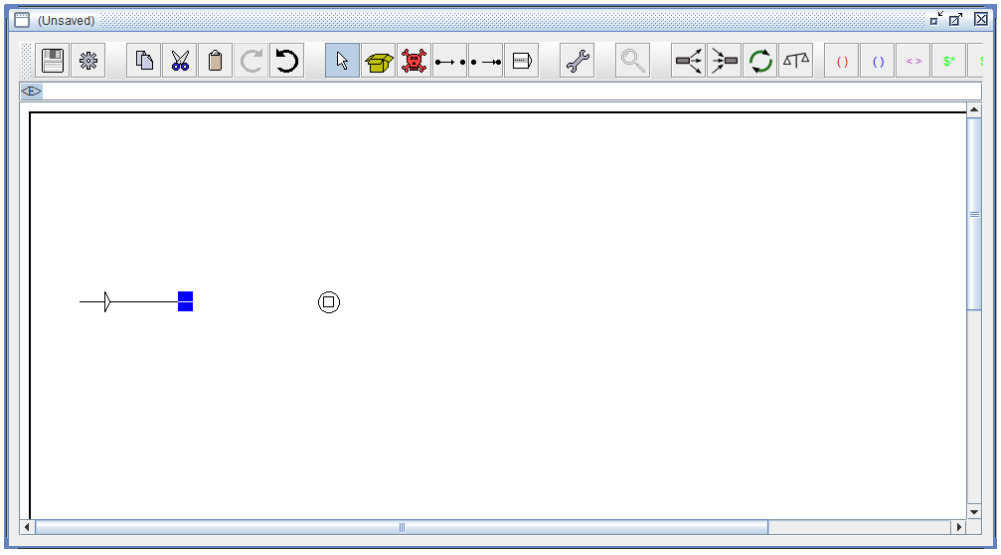
2.3.1 A first graph

Let us open the *FSGraph/New* menu. We will get a window with a starting point on the left and an ending point on the right.



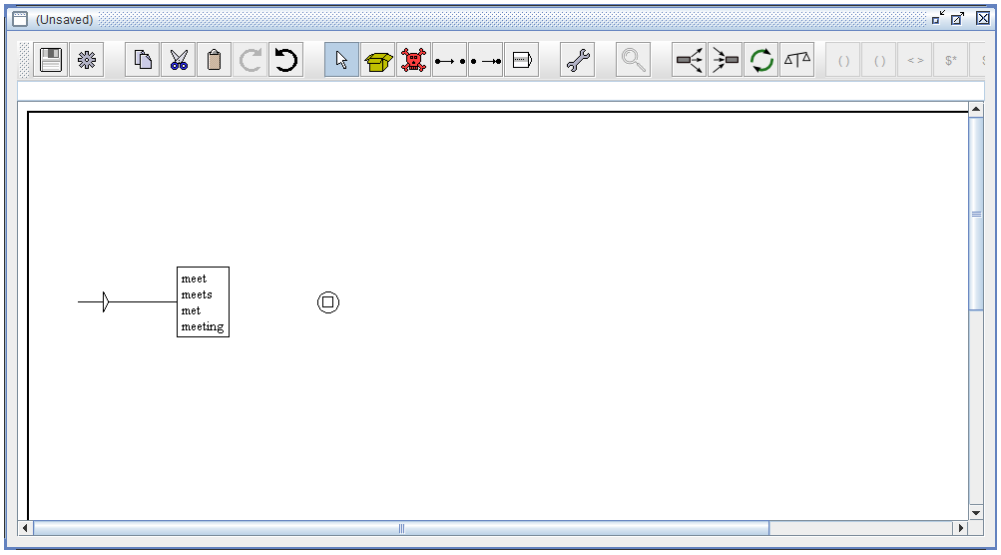
We will now create in five steps a graph that recognizes four word forms *meet*, *meets*, *met* and *meeting*:

1. Click on the starting point (start box).
2. Position the cursor a little more to the right, click the right mouse button and select *Create box* from the displayed menu.

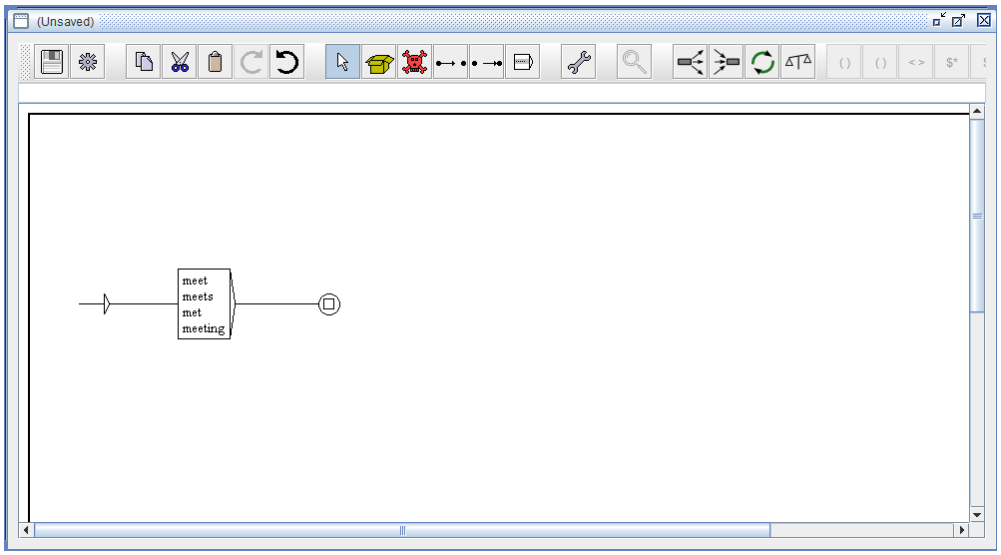


3. Type without any space *meet+meets+met+meeting*[34] and finish with clicking the *Enter* key.

[34] These words will appear in the formula bar, just below the task bar of the graph window. They will replace the <*E*> symbol, which means *empty*. The plus sign means *or*.



4. Click on the *meet+meets+met+meeting* box and then the ending point (the end box).



5. Click on the *Save* button (the leftmost in the task bar). Unitex will open the Graph folder to save it.[35]

Go to the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Meet

And save this graph in it using the name *meet*. Unitex will add the *.grf* extension.

[35] The whole process is shown in the video *2.2.3.1.meet.mp4* which is in the folder:
MyUnitex\English\Graphs\UnitexGettingStarted\Meet\2.2.3.1

Notes

1. Clicking on a box, then on another one creates a path between them or, if it already exists, removes it.

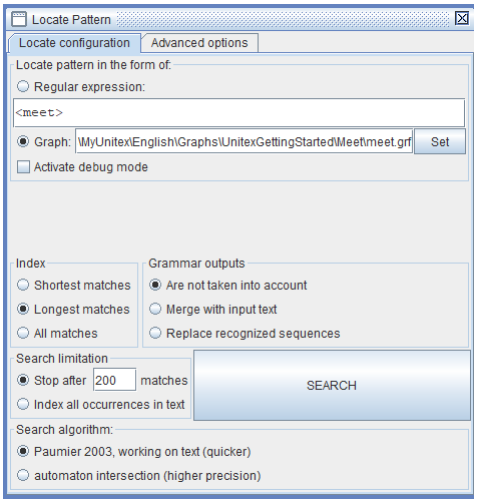
2. To delete a box, you must delete its content (by using the *Delete* key, for example), then validate with clicking the *Enter* key.

2.3.2 Concordances

In order to produce a concordance with our first graph, we will open the *Text/Locate Pattern* menu and click the *Set* button next to the *Graph* field. The *Graphs* folder will open and we will move to the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Meet

where we will find and select the *meet.grf* file.

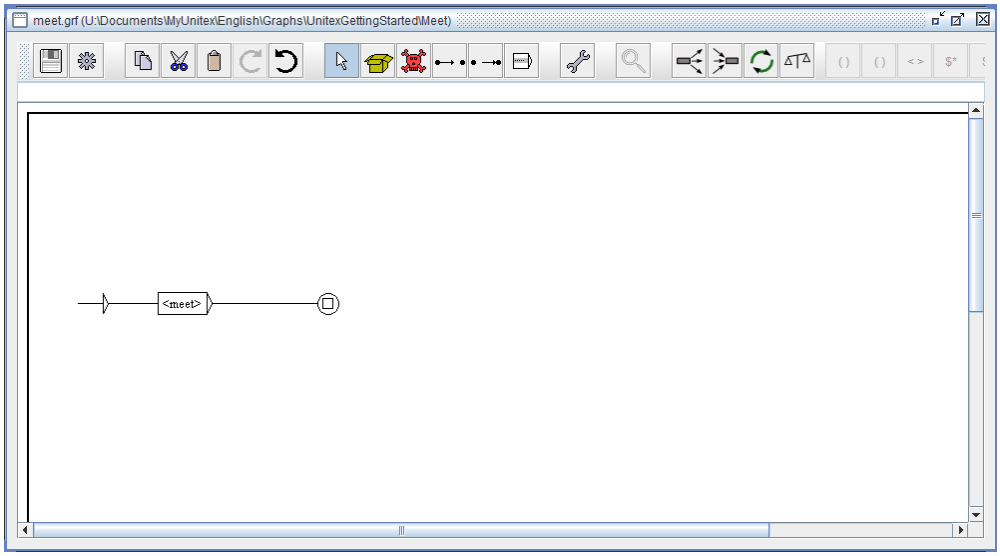


We will launch the search, as before, by clicking the *SEARCH* button. Finally, we will obtain the same list of 131 concordance lines.[36]

As before, we can replace four word forms with their common lemma.

1. Select the meet+meets+met+meeting box, type <meet> instead of meet+meets+met+meeting then press the *Enter* key.
2. Click the diskette icon, at the left end of the task bar, to save the graph.

[36] Note that *Locate Pattern* creates a new file, *meet.fst2*: it is a compiled version of the *meet.grf* graph. If we used the compiled version of a graph in *Locate Pattern* after modifying it, the modified version would not be used for search.



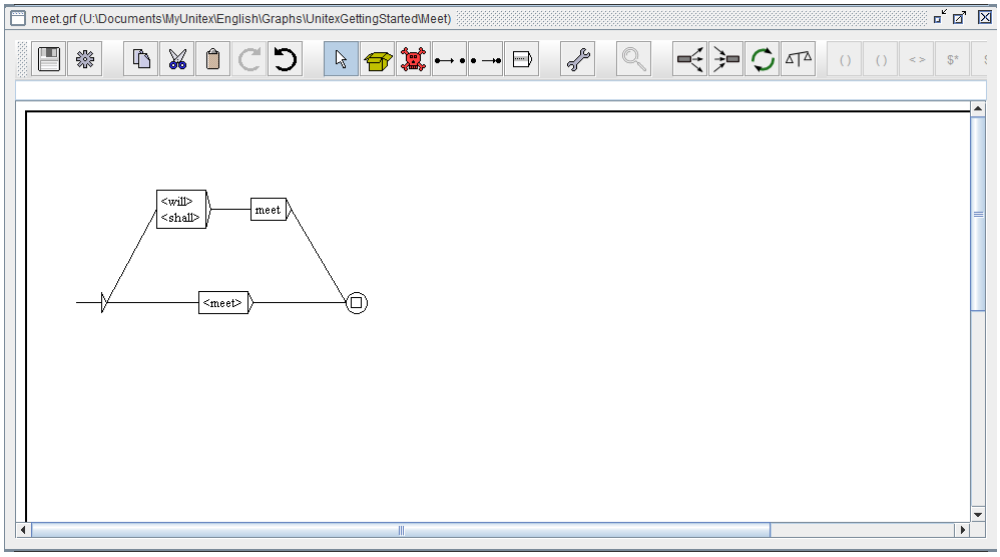
Now, we will begin to highlight longer sequences in concordance lines by adding the future tense and the conditional, i.e. four examples in our concordance: *shall meet*, *will meet*, *should meet* and *would meet*.

2.3.3 Adding the future tense and the conditional

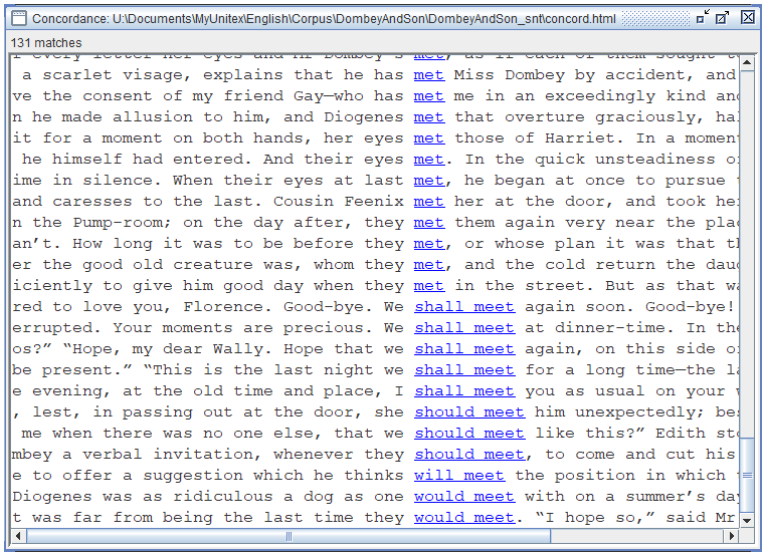
The future and the conditional use the forms of the auxiliaries, *will*, *would*, *shall* and *should*, which can be represented by two lemmas *<will>* and *<shall>*. We will add two boxes to our graph:

1. Click on the starting box, then move the cursor up and to the right, click the right mouse button and select *Create box* from the displayed menu.
2. Type *<will>+<shall>*, then press the *Enter* key.
3. Click on the *<will>+<shall>* box, then move the cursor right, click the right mouse button and select *Create box* from the displayed menu.
4. Type *meet*, then press the *Enter* key.[37]
5. Click on the *meet* box and then on the final box.
6. Rearrange the graph.
7. Click the Save button.

[37] Pay attention: *meet* without angle brackets.



The concordance produced with this new *meet.grf* graph contains again 131 lines and among them lines with highlighted *shall meet*, *should meet*, *will meet* and *would meet*. You will find these occurrences at the end of the concordance list.



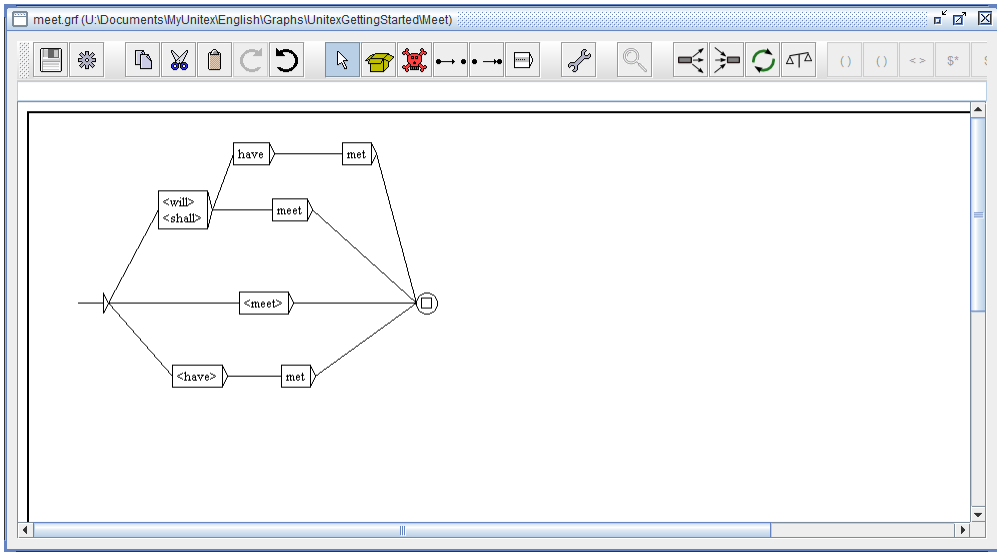
Let us move to the perfect tenses. They consist of the verb *to have* followed by the past participle, *met*. Three such forms can be spotted in our concordance, in the left context of the form *met*: *had met*, *have met* and *has met*.

2.3.4 Adding the perfect tenses

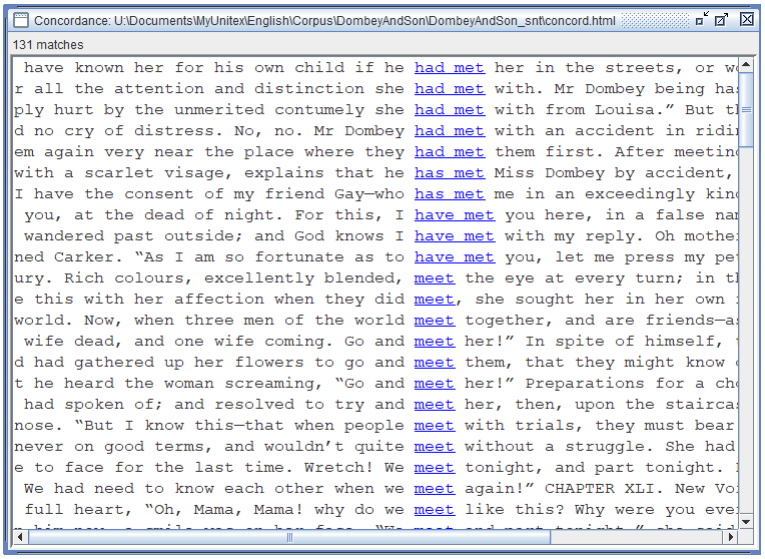
The perfect tenses use the auxiliary forms *have*, *has*, *had* and *having*, which can be represented by the lemma *<have>*. We will enhance our graph by using the copy-paste

technique. We will add four boxes to our graph, one for the *<have>* lemma, one for the *have* form, and two for the *met* form:

1. Click on the starting box, then move the cursor down and to the right, click the right mouse button and select *Create box* from the displayed menu.
2. Type *<have>*, then press the *Enter* key.
3. Click on the *<have>* box, then move the cursor right, click the right mouse button and select *Create box* from the displayed menu.
4. Type *met*, then press the *Enter* key.
5. Click on the *met* box and then on the end box.
6. Select the *<have>* box and the *met* box by drawing with the mouse a rectangle that encloses both boxes, then click the *Copy* button (or use *CTRL-C* on the keyboard).
7. Click the *Paste* button (or use *CTRL-V* on the keyboard): the two boxes appear twice.
8. Move the copied boxes to the top of the graph. Click outside the graph to deselect them.
9. Click on the top *<have>* box and delete the angular brackets (*have* replaces *<have>*), then press the *Enter* key.
10. Click on the *<will>+<shall>* box and then on the *have* box.
11. Click on the *met* box and then on the end box.
12. Rearrange the graph.
13. Click the *Save* button.



The concordance produced with this graph contains the lines with five highlighted sequences *had met*, two sequences *has met* and three sequences *have met*.



Two important notes on how copy-paste functions in Unitex

1. If only one box is selected, the box is not copied, only its content is. In order to copy-paste a single box, you have to copy the box, create a new one and paste the content.

2. Copying and pasting multiple boxes does not use the usual clipboard. Therefore, there may be competition between pasting text and pasting boxes. If some operation pastes boxes when you wanted to paste text, just erase the unwanted boxes by clicking the *Delete* key, then the *Enter* key. If you have copied both boxes and text and if you want to paste the text into some box, there is a trick that ensures that what is pasted is the text and not the boxes: just type or erase some character in the box before pasting the text.

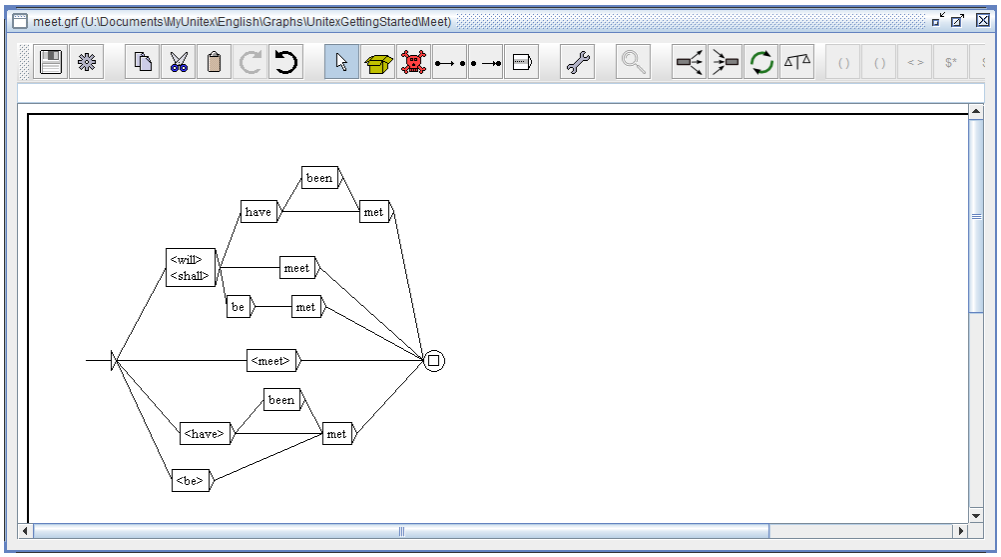
It remains to insert the passive forms to our graph. They consist of the verb *to be* followed by the past participle, *met*. Three such forms can be found in our concordance, if we look at the left context of *met*: *being met*, *had been met* and *have been met*.

2.3.5 Adding the passive forms

The passive forms use the forms of the auxiliary *to be* (*am*, *are*, *was*, *were*, *be* and *being*) which can be represented by their lemma *<be>*.

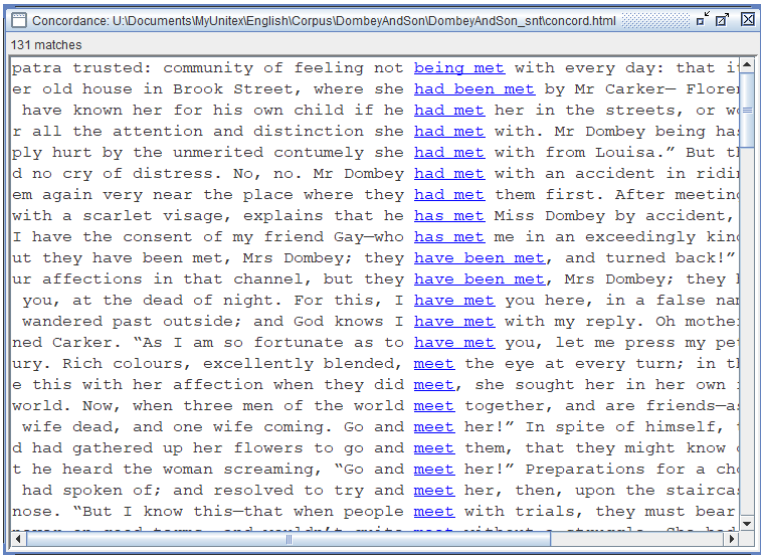
We will add five boxes to our graph, one for the *<be>* lemma, one for the *be* form and one for the *met* form, two for the *been* form:

1. Select the whole graph by drawing with the mouse a rectangle that encloses all the boxes (or use *CTRL-A*[38] on the keyboard) and then move it down. Click somewhere outside the graph to deselect it.
2. Click on the starting box, then move the cursor down and to the right, click the right mouse button and select *Create box* from the displayed menu.
3. Type *<be>*, then press the *Enter* key.
4. Click on the *<be>* box and then on the *met* box.
5. Click on the *<have>* box, then move the cursor up on the right, click the right mouse button and select *Create box* from the displayed menu.
6. Type *been*, then press the *Enter* key.
7. Click on the *been* box, then click on the *met* box.
8. Click on the *have* box, then move the cursor up and to the right, click the right mouse button and select *Create box* from the displayed menu.
9. Click the *Paste* button (or use *CTRL-V* on the keyboard): *been* is pasted. Press the *Enter* key.
10. Click on the *been* box and then on the *met* box.
11. Click on the *<will>+<shall>* box, then move the cursor down and to the right, click the right mouse button and select *Create box* from the displayed menu.
12. Type *be*, then press the *Enter* key.
13. Click on the *be* box, then move the cursor to the right, click the right mouse button and select *Create box* from the displayed menu.
14. Type *met*, then press the *Enter* key.
15. Click on the *met* box and then on the end box.
16. Click the *Save* button.



The concordance produced with this graph highlights sequences *being met*, *had been met* and twice *have been met*.

[38] After the selection of two boxes by drawing with the mouse a rectangle that encloses both.



If we look closer at our concordance, we will notice that sometimes an adverb is inserted between an auxiliary and a main verb, as in the sequences *would seldom meet*, *had ever met* and *had never met* which are therefore not recognized.

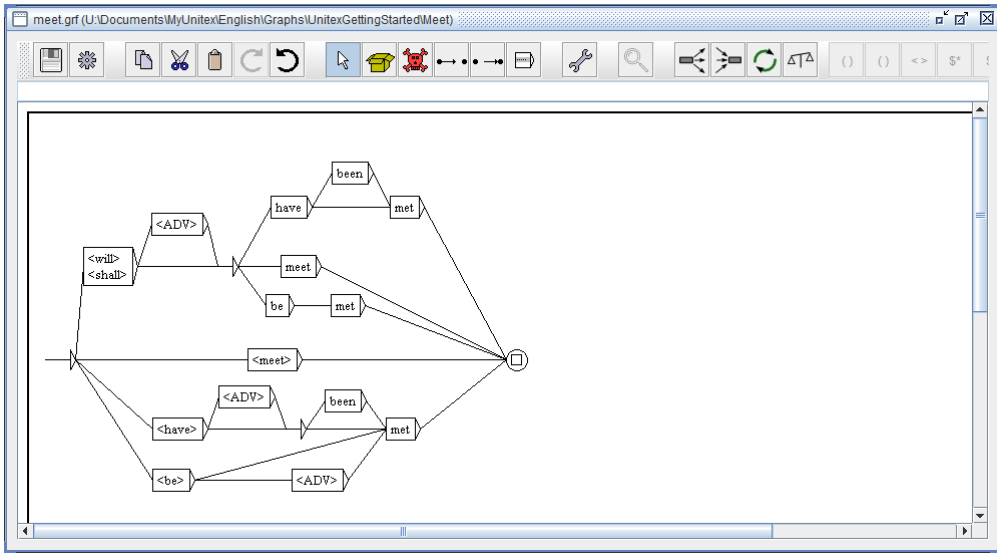
2.3.6 Adding adverbs

In order to retrieve these sequences as well, we can insert in our graph a box containing *seldom*, *ever* and *never*. However, we should try to be more general because almost any adverb can occur at this position. If we look closer at the *Word List* (see Section 1.2.4, page 14), we will see that its third line is:

aback,.ADV

We learn from it that the code for adverbs used in dictionaries is *ADV*. We will therefore add a box containing *<ADV>* at appropriate positions in our graph.

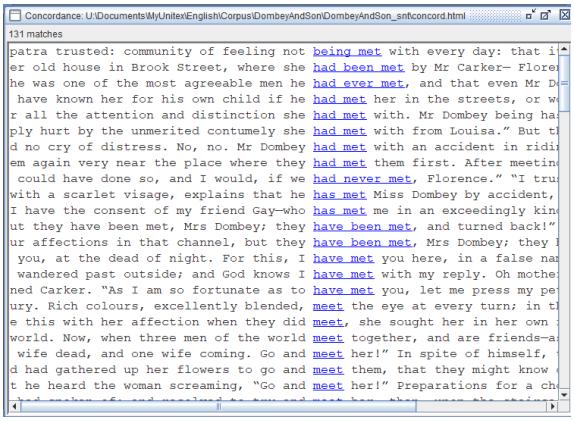
1. Click on the *<be>* box, then move the cursor to the right, click the right mouse button and select *Create box* from the displayed menu.
2. Type *<ADV>*, then press the *Enter* key.
3. Click on the *<ADV>* box and on the *met* box.
4. Rearrange the graph.
5. Click on the *<have>* box, then click the *Copy* button (or use *CTRL-C* on the keyboard). Click somewhere in the blank space to deselect it.
6. Click on the start box, then move the cursor down and to the right, click the right mouse button and select *Create box* from the displayed menu.
7. Click the *Paste* button (or use *CTRL-V* on the keyboard): *<have>* is pasted. Then press the *Enter* key.
8. Click on the previous *<have>* box and type *<E>*.[39]
9. Click on the new *<have>* box and on the empty box.
10. Click on the start box and on the empty box. [40]
11. Click on the *<have>* box, then move the cursor up and to the right, click the right mouse button and select *Create box* from the displayed menu.
12. Type *<ADV>*, then press the *Enter* key.
13. Click on this new *<ADV>* box and on the empty box.
14. Rearrange the graph.
15. Click on the *<will>+<shall>* box, then click the *Copy* button (or use *CTRL-C* on the keyboard). Click somewhere in the blank space to deselect it.
16. Click on the start box, then move the cursor up and to the right, click the right mouse button and select *Create box* from the displayed menu.
17. Click the *Paste* button (or use *CTRL-V* on the keyboard): *<will>+<shall>* is typed. Then press the *Enter* key.
18. Click on the previous *<will>+<shall>* box and type *<E>*.
19. Click on the start box and on the new empty box.
20. Click on the *<will>+<shall>* box and on the new empty box.
21. Click on the *<will>+<shall>* box, then move the cursor up and to the right, click the right mouse button and select *Create box* from the displayed menu.
22. Type *<ADV>*, then press the *Enter* key.
23. Click on the new *<ADV>* box and on the new empty box.
24. Rearrange the graph.
25. Click the *Save* button.

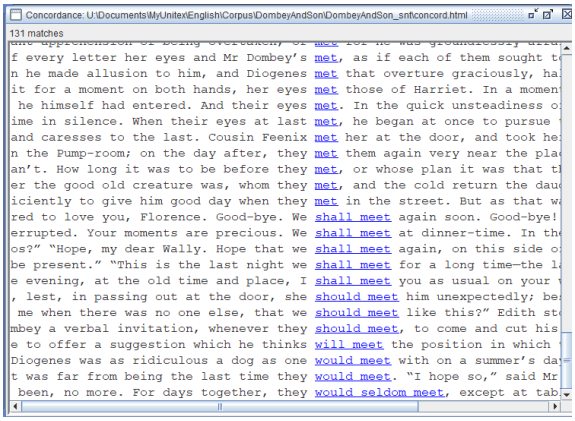


[39] Remember that the *<E>* symbol means *empty*.

[40] This will remove the existing path.

Finally, the concordance produced with this graph highlights the sequences *had ever met*, *had never met* and *would seldom meet*.





2.4 Contexts

2.4.1 *Left context*

2.4.1.1 *The verb* to meet *preceded by a pronoun*

Suppose that instead of displaying all occurrences of the verb *to meet*, we only want to display occurrences in which the verb is preceded on the left by a pronoun (or a pronoun followed by an adverb). We will erase one box and add two boxes in our graph:

1. Open the *FSGraph/Save as…* menu to rename the graph to *meetLeft.grf*.
2. Select the whole graph by drawing with the mouse a rectangle that encloses all the boxes, except the starting box, and then move it to the right. Click somewhere in the blank space to deselect it.[41]
3. Rearrange the graph to align vertically the *<will>+<shall>* box, the *<meet>* box, the *<have>* box and the *<be>* box.[42]

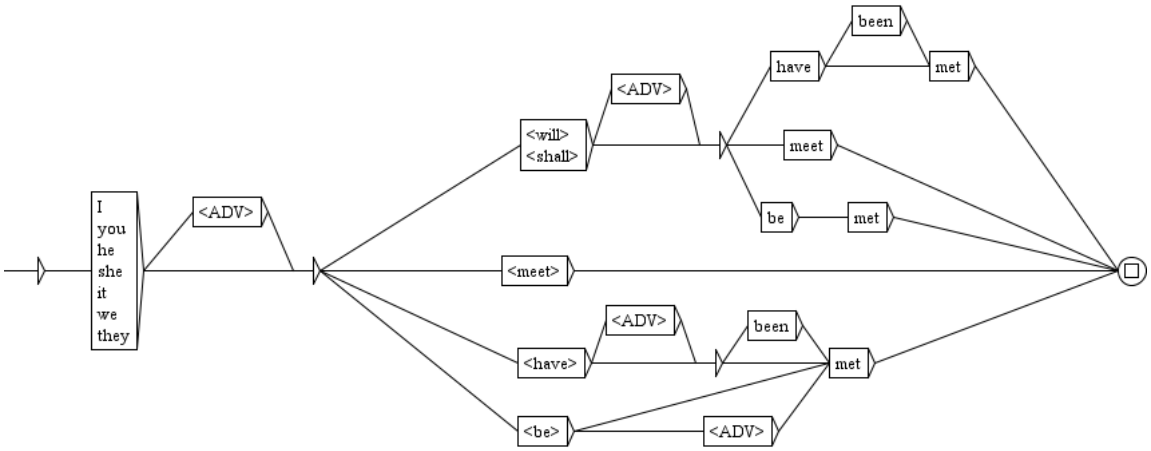
4. Select the *<will>+<shall>* box, the *<have>* box and the *<be>* box, then click the *Reversed link between boxes* button in the tool bar, in the third group of options, fifth in the group. Click on the starting box (this will remove three paths).

5. Click the *Normal editing mode* button in the same third group of options, first in the group.
6. Click on the starting box, then move the cursor to the right, click the right mouse button and select *Create box* from the displayed menu.

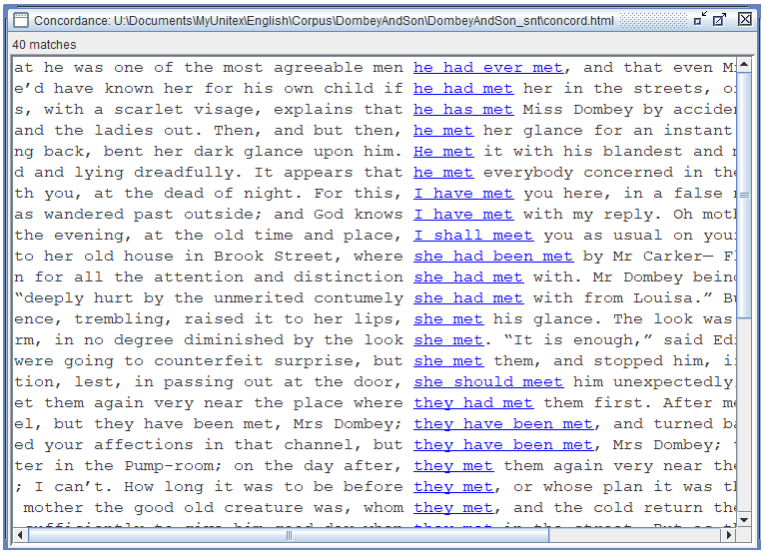
[41] Or select any two boxes by drawing with the mouse a rectangle that encloses both, then select the whole graph by typing *CTRL-A* on the keyboard and then move it right. Click somewhere in the blank space to deselect it. Click on the starting box, then move it left.

[42] To align boxes vertically (or horizontally), we can also select them and right-click to open the *Format/Alignment…* menu.

7. Click on the newly created box and type *I+you+he+she+it+we+they*, then press the *Enter* key.[43]
8. Click on the *I+you+he+she+it+we+they* box, then move the cursor to the right, click the right mouse button and select *Create box* from the displayed menu. This box will remain empty.
9. Click on the *I+you+he+she+it+we+they* box, move the cursor up and to the right, then click the right mouse button and select *Create box* from the displayed menu.
10. Click on the newly created box and type *<ADV>*, then press the *Enter* key.
11. Click on the *<ADV>* box and on the empty box.
12. Select the *<will>+<shall>* box, the *<meet>* box, the *<have>* box and the *<be>* box.
13. Click the *Reversed link between boxes* button. Click on the empty box (this will create three paths).
14. Click the *Normal editing mode* button.
15. Click the *Save* button.



This graph will produce the concordance with forty matched sequences.[44]



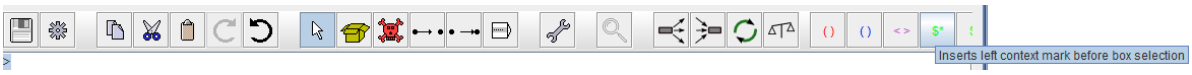
[43] If you wish to use the corpus *Ivanhoe.txt*, distributed with Unitex, you must add to this list the second person singular, *thou*. However, *thou* never precedes a form of the verb *to meet* in the text we use.

[44] Do not forget to use the new *meetLeft.grf* graph in *Locate pattern*.

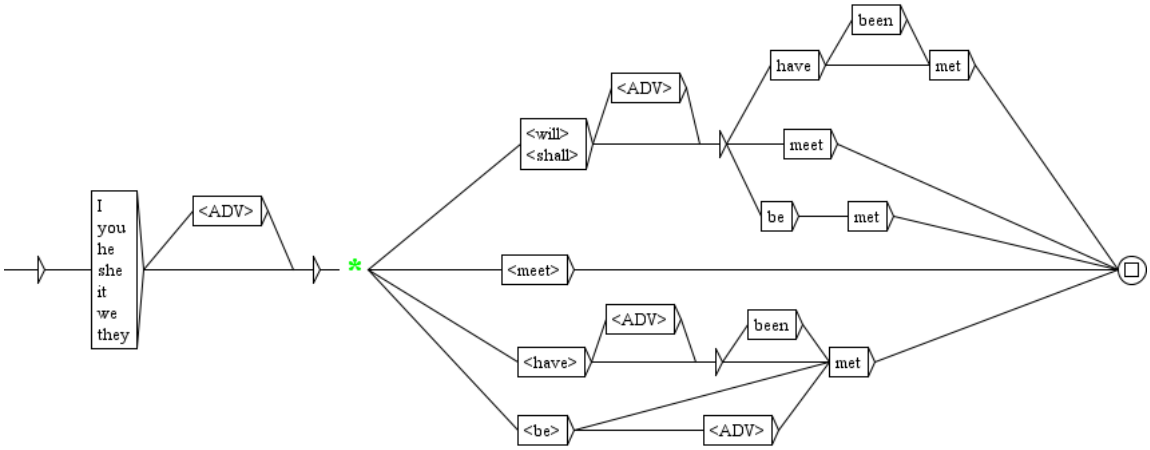
2.4.1.2 The verb to meet preceded by a pronoun and a left context

The concordance we obtained is not like the previous ones, because pronouns are part of the highlighted sequences. It is possible to avoid this if we use the so-called left context:

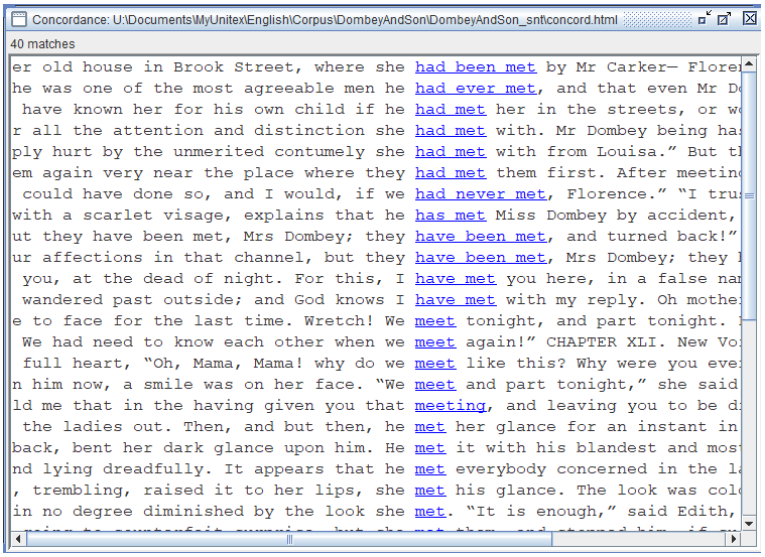
1. Select the *<will>+<shall>* box, the *<meet>* box, the *<have>* box and the *<be>* box.
2. Click the *Reversed link between boxes* button in the tool bar, in the third group of options, forth in the group.



3. Click the first green button (*$**, *Inserts left context mark before box selection*) situated at the right end of the graph editor toolbar.
4. Rearrange the graph.
5. Click the *Save* button.



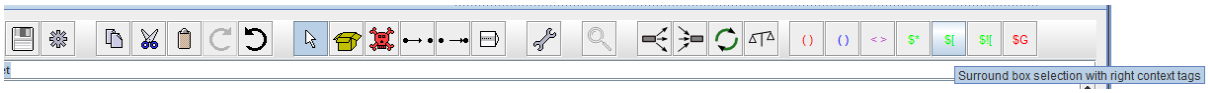
We obtain nearly the same concordance: pronouns are not highlighted anymore, but occurrences are still displayed only when the pronoun is present.



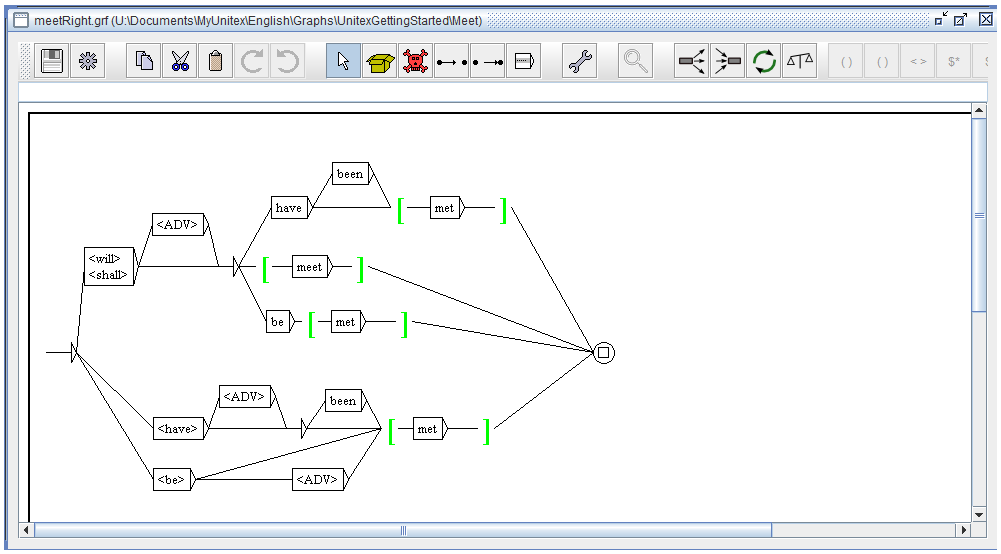
2.4.2 Right context

It is possible to do the same thing on the right. It is done in a slightly different way. We will start from the *meet.grf* graph and suppose that we want to highlight only the left context of the verb *to meet* (the auxiliaries) and not the verb itself.

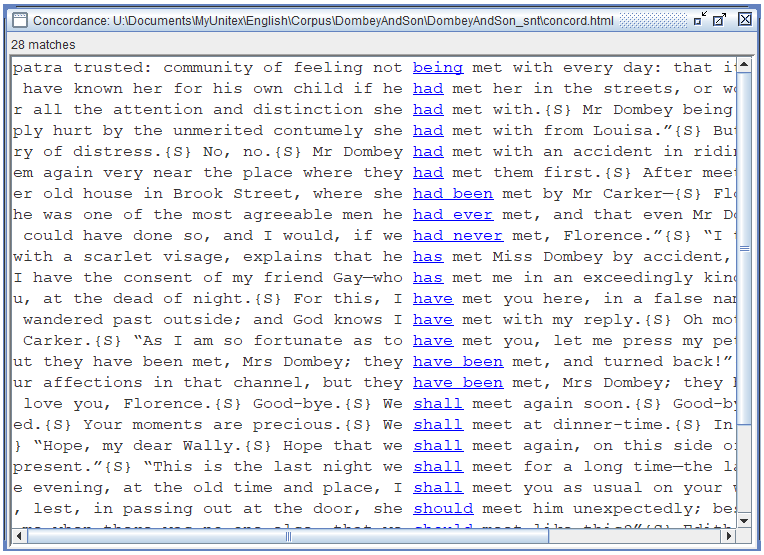
1. Open the *meet.grf* graph.
2. Open the *FSGraph/Save as…* menu to rename the graph as *meetRight.grf*.
3. Click on the <*meet*> box and type *Delete* and *Enter*.



4. Select the *meet* box and click the second green button (*$[, Surround box selection with right context tags*) situated at the right end of the graph editor toolbar.
5. Select the *met* box and click again the *Surround box selection with right context tags* button (three times).
6. Rearrange the graph.
7. Click the *Save* button.



The concordance obtained by applying this graph contains 28 matches (among 131 matches obtained by the *meet.grf* graph, 103 recognize only the main verb *to meet* in its various forms). In the recognized sequences, the verb *to meet* is not highlighted.



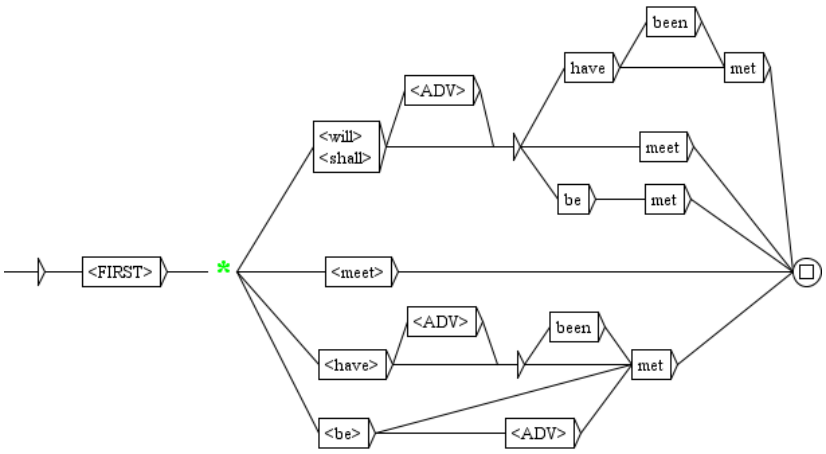
2.4.3 Lexical masks

We have already introduced two types of lexical masks: for instance, *<meet>* matches all forms of the lemma *meet* (see Section 2.2, page 18) while *<ADV>* matches all adverbs (see Section 2.3.6, page 28). These masks rely on the dictionaries used.

There are other masks depending on the use of upper and lower case letters:[45] *<UPPER>* for a word in uppercase, *<FIRST>* for a word starting with uppercase and *<LOWER>* for a word in lowercase. Besides them, the mask *<WORD>* corresponds to a sequence of letters; the mask *<NB>* to a sequence of digits; the mask *<DIC>* to any dictionary entry; and the mask *<TOKEN>* to any token (see Section 1.2.3, page 13).[46]

We will now transform the *meetLeft.grf* graph to recognize all forms of the verb *to meet* that are preceded by a word starting with uppercase. We will name this graph *meetLeftRight.grf*:

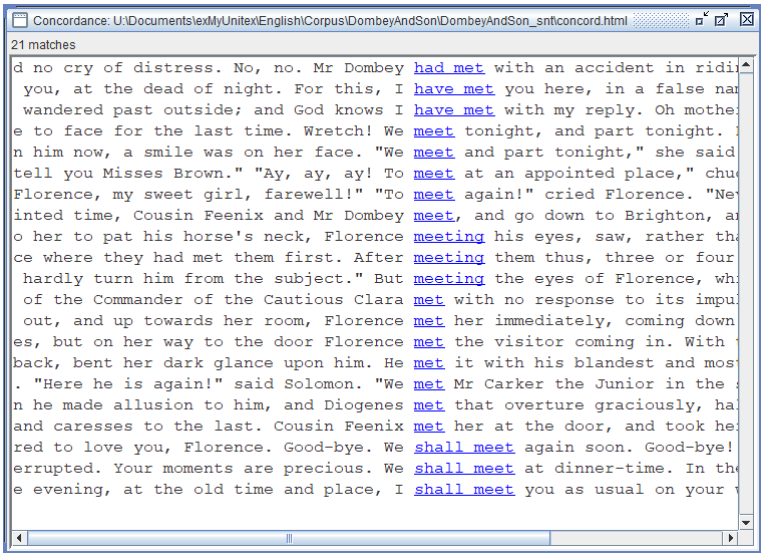
1. Open the *meetLeft.grf* graph.
2. Open the *FSGraph/Save as…* menu to rename the graph as *meetLeftRight.grf*.
3. Click on the *I+you+he+she+it+we+they* box and type *<FIRST>*.
4. Select the *<ADV>* box and the empty box by drawing the rectangle over these boxes, then type *Delete* and *Enter*.
5. Click on the *<FIRST>* box and the *$** box.
6. Rearrange the graph.
7. Click the *Save* button.



Now, the concordance displays only twenty-one lines.

[45] The uppercase and lowercase letters are defined in the Alphabet.txt file, see Section 3.3.3.1, page 41.

[46] There are some other masks, see the Unitex user manual, Section 4.3.1. Beware that for the preprocessing, the topic of the next section, different masks are valid (see Section 2.5.2 in the Unitex user manual).



3 The .snt file

3.1 Document transformation

3.1.1 Standard normalization

Before splitting the text into tokens (see Chapter 2, Section 1.2.3, page 13), Unitex normalizes the text. This new text is saved with the same name as the original text, but with the *.snt.* extension. It is constructed as follows:

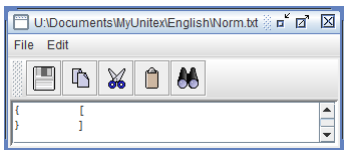
1. Any sequence of spaces and tabulation characters is replaced by a single space (recognized by a path between two boxes).
2. Any series of line breaks (possibly with spaces and tabs) is replaced by a single line break (also recognized by a path between two boxes).[47]
3. Braces are replaced by square brackets, unless they correspond to a dictionary entry (a *lexical tag*, see Section 1.2.5, page 15).

3.1.2 The Norm.txt *file*

The *Norm.txt* file in the Unitex distribution is important:

MyUnitex\English\Norm.txt

We are going to open this file using the *File Edition/Open/Other files* menu. It only contains two lines.



[47] The *No separator normalization* option, in the *Preprocessing* dialog box, only works for preprocessing graphs. For other graphs, a path on a graph between two boxes may pass not only through one space or one line break, but also through a series of spaces, tabs and line breaks.

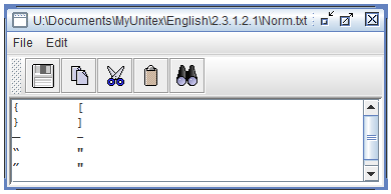
These two lines define the rule for transforming braces into brackets. Each line consists of a character string (the text to be replaced), followed by the tabulation character, then another character string (the replacement text) and finally a line break (*Enter*).

Note that this file is strictly speaking not necessary, because this rule applies anyway, even if the *Norm.txt* file is not present. It is actually there to indicate how to add more lines.

3.1.2.1 The standardization of dashes and quotation marks

There are two types of dashes in our text: 2 034 *hyphens* (-) and 2 378 *em dashes* (—). Similarly, there are three types of quotation marks in our text: 7 766 *left double quotation marks* (“), 7 824 *right double quotation marks* (”) and 2 *quotation marks* (").[48] If we would like to normalize them to hyphens (-) and quotation marks ("), we can add three lines to our *Norm.txt* file:[49]

1. Open the *Norm.txt* file.
2. Type —, *Tab*, – and *Enter*.[50]
3. Type “, *Tab*, " and *Enter*.
4. Type ”, *Tab*, " and *Enter*.
5. Click the *Save* button.



Now, close Unitex and reopen it. Open the text (the original *.txt* text, not the already processed *.snt* text) and skip all preprocessing to get 4 412 hyphens (-) and 15 592 quotation marks ("). However, if we look at the number of different tokens, we can notice that we have three unique tokens less than when processing the text with the old *Norm.txt* file: 16 450. Three unique characters were replaced with two existing unique characters.

3.1.2.2 The standardization of apostrophes

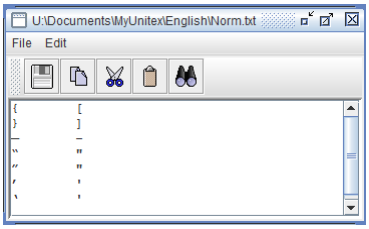
Looking closer, we see that the novel uses the *right single quotation mark* (’) as an apostrophe and not the keyboard character, the *punctuation apostrophe* ('). The novel also uses the left single quotation mark (‘). If we want to normalize these two single quotation marks to punctuation apostrophe, we can add two new lines to our *Norm.txt* file:

1. Open the *Norm.txt* file.
2. Type ’, *Tab*, ' and *Enter*.
3. Type ‘, *Tab*, ' and *Enter*.
4. Click the *Save* button.

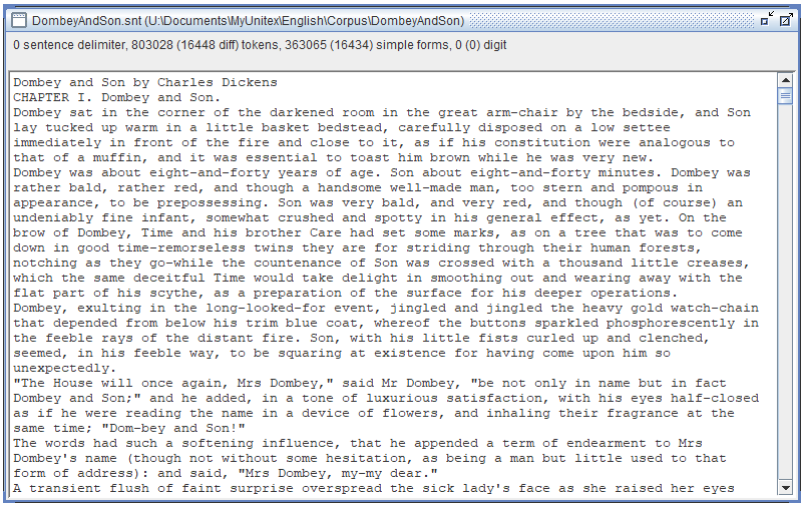
[48] There are more than two types of dashes and three types of quotation marks according to the Unicode standard; however only those mentioned here appear in our text. If you do not know how to type these characters, the best thing is to locate them in the text and copy and paste them.

[49] In order to apply the modified *Norm.txt* file, it is necessary to close Unitex and then to reopen it.

[50] For example, by copying and pasting non-ASCII characters from the novel.



Now, close Unitex and reopen it. Open the text (the original *.txt* text, not the already processed *.snt* text) and skip all preprocessing.



If we compare the .snt text from Section 1.2.3, page 13, we can see that five tokens are missing (16 453 different tokens versus 16 448).

3.2 The preprocessing

Unitex offers two possibilities of preprocessing:

1. Switch on one or two graphs, one working in *Merge* mode and one in *Replace* mode. These graphs can recognize a line break, which is represented by <^> in a box. After normalization, the line break will not be distinguished from the space and will be recognized by a path between two boxes.
2. Launch the default dictionaries (see Section 1.2.4, page 14).

By default, switching on *Merge* mode will use the graph:

MyUnitex\English\Graphs\Preprocessing\Sentence\Sentence.grf

while switching on *Replace* mode will use the graph:

MyUnitex\English\Graphs\Preprocessing\Replace\Replace.grf

3.3 Material (only for confident users)

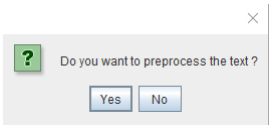
3.3.1 *The* Sentence.grf *graph*

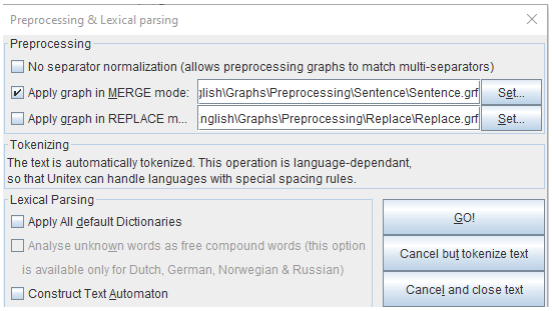
The *Sentence.grf* graph inserts the Unitex symbol *{S}* (*sentence delimiter*) at the end of the sentence.

We will open the file:

MyUnitex\English\Corpus\UnitexGettingStarted\UnitexGettingStarted\DombeyAndSon\DombeyAndSon.txt

by using the *Text/Open…* menu and then answer *Yes* to the question *Do you want to preprocess the text?* By default, two boxes are checked. If we check only the box next to *Apply graph in MERGE mode*

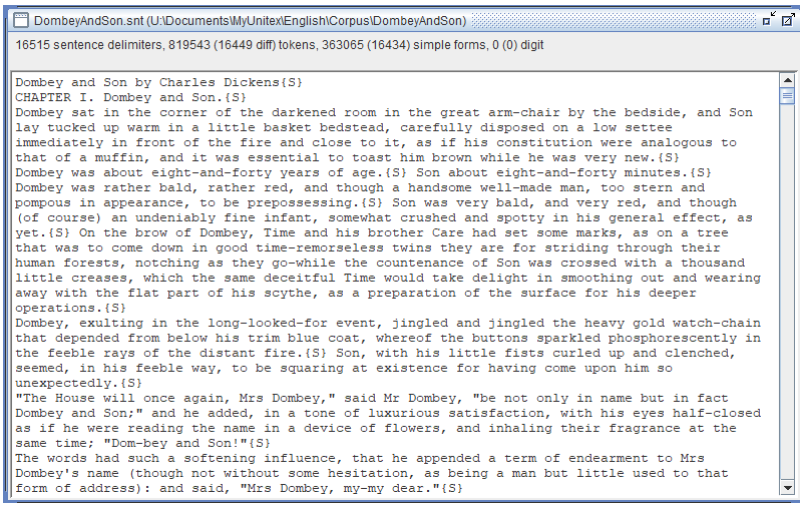




the displayed information for the file:

MyUnitex\English\Corpus\UnitexGettingStarted\UnitexGettingStarted\DombeyAndSon\DombeyAndSon.snt

will change.

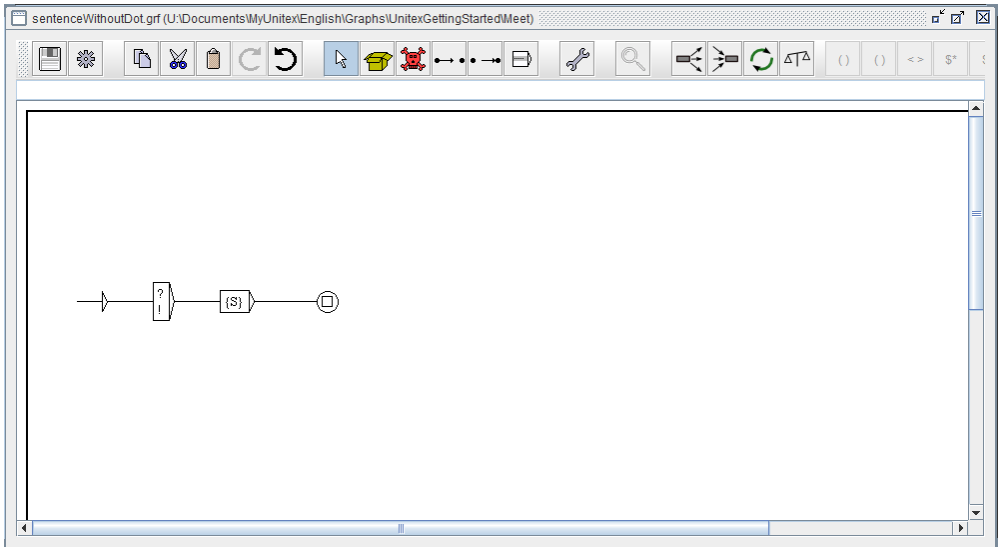


Unitex has inserted 16 515 sentence delimiters.[51]

A *sentence delimiter* can be placed and recognized in a box of a graph. For instance, create the *sentenceWithoutDot.grf* graph and record it in the same folder as the *Meet.grf* graph:

1. Open the *FSGraph/New* menu.
2. Click on the starting box, then move the cursor to the right, click the right mouse button and select *Create box* from the displayed menu.
3. Click on the newly created box and type *?+!*, followed by *Enter*.
4. Click on the *?+!* box, then move the cursor to the right, click the right mouse button and select *Create box* from the displayed menu.
5. Click on the newly created box and type *{S}*, followed by *Enter*.
6. Click on the *{S}* box and on the final box.
7. Click the *Save* button.

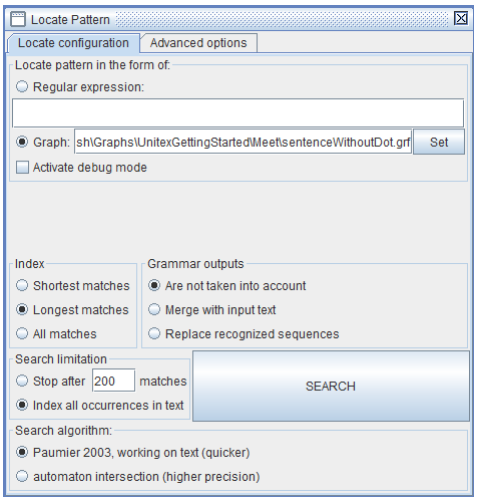
[51] Since the last sentence is not followed by the *sentence delimiter*, there are therefore 16 516 recognized sentences. The number of tokens is now 803 028+16 515=819 543, because {S} counts as one token. There is also one more different token.

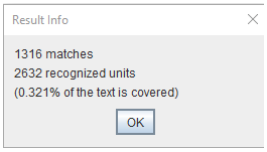


In order to produce a concordance with this graph, we will open the *Text/Locate Pattern* menu and click the *Set* button next to the *Graph* field. The *Graphs* folder will open, and we will move to the folder

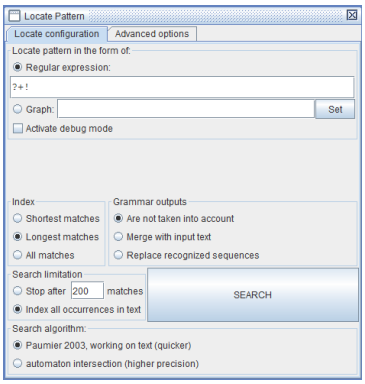
MyUnitex\English\Graphs\UnitexGettingStarted\Meet

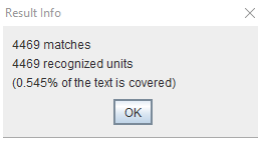
where we will find and select *sentenceWithoutDot.grf*. The search with this graph has more than 200 matches; therefore, we must modify the *Search limitation* option by checking *Index all occurrences in text*. When we click the *SEARCH* button, we obtain 1 316 sentence delimiters preceded by a question mark or an exclamation mark.





If we select the *Regular expression* option and type *?+!* in the search field, we obtain 4 469 question marks or exclamation marks in the text, difference due to the presence of direct speech.





This may seem confusing, but if we look at the following sequence: *Dear Mama! what is…* (among the first matched sequence in the concordance), we can see that the *sentence delimiter* is not inserted after each question or exclamation mark by the applied *Sentence.grf.* The users may modify the sentence graph supplied by the Unitex distribution or write their own sentence graph.

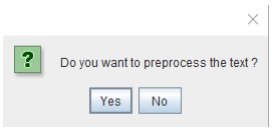
3.3.2 *The* Replace.grf *graph*

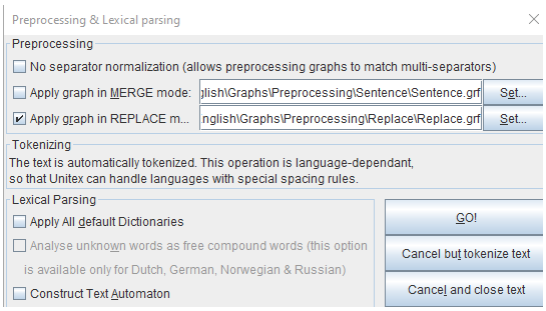
The *Replace.grf* graph modifies the text by replacing the contracted verb forms with the corresponding full forms, for instance *I'm* is replaced by *I am*.

We will open the file

MyUnitex\English\Corpus\UnitexGettingStarted\UnitexGettingStarted\DombeyAndSon\DombeyAndSon.txt

by using the *Text/Open…* menu and then answer *Yes* to the question *Do you want to preprocess the text?* If we check only the box next to *Apply graph in REPLACE mode*

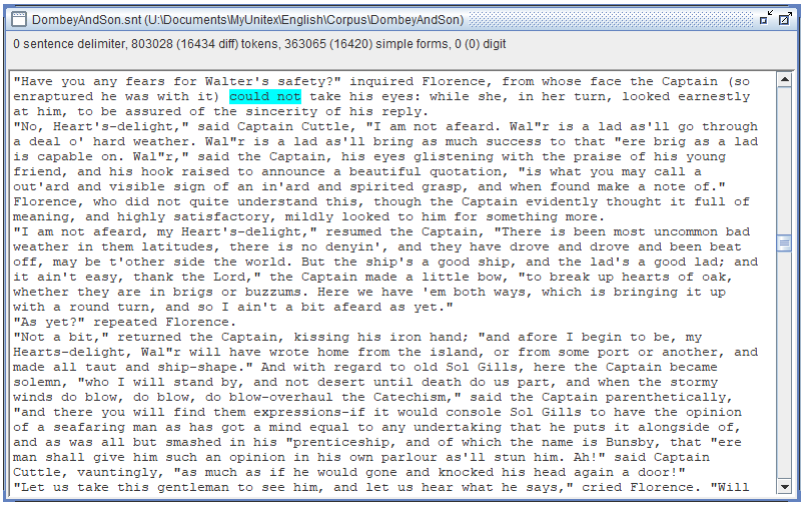




the displayed information for the file:

MyUnitex\English\Corpus\UnitexGettingStarted\UnitexGettingStarted\DombeyAndSon\DombeyAndSon.snt

does not change from the text without preprocessing (Section 3.1.2.2, page 36), because the number of tokens remains the same; however, frequencies of some tokens change. We can see, at the thirteenth paragraph, the first transformation: *couldn't* is replaced by *could not*.



The both sequences have the same number of tokens, three; that is the reason that the number of tokens in the whole novel has not changed. On the other hand, *could* already existed in the text, so the number of different tokens decreases (16 448 to 16 434).

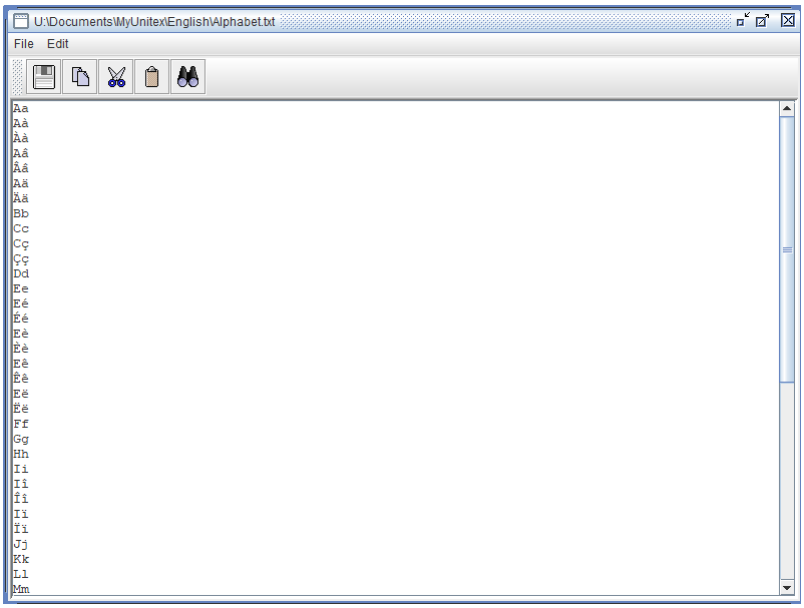
3.3.3 The two alphabet files

3.3.3.1 The Alphabet.txt file

Recall (see Section 1.2.3, page 13) that a token is either a string of letters or any other single character. What is considered a letter is defined by the *Alphabet.txt* file:

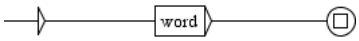
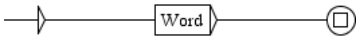
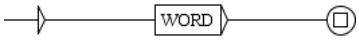
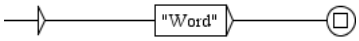
MyUnitex\English\Alphabet.txt

Since there are two cases in English, all letters are presented in pairs: uppercase and lowercase. We can open this file using the *File Edition/Open/Other Files* menu.



The pairs mean that a word in a box written in lowercase recognizes the same word with uppercase letters in a text. The reverse is not true; that is, if we place uppercase letters in a

box, only words with uppercase at that position are recognized. It is also possible to enforce case-sensitive matching using quotation marks.[52]

This box	recognizes
	*word*, *Word*, *WORD*, but also *wOrd*, etc.
	*Word*, *WORD*, *WOrd*…, but not *word*, *wORD*, etc.
	*WORD* and nothing else
	*Word* and nothing else

The same is true when a dictionary is consulted; the entry:

word,.N:s

matches words *word*, *Word*, *WORD*, but also *wOrd*, etc., while the imaginary entry

WORD,.N:s

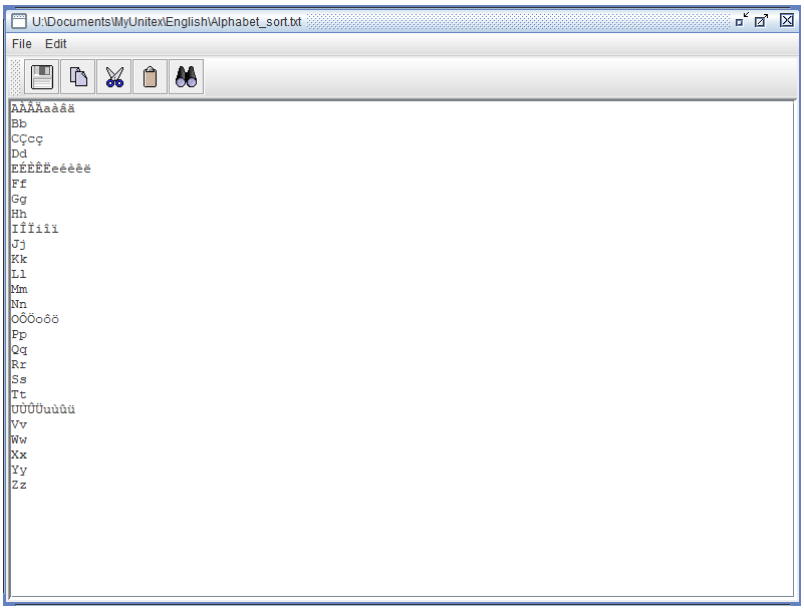
would match only the word *WORD*.

3.3.3.2 The Alphabet_sort.txt file

The *Alphabet_sort.txt* file:

MyUnitex\English\Alphabet_sort.txt

defines the sort order of the letters of the alphabet. It is used to sort concordances and dictionaries, when presented in alphabetical order. The alphabetical sort implemented in Unitex is very fast, even on very large files. We will open this file by using the *File Edition/Open/Other Files* menu.



[52] Or, as we have seen in Section 2.4.3, page 34, by using lexical masks.

In this file, letters are also represented in pairs, uppercase and lowercase. The sort order is defined in two levels. The first level is defined by lines, meaning that, for instance, each *A* is preceding each *B*, whether it is written in uppercase or not, whether it is accented or not. The second level is defined by each line. Consider the line:

EÉÈÊËeéèêë

It imposes that an *E* without an accent precedes an *É* with an acute accent, and also that an *É* with an acute accent precedes an *È* with a grave accent, and so on. For instance, in French, *PÊCHER* or *pêcher* (*to fish*) and *PÉCHER* or *pécher* (*to sin*) are ranked as follows:

PÉCHER, PÊCHER, pécher, pêcher

However, the whole group of letters has its own order in relation to other groups of letters. Thus, the words *ÉTÉ* or *été* (*summer*) and *ÈRE* or *ère* (*era*) and *EUROPE* or *Europe* are ranked:

ÈRE, ère, ÉTÉ, été, EUROPE, Europe

because the *Rr* group precedes the *Tt* group and the *Tt* group precedes the *Uu* group.

Chapter 3: corpus annotation

Before starting this chapter

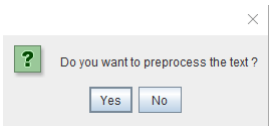
Go to the folder: *MyUnitex\English\Graphs\UnitexGettingStarted*
Create three new folders named *Will*, *Numbers* and *RomanNumerals*.

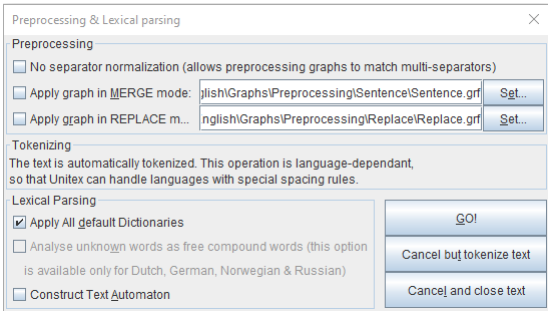
1 First example: compound verbs using the verb *will*

We will start this chapter by opening the text *DombeyAndSon.txt* (*Text/Open* menu) in the folder:

MyUnitex\English\Corpus\UnitexGettingStarted\UnitexGettingStarted\DombeyAndSon

and by answering *Yes* to the question *Do you want to preprocess the text*. A window will open in which we will remove the first two checkmarks, leaving just the last one *Apply All default Dictionaries*.





1.1 First step: the simplest graph

Let us create a first graph that we will save, like all graphs in this chapter, in the folder:

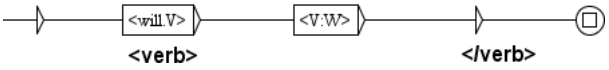
MyUnitex\English\Graphs\UnitexGettingStarted\Will

We will name it *willV.grf*. This graph will recognize all forms of the verb *will* (forms *will* and *would*) followed by a verb in the infinitive (the future tense) and insert the XML tags *<verb>…</verb>* for the recognized forms. To insert an annotation into a box, add a trailing slash and type the desired annotation. This annotation will be placed just before the text recognized by the box. To place an annotation after the box, you must therefore add an empty box after it (with the annotation).

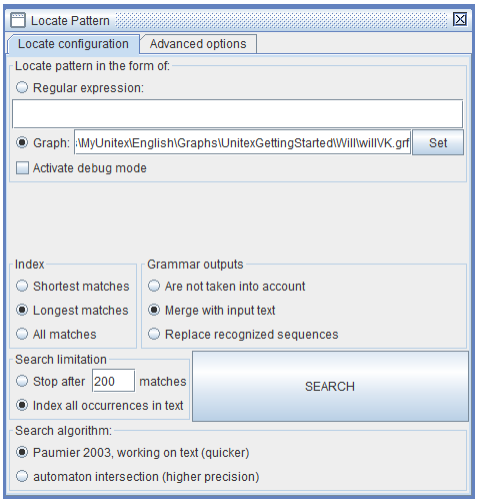
Generally, the lexical mask *<will>* represents all forms of the lexeme *will*. However, there are two lexemes *<will>*, a verb (*will*) and a noun (*a will*, *wills*). In order to avoid the recognition of nominal forms of *will*, we have to precise its part-of-speech with an appropriate code *<will.V>*.[53]

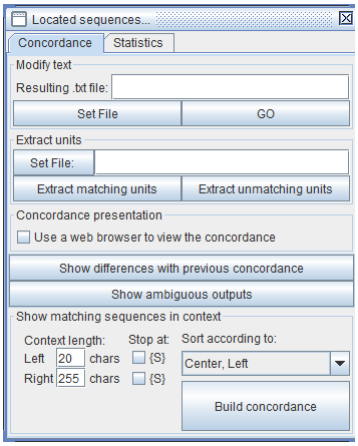
[53] Pay attention to the fact that information inside a lexical mask is case sensitive.

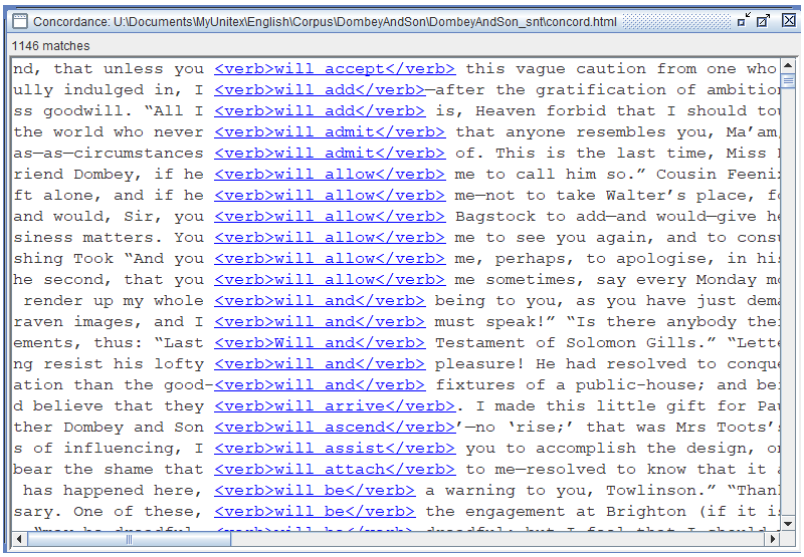
In the English dictionary distributed with *Unitex*, the code for infinitive forms of verbs is *W*. The lexical mask that recognizes an infinitive form of any verb is *<V:W>*.



Start the search (*Text/Locate patterns*) using the *willV.grf* graph (*Set* button in the *Graph* line) with options *Merge with input text* and *Index all occurrences in text*. Click the *Search* button and then click the *OK* button. We get 1 146 matches. To make the concordance easier to read, we are going to slightly modify the display parameters by choosing to display 20 characters on the left and 255 characters on the right of the matched sequences. Finally, click on the *Build concordance* button.





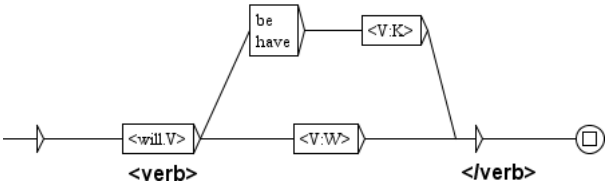


You will see that at the beginning of the concordance list there are some false recognitions. Namely, in a text segment *could I do more than render up my whole will and being to you*, *will* is a form of a noun and *and* is a conjunction. This is because, in the English dictionary distributed with *Unitex* a form *will* has two interpretations: a noun and a verb, while *and* has

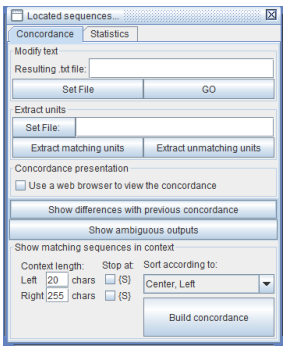
also two interpretations: a conjunction and a verb.[54] We will see in the section devoted to dictionaries how some cases of ambiguities can be avoided.

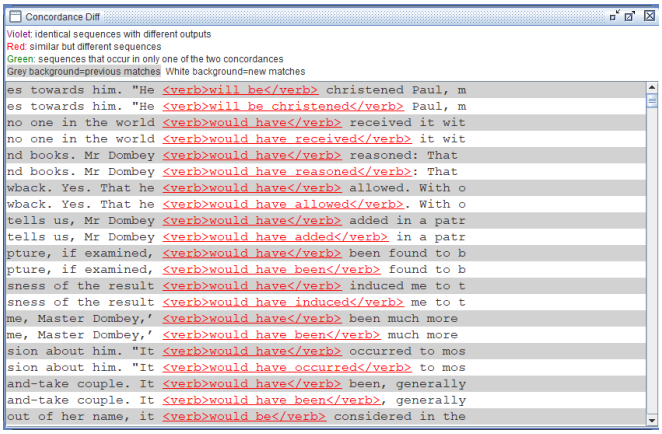
1.2 Second step: future passive and future perfect tense

When looking at the produced concordance we observe that in several cases *will be* and *will have* were recognized as a form of future tense, but the recognition was not complete. Now we will modify our graph so that it recognizes also passive tenses and forms of the future perfect tense. In the English dictionary distributed with *Unitex,* past participles are encoded with *K* and a lexical mask for retrieval of past participle word forms is *<V:K>*. In order to extend our graph, we will add a new path that exits from the *<will:V>* box. In this path, a form of a past participle must be preceded by *be* or *have.* Note that in this case, we use strings and not lexical masks, because these are exact forms that have to be recognized. We could have also used *<be.V:W>* and *<have.V:W>* but these masks correspond to exactly one form each.



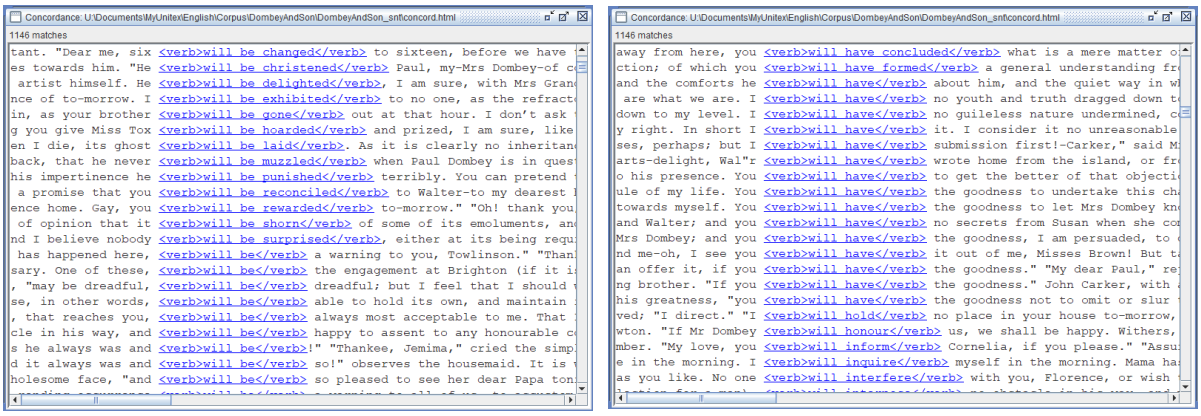
Open the *Text/Locate Pattern…* menu and click the *Search* button. We will obtain again 1 146 matches. Instead of producing concordances, we will click the *Show differences with previous concordances* button in the *Locate sequences* dialog box. We will obtain the parallel list of matches obtained by the previous graph and matches obtained by the modified graph.



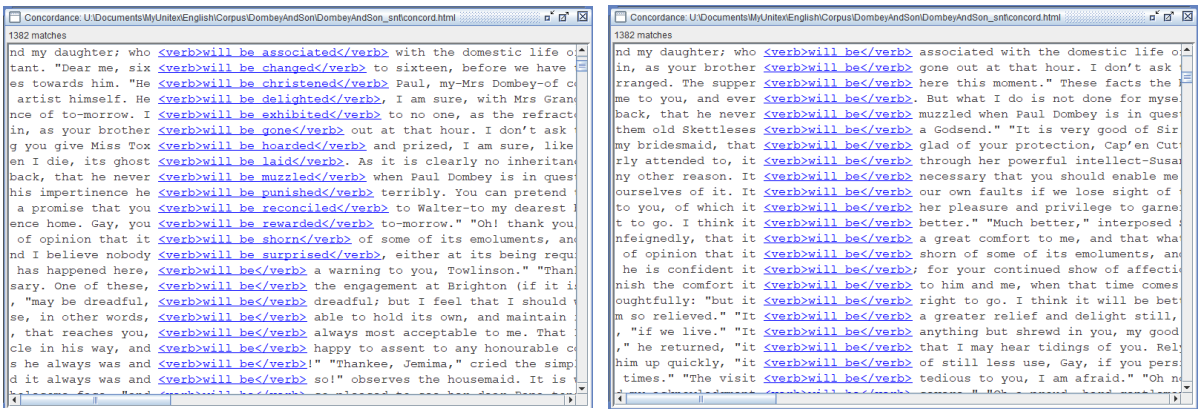


Open the *Text/Located sequences…* menu and click the *Build concordance* button. You will obtain the new concordance list.

[54] An intransitive verb with dialectal use, meaning: to breathe; whisper; devise; imagine (source: Wiktionary).

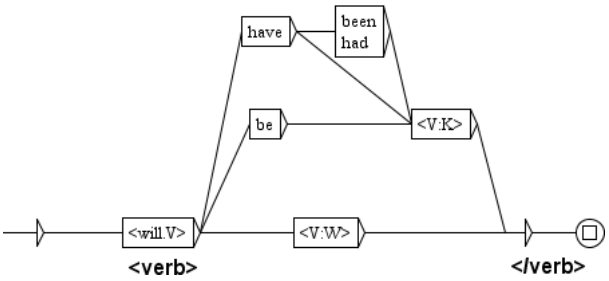


You should note that, in the *Locate pattern* dialog box, we used the default option *Index longest matches*, which means that shortest matches at the same position will not be indexed or listed in concordances. In order to understand what it means, open the *Text/Locate pattern…* menu, change the default option to *Index all matches* and click the *Build concordance* button. Now, you will obtain 1 382 matches and among them are both shorter and longer matches, for instance *will be associated* and *will be.* You may wonder why two concordance lines containing these two matches are far apart in the concordance list. When building concordances, we used the default option for sorting concordance lines (*Located sequences/Sort according to*) which is *Center, Left*. This option means that concordances are first sorted according to matches (from left to right) and then according to the left context (from right to left). In this case, the occurrence *<verb>will be associated</verb>* will be listed before *<verb>will be</verb> associated*.



1.3 Third step: recognizing the future perfect passive

We will modify our graph to recognize more compound verb forms with the lexeme *<will.V>*. Now we must split the *be+have* box because the forms *been* and *had* can follow *have* but not be.

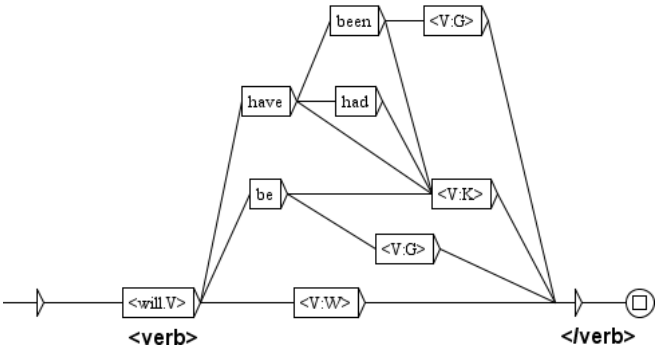


Open the *Text/Locate Pattern…* menu, return to the original indexing option: *Longest matches* and click the *Search* button. By clicking the *Build concordances* button in the *Locate sequences* dialog box, you will obtain the new concordance. This list again contains 1 146 lines. Note that the added forms *been* and *had* are also past participles of verbs *to be* and *to have* respectively, so now some longer (complete) compound forms are recognized instead of shorter ones: for instance, *would have been* (before) is replaced by *would have been added* (now).[55]

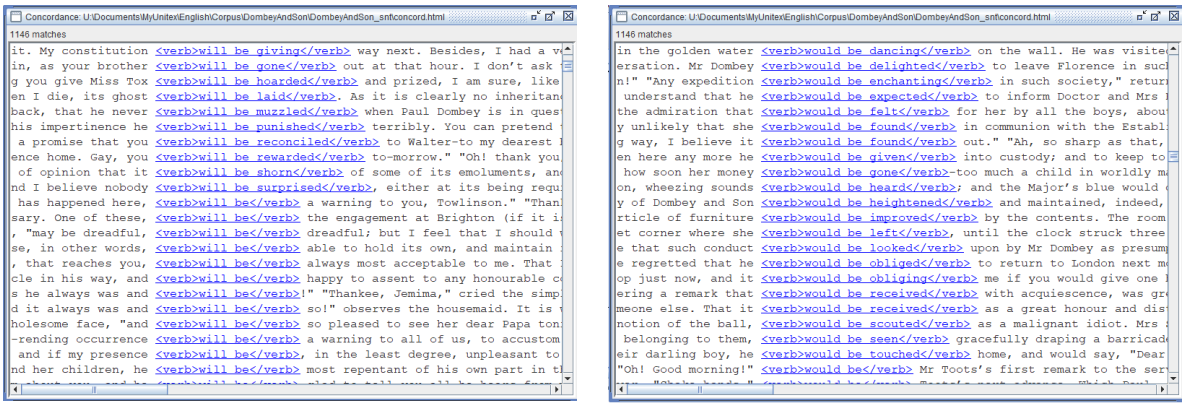
1.4 Fourth step: recognizing continuous forms

In order to recognize continuous forms: future continuous and future perfect continuous, we will modify our graph again. Continuous forms are built using the present participle for which we use the lexical mask *<V:G>*. This mask will be used in two new boxes, once for the future continuous tense and once for the future perfect continuous tense. We must split the *been+had* box because the present participle can follow *will have been,* but not *will have had*.

[55] You can analyze all differences obtained by this new graph if you use the *Show differences with previous concordances* option in the *Located sequences* dialog box.



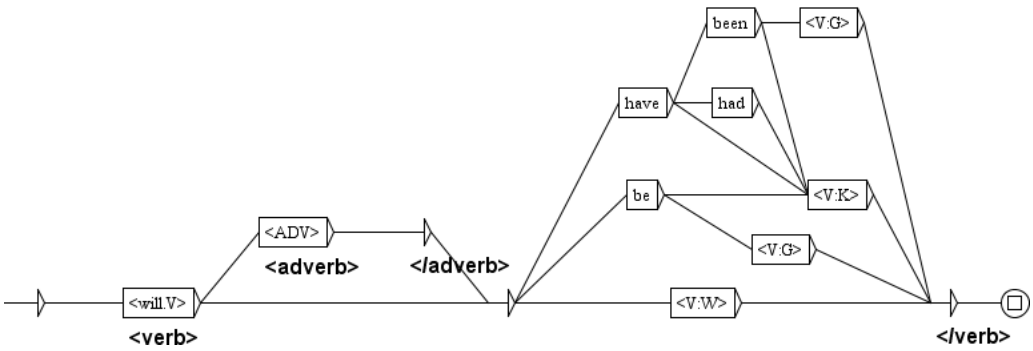
Now the concordance highlights four new sequences:[56] *will be giving*, *would be dancing*, *would be enchanting* and *would be obliging*.



1.5 Fifth step: insertion of an adverb after *will* or *would*

1.5.1 A single adverb

Now we will take into account that an adverb can be inserted after the verb *<will.V>*. Since the code used for adverbs in the English dictionary distributed with Unitex is *ADV*, we will use for them the lexical mask *<ADV>*. The tags around them in a text will be *<adverb>...</adverb>*.

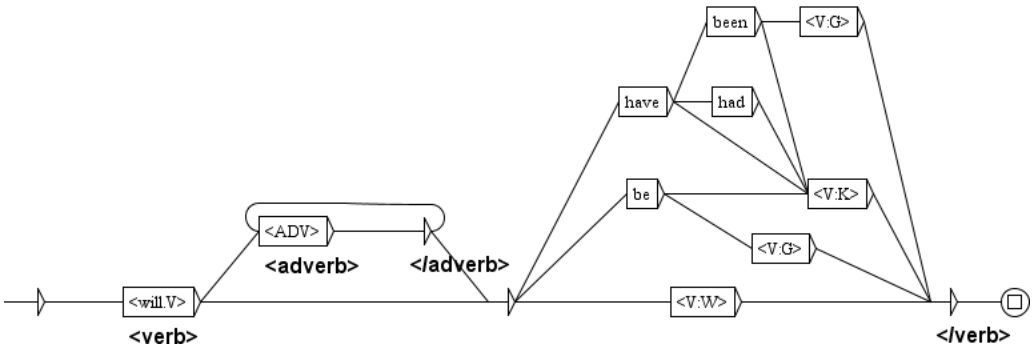


Save the modified graph and use it in *Locate pattern*. You will obtain now 1 330 concordance lines. This time, they contain new matches, because occurrences with an adverb after *<will.V>* could not be retrieved with the previous version of the graph. Some of the occurrences can be seen in the following figure.

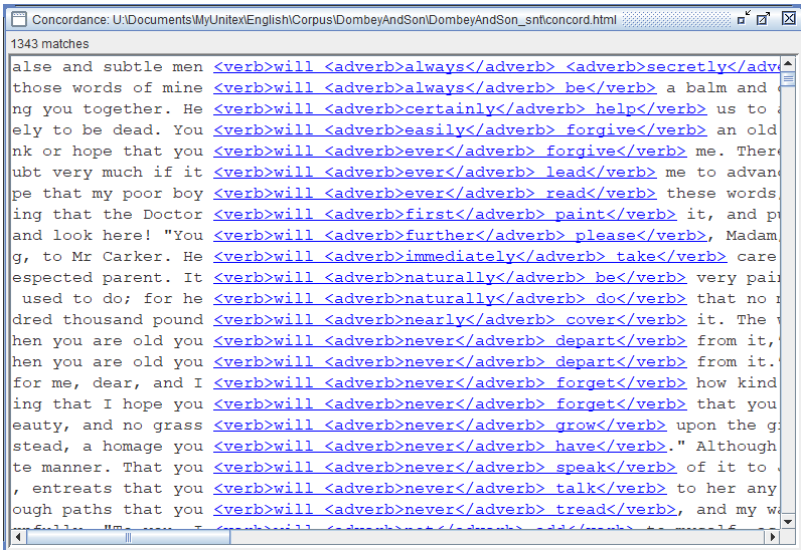
[56] The number of matches remains the same because the previous graph recognized *will be*.

1.5.2 Several adverbs

However, it sometimes happens that an adverb is followed by another adverb or even more. We will add a loop between the two boxes concerned.



The new concordance shows 13 new matches. You can also use the *Show differences with previous concordance* option in the *Located sequences* menu to spot them.



1.6 Sixth step: insertion of an adverb after *be*

The position after the verb *<will.V>* is not the only one where an adverb can occur. It can also be inserted between the verb *to be* and the present or past participle.

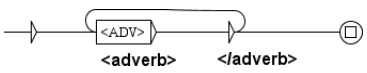
This should be added as optional after the *been* and *be* boxes in our previous graph. Therefore, we will modify our graph once again. However, rather than duplicating the boxes and loop of the previous section, we will create a subgraph to recognize adverbs.

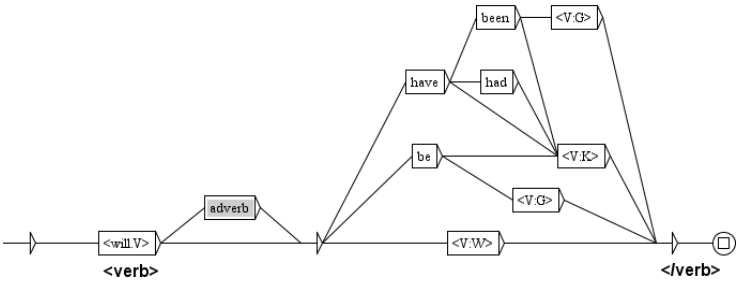
1.6.1 A graph that recognizes and annotates adverbs

We observe that adverbs can appear at two positions in a verb group, so the same boxes for recognition and annotation would appear twice. In such situations, it is good to use subgraphs. First, the same problem of recognizing and annotating adverbs can appear in some other graph and then the subgraph can be reused.[57] Second, the graph that recognizes verb groups with *will/would* becomes less complex and easier to read and to develop further.

The easiest way to create a subgraph named *adverb.grf* from the existing graph is to use the *Export as new graph* option (select boxes that will belong to a subgraph and use the right mouse button). Here we select the *<ADV>/<adverb>* and *<E>/</adverb>* boxes, then we right-click to choose the *Export as new graph* option. We call this graph *adverb.grf*.

The boxes of the main graph, which recognize and annotate the adverbs, must then be replaced by a call to the subgraph (name preceded by a colon, here *:adverb*). In the main graph, boxes that call a subgraph have a gray background.



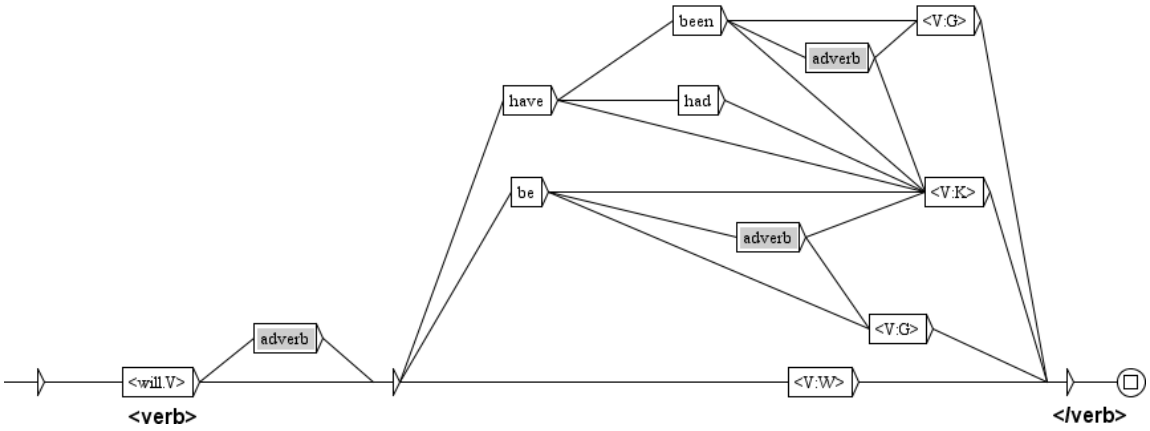


The concordance obviously remains the same.

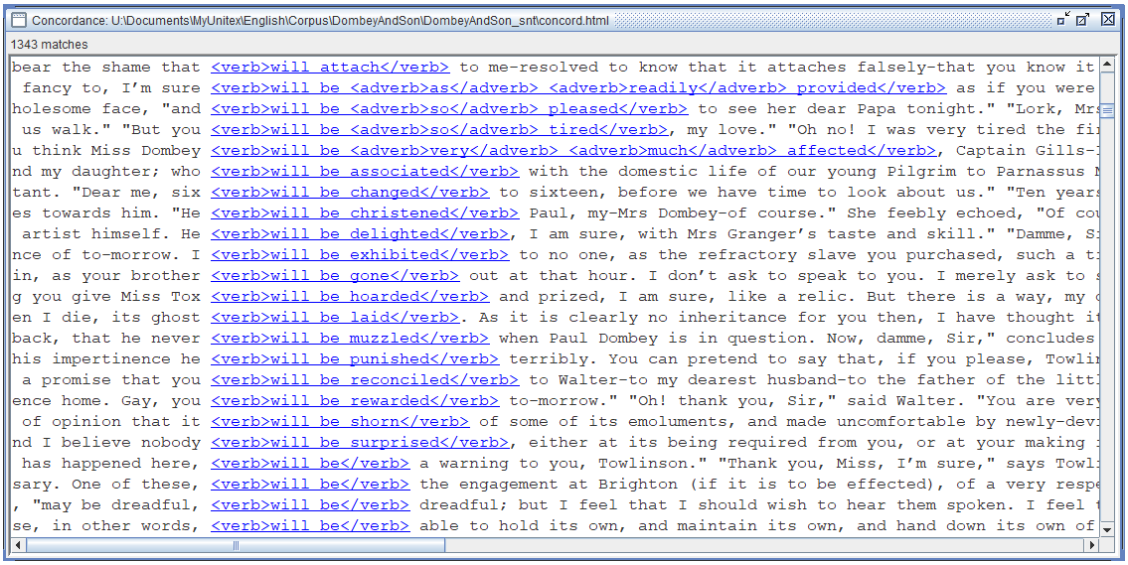
1.6.2 The main graph

Let us modify our graph again to recognize occurrences of adverbs after the verb *to be* (in two places).

[57] See Chapter 4, Section 4.3, page 95 (only for confident users).

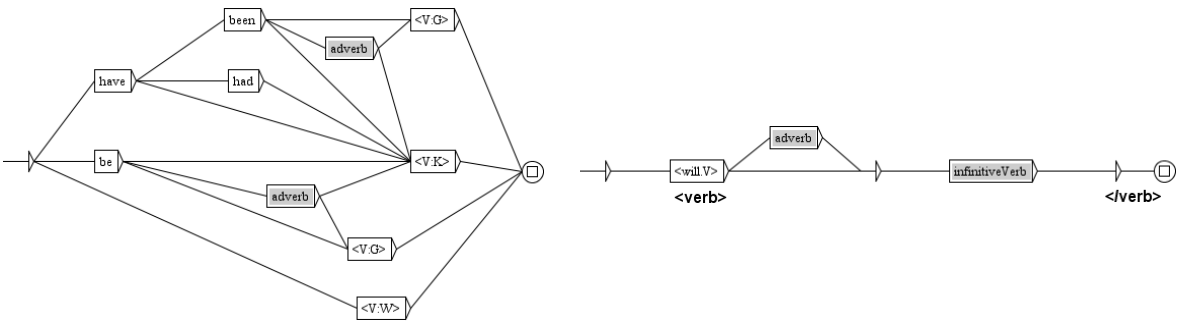


We obtain the same number of concordance lines, but longer sequences are recognized and highlighted.



1.6.3 A new subgraph

Before continuing to improve our graph, we will simplify it by creating a new subgraph. Since the next steps will enhance the beginning of the graph without changing anything in its second part, we will create a new subgraph from it. In order to make the main graph easy to understand we will leave the insertion of the end-tag *</verb>* in the main graph. We will give the name *infinitiveVerb.grf* to the new subgraph.



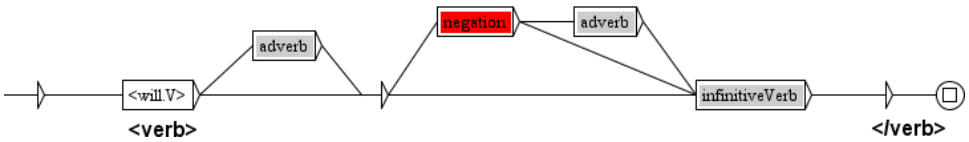
If you apply this modified graph to the text, you will again obtain the same concordances, because we have only reorganized the graph.

1.7 Seventh step: negative adverbs

1.7.1 The main graph

In the concordance, you can see two special adverbs, *never* and *not*, which mark negative forms.

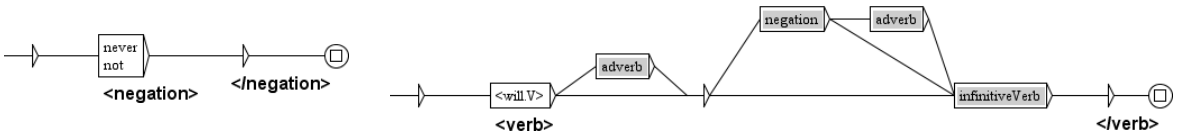
We will now treat them separately, as a negation. They usually follow another adverb, although sometimes they can precede it. We will create a subgraph for negation, but before doing that we will put a call to a future subgraph named *negation.grf* at the right place in the main graph. This graph does not exist yet, so this box has a red background.



1.7.2 The negation subgraph

We can easily create this new subgraph from the main graph: we need to click on the *:negation* box, click the right mouse button and select *Open subgraph* from the menu displayed, then we need to click the *Yes* button to create the new graph.

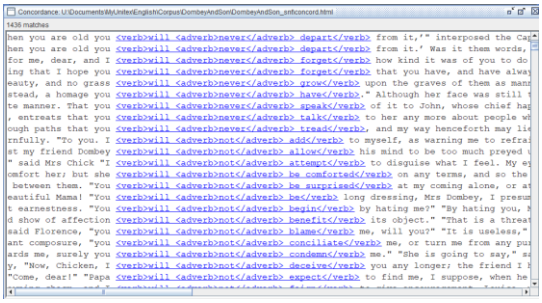
After creating and saving the negation graph, the background of the *:negation* box in the main graph becomes gray.

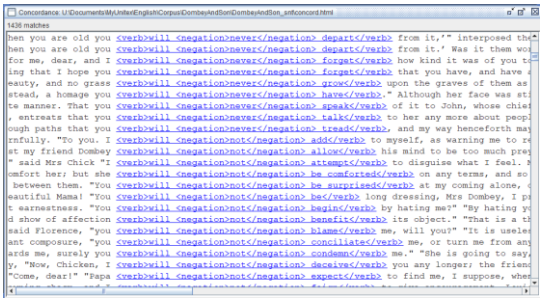


> Notes
>
> A call to a subgraph has a red background if:
> 1. The subgraph exists, but the main graph has not yet been saved.
> 2. The subgraph exists but it is not in the same folder as the main graph.
> 3. The subgraph does not exist, it will be created later.
> 4. The subgraph does not exist because the call contains a spelling error.
>
> If not, the call to a subgraph has a grey background.

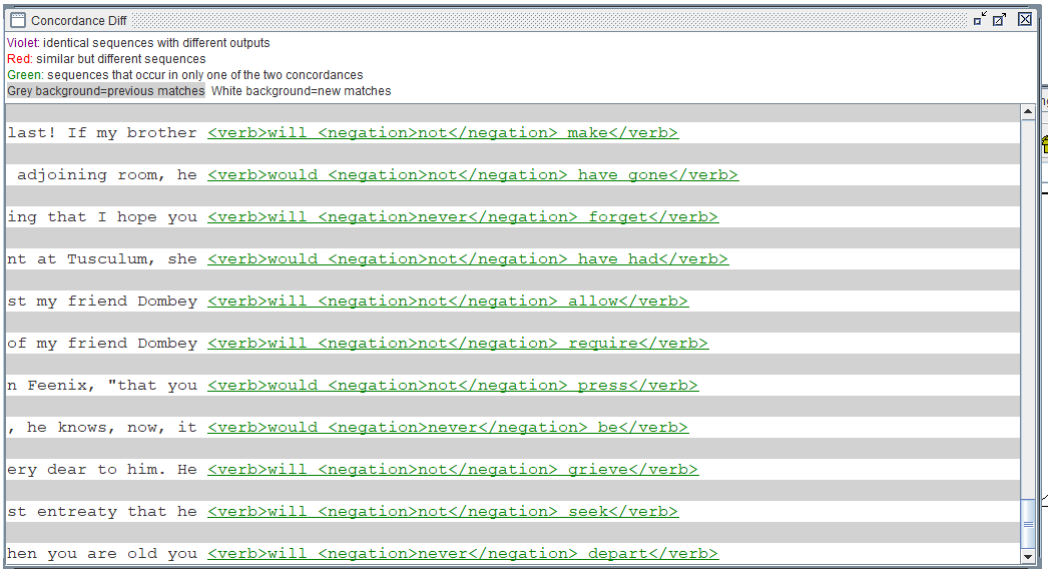
1.7.3 Negative right context

If we look closely at the concordance obtained, we may be surprised to see 93 more lines, a total of 1 436 lines.[58] These are lines in which *never* and *not* are annotated as *negation*; they appear in addition to the 93 lines annotated as *adverb*. The *Longest matches* option does not allow for choosing between the two annotations; they are therefore both displayed.

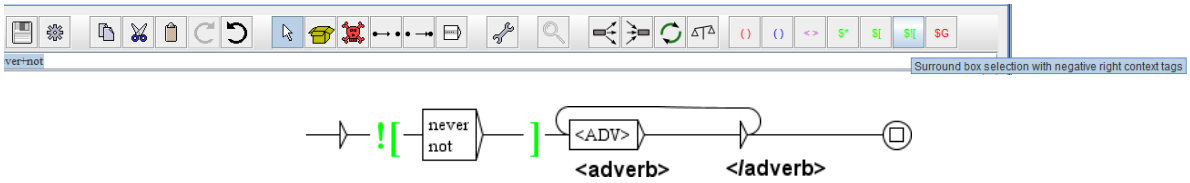




If we look at the list of differences between the two concordances (see Section 1.2, page 47), we will see that the new concordance includes all forms of adverbs *never* and *not* labeled also as negation.



This is normal, because *never* and *not* are both adverbs and negations, but that is not what we want. To force the negation tag, we will use a *negative right context* in the *adverb.grf* subgraph, which is not to be confused with the *right context* seen in Chapter 2, Section 2.4.2, page 32.

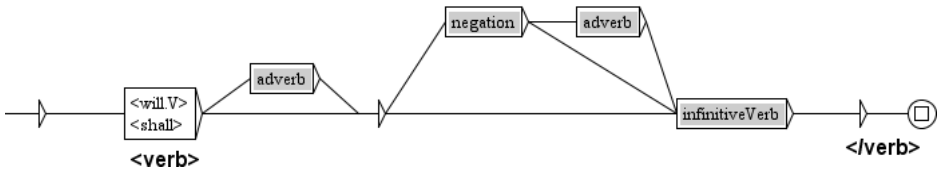


[58] There is the same number of matches as before, 1 343, but more concordance lines, 1 436.

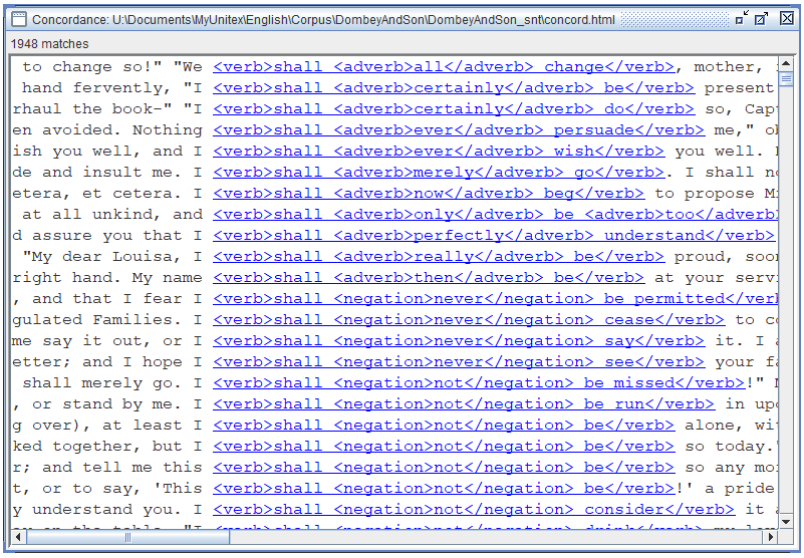
Now, *never* and *not* are always annotated as negation. There are again 1 343 lines in the produced concordance.

1.8 Eighth step: adding the verb *shall*

The verb *shall* follows the same rules as the verb *will*, making it easy to add it in the main graph. Since this verb is not ambiguous with a noun or some other category, as *will* is, it is not necessary to add its category in a lexical mask, so we will use *<shall>*.[59]



The new concordance contains 1 948 lines.



[59] Keep in mind that the lexical mask *<shall.V>* would not be incorrect; these two masks produce the same results.

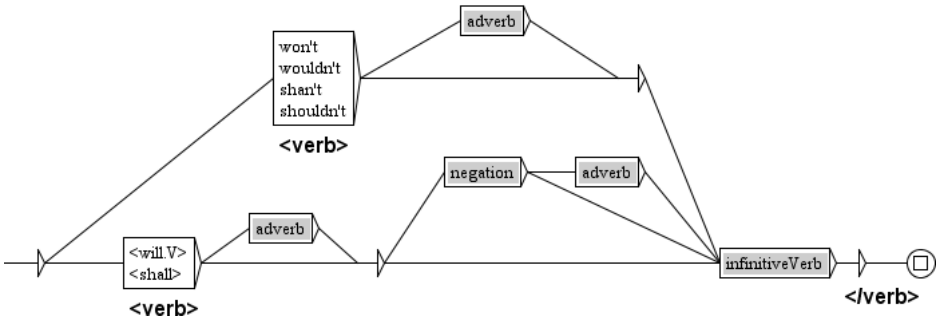
1.9 Continuation of this graph (only for confident users)

Before starting this section

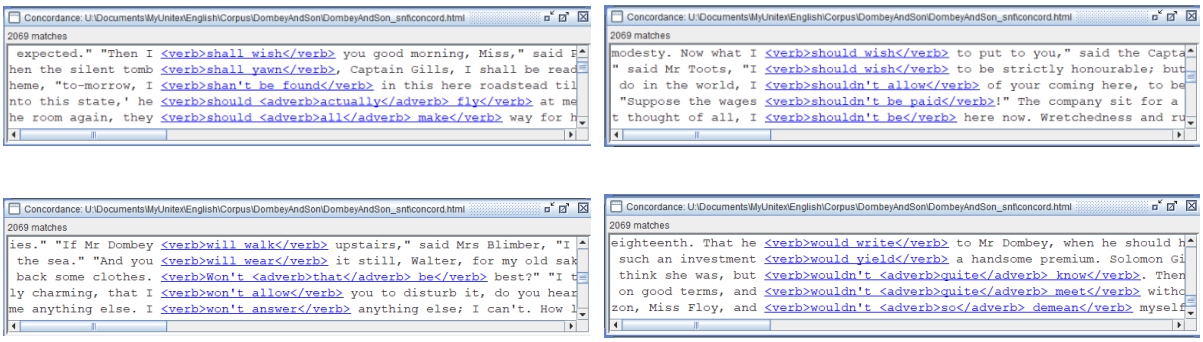
In order to retrieve all results you should use the *Norm.txt* file introduced in Chapter 2, Section 3.1.2, page 35. If you have not used it when you preprocessed the text, download it, then close and reopen Unitex.
Open the original *DombeyAndSon.txt* file, process it by applying the default dictionaries (do not use *Sentence* or *Replace* graphs).

1.9.1 Addition of contracted forms

In the *willV* graph, we have omitted the recognition of contracted forms *won't*, *wouldn't*, *shan't* and *shouldn't*. We need a separate path for them because, in the case of contractions with *not,* additional negations are not allowed.

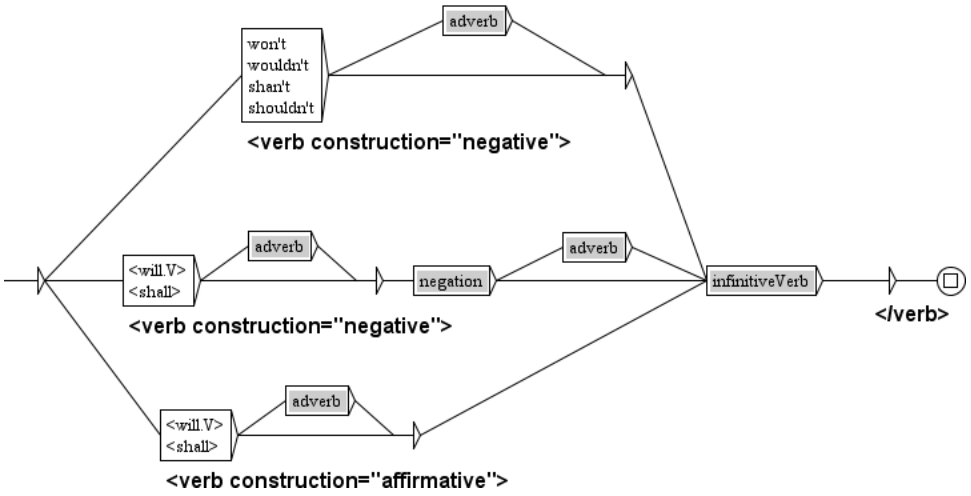


If you now apply the new *willV* graph to the text, you will get 2 069 lines, including 121 contracted forms, and among them all four possibilities.

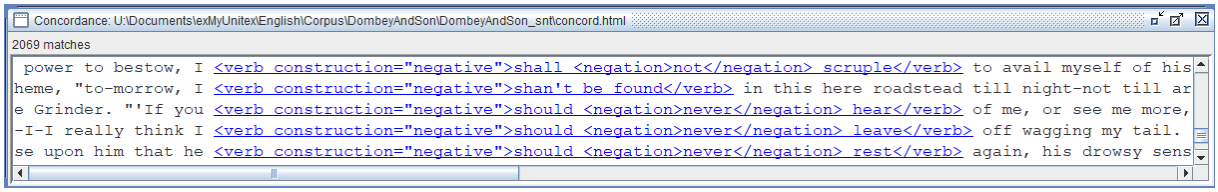


1.9.2 Affirmative and negative constructions

We can now distinguish between affirmative and negative forms. We will add the attribute *construction* with possible values *affirmative/negative* to the XML start tag *<verb>*.

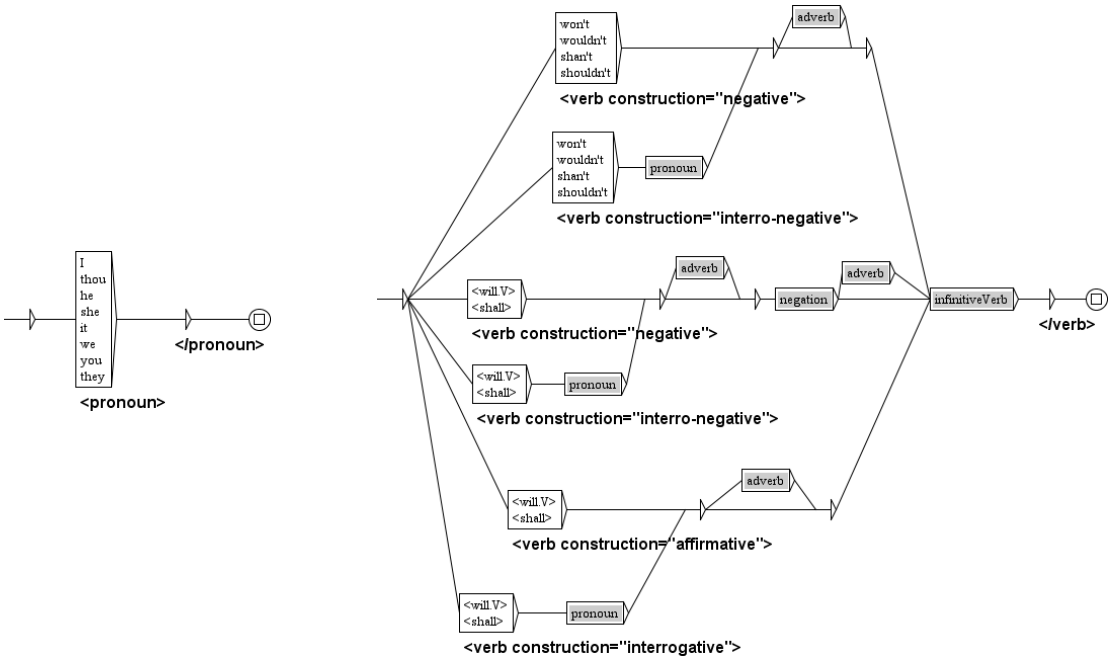


When you apply this graph to the text, you will obtain the same number of concordance lines, 2 069. The output is different since recognized verb groups are now annotated as affirmative or negative.



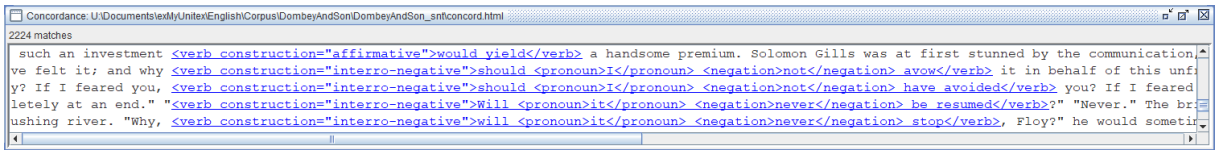
1.9.3 Interrogative constructions

In order to recognize interrogative constructions as well it is enough to create a subgraph for personal pronouns[60] and to duplicate several paths. The attribute type will now have two more possible values: *interro-negative* and *interrogative*.



[60] We add to the list of common pronouns the old form of the second person singular, *thou*.

If you apply this graph to the text you will obtain 2 224 concordance lines, some of which are represented below.



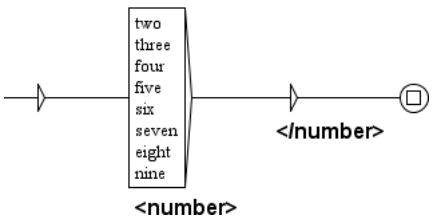
2 Second example: numbers written with words

Our second exercise will deal with the annotation of numerals written using only words (not digits). We will proceed step-by-step using subgraphs in each of them. We will save this graph in the *Numbers* folder:

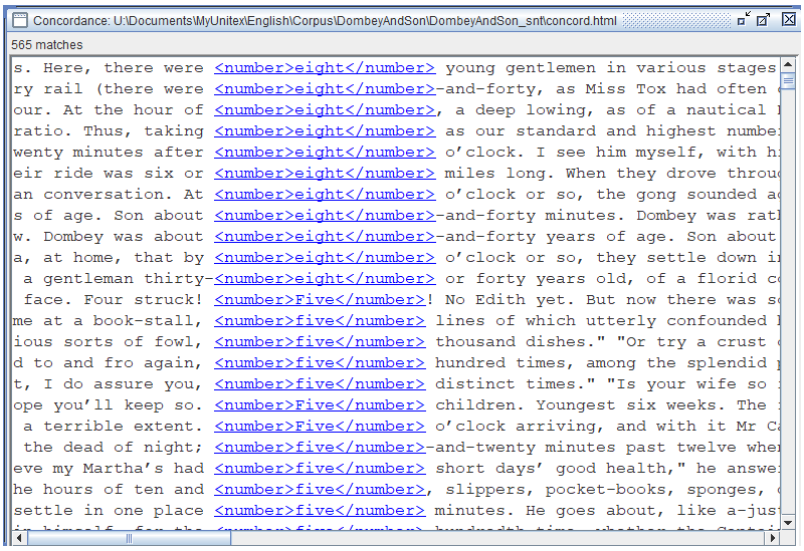
MyUnitex\English\Graphs\UnitexGettingStarted\Numbers

2.1 First step: from 2 to 9

We will start with numbers *two* to *nine*. We will skip the number *one* because it is ambiguous with the pronoun *one*. We will save this graph using the name *NB2-9.grf*.



When we apply this graph to our text *Dombey and son* in *Merge* mode and without the search limitation (*Index all occurrences in text*), we obtain 565 concordance lines, some of which are displayed in the screenshot.

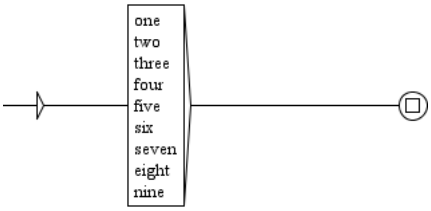


2.2 Second step: from 2 to 99

2.2.1 A subgraph without output

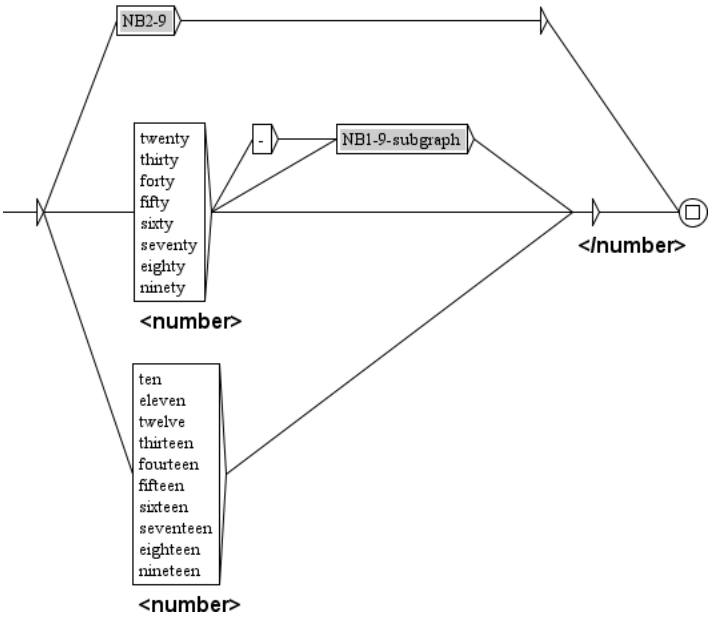
We would like to use the previous graph as a subgraph. It is possible to do it for numbers from 2 to 9; for numbers from 21 to 29 it is also possible to use the previous graph as a subgraph

but the result would contain some unwanted nesting: for instance, *twenty-two* would be annotated as *<number>twenty-<number>two</number></number>*. In order to avoid this, we will create, based on *NB2-9* graph, an *NB1-9-subgraph* graph that produces no output and contains the number one. Note that *one* that follows other numbers written in words is no longer ambiguous. This graph annotates *twenty-two* as *<number>twenty-two</number>*.

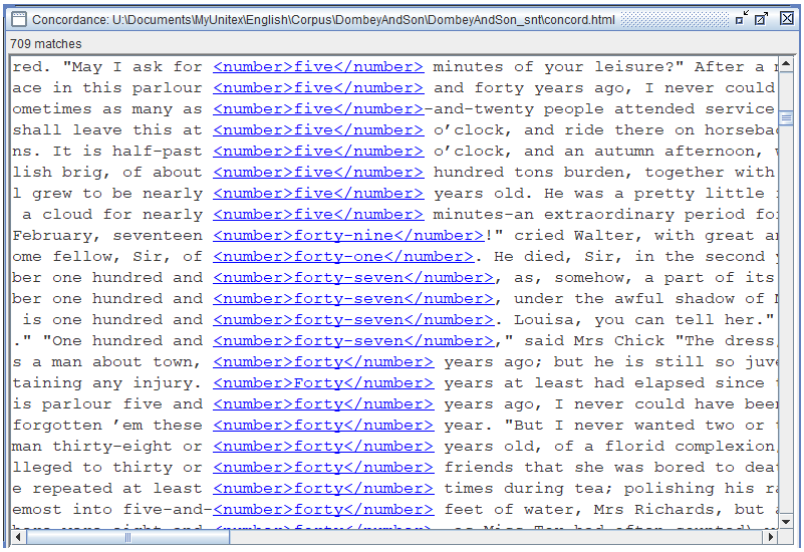


2.2.2 *The* NB2-99.grf *graph*

Now we can produce a graph that recognizes numerals from 2 to 99. It will have one path for the recognition of numbers *two* to *nine*: this path will consist of a box that calls the *NB2-9.grf* subgraph, which means that the annotation will also be performed by the subgraph. Another path will contain the box that recognizes numbers *ten* to *nineteen*. The box that recognizes these numbers will insert the start tag *<number>*. The third path will recognize numbers *twenty* to *ninety*, which can be followed by numbers *one* to *nine*. For this recognition, the subgraph *NB1-9-subgraph.grf* will be used. A hyphen can be used between numbers written in words. The box that will recognize numbers *twenty* to *ninety* will insert the start tag *<number>*, while the end tag *</number>* will be produced by an empty box common to the second and third paths (but not the first one). We will save this graph under the name *NB2-99.grf.*



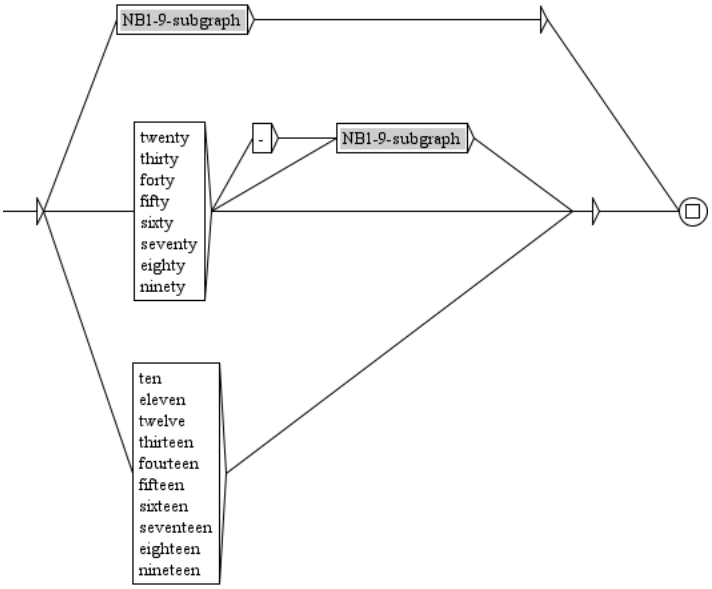
A few of the 709 concordance lines produced when applying this graph to the text are displayed in the screenshot.



2.3 Third step: from 2 to 999

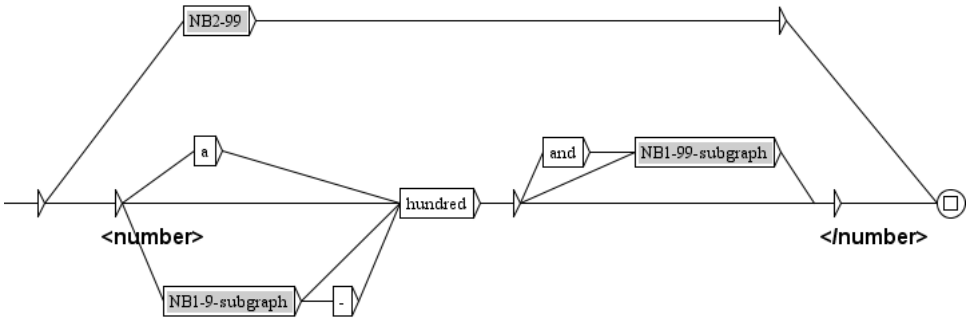
2.3.1 The NB1-99-subgraph.grf *graph*

For the same reason as before, we will produce a graph that recognizes numbers *two* to *ninety-nine* but does not insert any tag. Namely, we want to use it as a subgraph of the graph that will recognize numbers from *two* to *nine hundred ninety-nine*. We will base this graph on the already developed *NB2-99* graph and name it *NB2-99-subgraph.grf*.

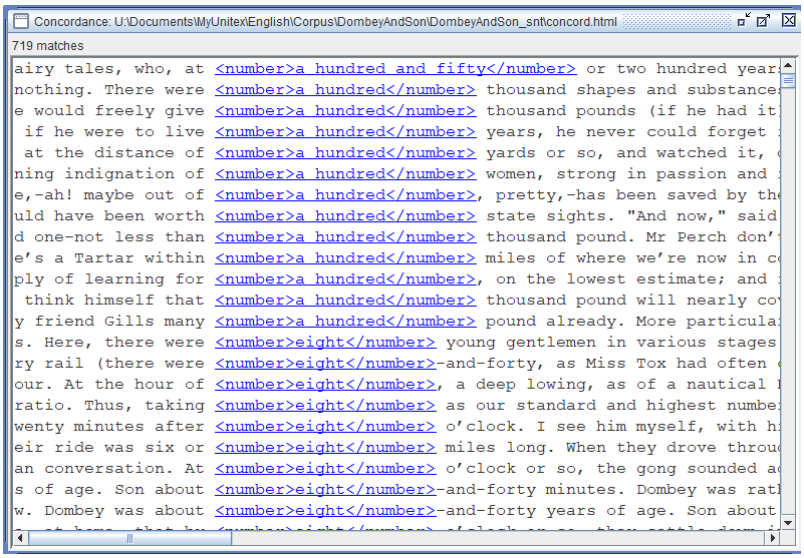


2.3.2 The NB2-999.grf *graph*

In this graph we will use as a subgraph *NB2-99.grf* to recognize and annotate all numbers in the interval from 2 to 99. For numbers in the interval from 100 to 199, the word *hundred* must be present and can be preceded by an indefinite article *a* or by a number *one*. For numbers in the interval 200 to 999, the word *hundred* has to be preceded by a number from 1 to 9 (possibly followed by a hyphen); for that, we will use the appropriate subgraph that does not insert tags. The previously produced *NB1-99-subgraph.grf* subgraph recognizes numbers from 1 to 99, without annotating them. Note that numbers from 1 to 99 that follow hundreds can be preceded with the conjunction *and*.



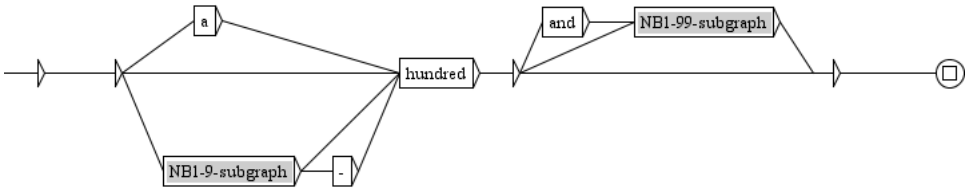
The application of the main graph to the text produces 719 concordance lines.



2.4 Fourth step: from 2 to 999 999

2.4.1 *The* NB100-999-subgraph.grf *graph*

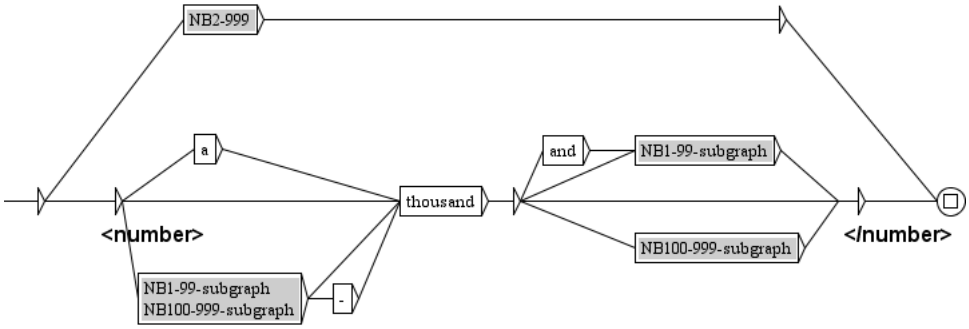
We will produce a graph named *NB100-999-subgraph.grf* that recognizes numbers from 100 to 999 without annotating them. It presents the modification of the previously developed *NB2-999.grf* graph, but the path recognizing numbers from 2 to 99 has been deleted. The reason why this path was deleted will become clear when observing the next graph, which recognizes numbers from 2 to 999 999.



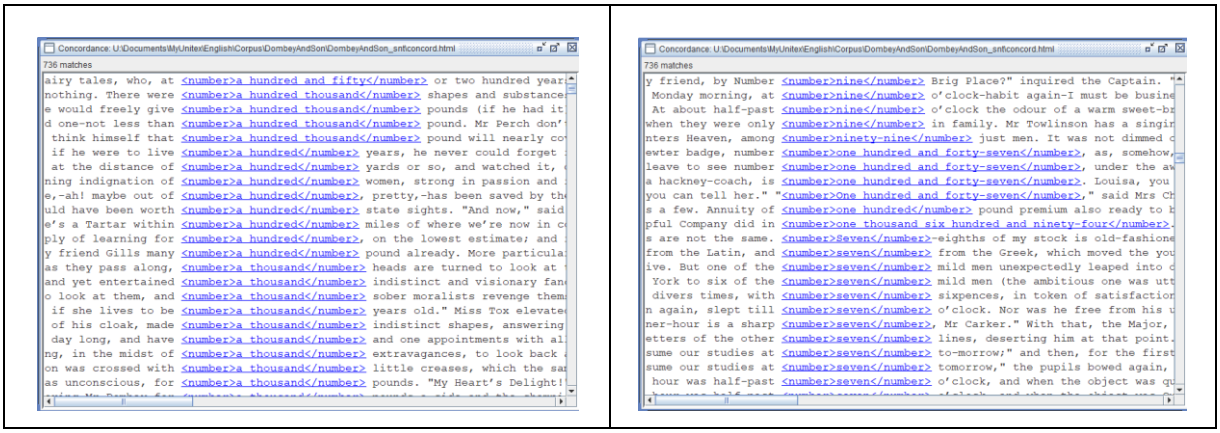
2.4.2 *The* NB2-999999.grf *graph*

This graph will look similar to the *NB2-999.grf* graph, except that here the word *thousand* has to be used, optionally preceded by *a* or a number between 1 to 99 (*NB1-99-subgraph.grf*, see Section 2.3.1, page 61) or between 100 to 999 (*NB100-999-subgraph.grf*). The word *thousand* can be followed by a number between 1 to 99 (subgraph *NB1-99-subgraph.grf*), which can be preceded by a conjunction *and*. The word thousand can also be followed by a number between 100 to 999 (*NB100-999-subgraph.grf*), but in that case, the conjunction *and* is not

used and that is the reason that we separated the cases of numbers between 1 to 99 and between 100 to 999.



If we apply this graph to the text, we will obtain 736 concordance lines for numbers from 2 to 999 999.



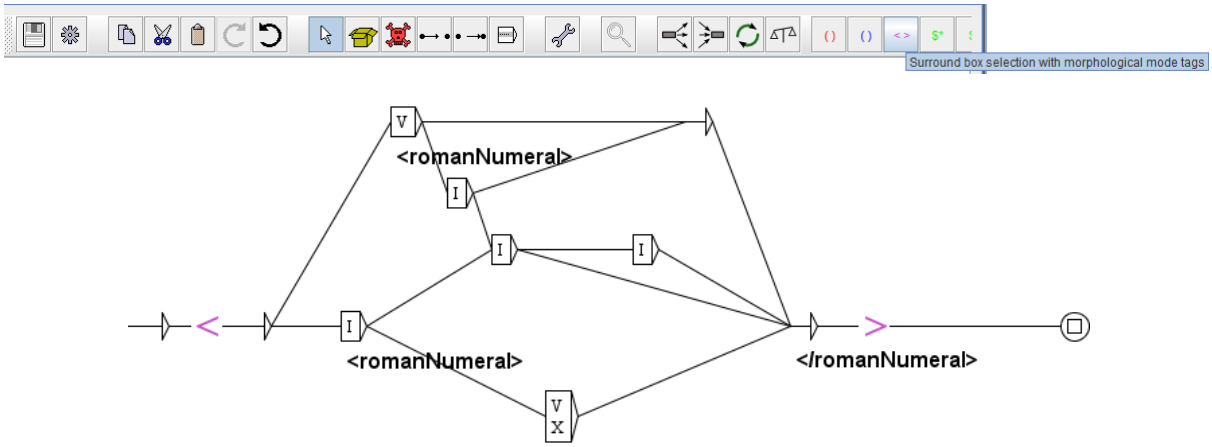
3 Third example: annotating Roman numerals

We will now continue with Roman numerals which we want to tag as *romanNumeral*. We want to describe them, as number written in words in the previous section, with graphs. However, we encounter a problem if we use graphs in a similar way to before. Namely, with three successive boxes containing the letter *I*, we can recognize three successive words *I I I* and not one word *III*. In order to solve this problem, we use the morphological mode that allows for matching a word, that is, a sequence of letters.

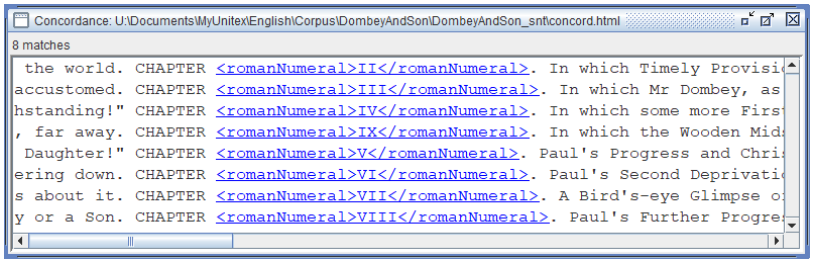
3.1 First step: from 2 to 9

We will start with the graph that recognizes Roman numerals with values 2 to 9. As before, we will skip the numeral *I* because it is ambiguous with the pronoun *I*. We will name this graph *RN2-9.grf*. We will build this graph bearing in mind the rules for constructing Roman numerals. Once we have done that we will select all boxes, except the start box and the end box. Then we will click the *Surround box selection with morphological mode tags* button (you will find this button at the right end of the toolbar represented as violet angular brackets). The graph is now surrounded with two boxes that contain codes *$<* and *$>*. Inside this *morphological part*, the content of boxes is read as concatenated.

In order to obtain a tidy graph, we will align the left and right parts of the graph. This can be done by opening the *FSGraph/Format/Alignment/Horizontal/Center* menu option. This menu can be accessed by the menu bar or by a click with the right mouse button.



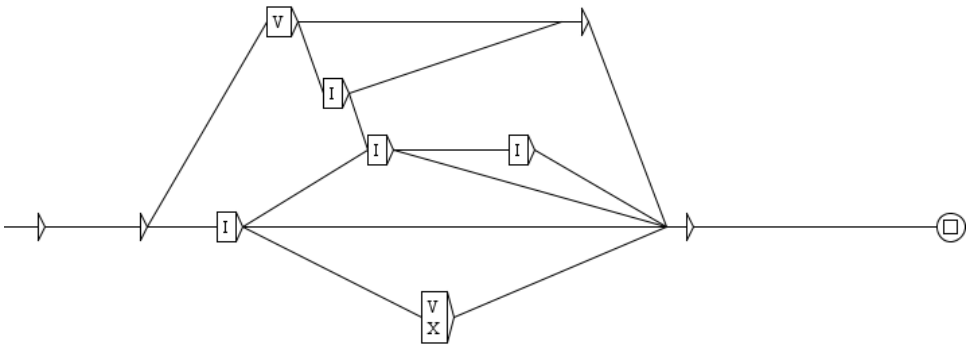
When applying this graph to the text we obtain a concordance with the numbers of the first eight chapters of our novel *Dombey and Son* (except the first one, which we omitted on purpose).



3.2 Second step: from 2 to 99

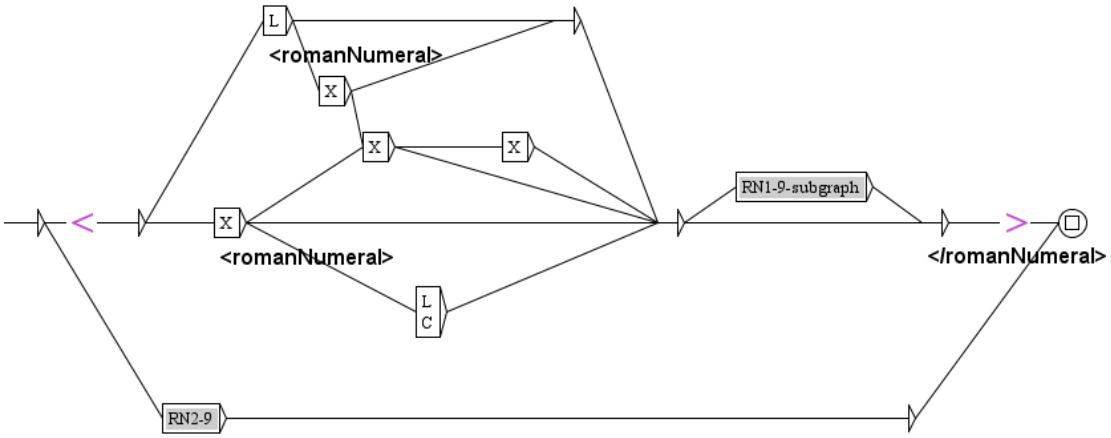
3.2.1 The subgraph from 1 to 9

As before, we will use the previously created *RN2-9.grf* graph to create a subgraph that does not insert tags. We will also cancel the morphological mode (by deleting the boxes that contain *$<* and *$>*), because it will have been introduced by the invoking graph, and nesting of morphological modes is not allowed. We will add a path that recognizes the roman numeral *I*, because in the context it will be used. We will name this graph *RN1-9-subgraph.grf*.

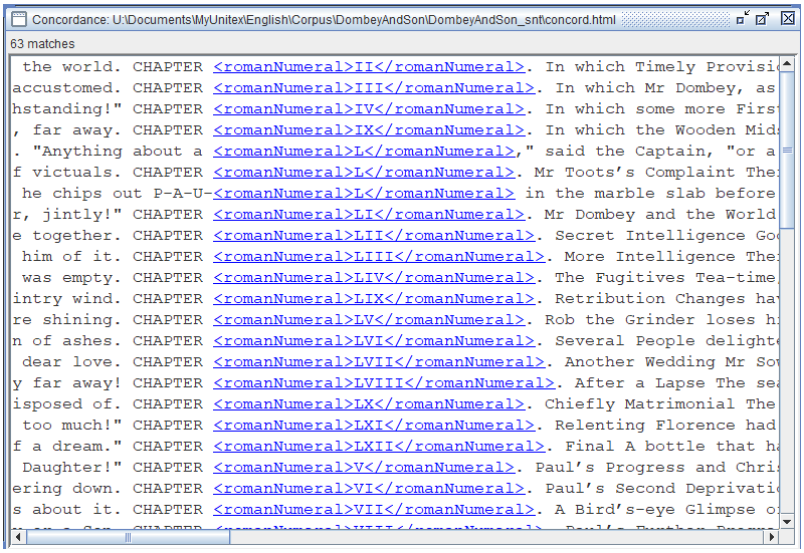


3.2.2 The graph from 2 to 99

Roman numerals are constructed very regularly. Therefore, in order to recognize numbers greater than 10 and up to 99, we can use the preceding *RN2-9.grf* graph, and replace in it *I* with *X*, *V* with *L* and *X* with *C*. Then we have to add a path for *X*, a call to the *RN2-9.grf* subgraph outside the morphological mode (this to recognize numerals from 2 to 9), and a call to the *RN1-9-subgraph.grf* (inside the morphological mode) for numerals from 1 to 9 when they are added to numerals greater or equal to ten. We will name this graph *RN2-99.grf.*



When we apply this graph to our text, we obtain 63 concordance lines. Note that this text (novel) has 64 chapters designated with Roman numerals. With the *RN2-99.grf* graph we recognize 63 of them (we do not recognize the first chapter "I").


Concordance: U:\Documents\MyUnitex\English\Corpus\DombeyAndSon\DombeyAndSon_snt\concord.html

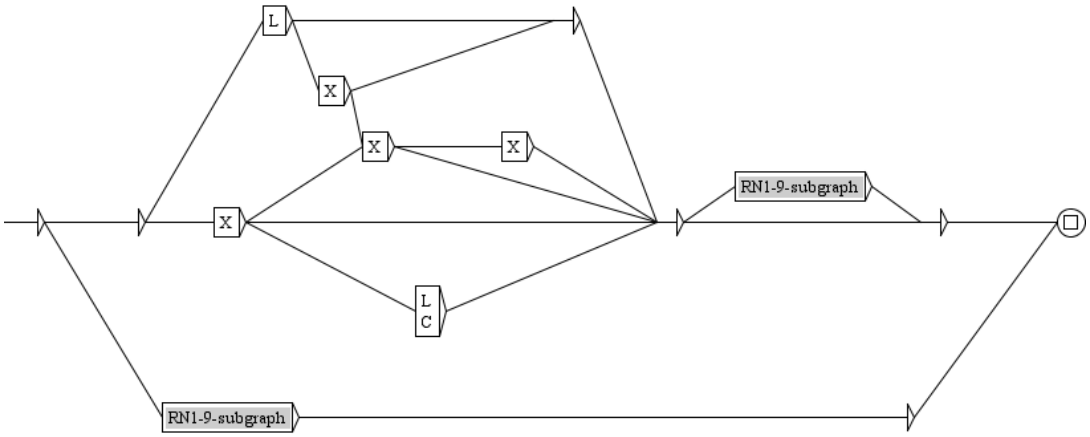
63 matches

the world. CHAPTER <romanNumeral>II</romanNumeral>. In which Timely Provisi
accustomed. CHAPTER <romanNumeral>III</romanNumeral>. In which Mr Dombey, as
hstanding!" CHAPTER <romanNumeral>IV</romanNumeral>. In which some more Firs
, far away. CHAPTER <romanNumeral>IX</romanNumeral>. In which the Wooden Mid
. "Anything about a <romanNumeral>L</romanNumeral>," said the Captain, "or a
f victuals. CHAPTER <romanNumeral>L</romanNumeral>. Mr Toots's Complaint The
he chips out P-A-U-<romanNumeral>L</romanNumeral> in the marble slab before
r, jintly!" CHAPTER <romanNumeral>LI</romanNumeral>. Mr Dombey and the World
e together. CHAPTER <romanNumeral>LII</romanNumeral>. Secret Intelligence Go
him of it. CHAPTER <romanNumeral>LIII</romanNumeral>. More Intelligence The
was empty. CHAPTER <romanNumeral>LIV</romanNumeral>. The Fugitives Tea-time
intry wind. CHAPTER <romanNumeral>LIX</romanNumeral>. Retribution Changes ha
re shining. CHAPTER <romanNumeral>LV</romanNumeral>. Rob the Grinder loses h
n of ashes. CHAPTER <romanNumeral>LVI</romanNumeral>. Several People delight
dear love. CHAPTER <romanNumeral>LVII</romanNumeral>. Another Wedding Mr So
y far away! CHAPTER <romanNumeral>LVIII</romanNumeral>. After a Lapse The se
isposed of. CHAPTER <romanNumeral>LX</romanNumeral>. Chiefly Matrimonial The
too much!" CHAPTER <romanNumeral>LXI</romanNumeral>. Relenting Florence had
f a dream." CHAPTER <romanNumeral>LXII</romanNumeral>. Final A bottle that h
Daughter!" CHAPTER <romanNumeral>V</romanNumeral>. Paul's Progress and Chri
ering down. CHAPTER <romanNumeral>VI</romanNumeral>. Paul's Second Deprivati
s about it. CHAPTER <romanNumeral>VII</romanNumeral>. A Bird's-eye Glimpse o

3.3 Third step: from 2 to 999

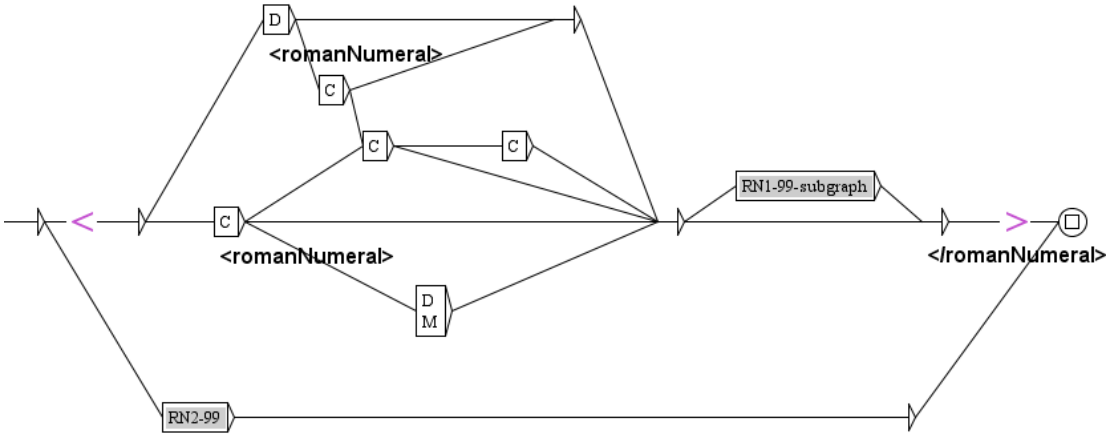
3.3.1 The subgraph from 1 to 99

We will repeat the same procedure to produce the *RN2-9-subgraph.grf*, namely, we will cancel the morphological mode and the insertion of tags. We should not forget to replace the call to the *RN2-9.grf* subgraph with *RN1-9-subgraph.grf* because the later subgraph does not insert tags, does not impose morphological mode and recognizes *I* (this subgraph is never used independently).



3.3.2 The graph from 2 to 999

In order to recognize Roman numerals between 2 and 999, we will again use the already developed *RN2-99.grf* graph in which we will replace *X* with *C*, *L* with *D* and *C* with *M*. Again, we will replace the call to the *RN2-9.grf* subgraph with *RN2-99.grf* outside the morphological mode (for numerals between 2 and 99), and a call to the subgraph with *RN1-99-subgraph.grf* (inside the morphological mode) for numerals between 1 to 99, when they are added to hundreds. We will name this graph *RN2-999.grf.*

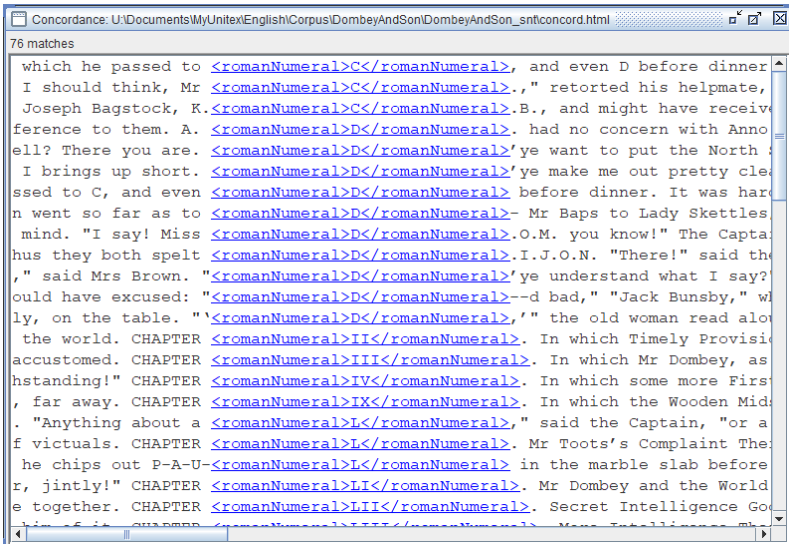


When we apply this graph to our text, we will obtain 76 concordance lines. But there are no Roman numerals greater than 62 (the last chapter) in this novel, so these thirteen new lines are due to false recognitions: The Roman numerals for 100 and 500 are confused:

- With the letters *C* and *D*.[61]
- With abbreviations, i.e. *K.C.B.*: *Knight Commander of the Bath* (see the Wikipedia).[62]

[61] *He acquitted himself very well, nevertheless; and Miss Blimber, commending him as giving promise of getting on fast, immediately provided him with subject B; from which he passed to C, and even D before dinner*.

[62] *If that had been Joe's character, Joe might have been, by this time, Lieutenant-General Sir Joseph Bagstock, K.C.B., and might have received you in very different quarters*.



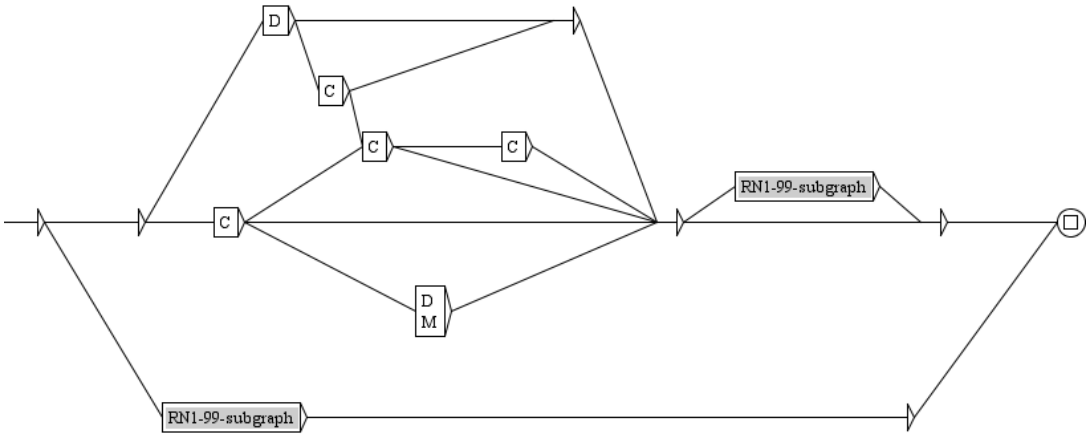
Unfortunately, without linguistic analysis, the ambiguity of the language prevents us from distinguishing a letter from a Roman numeral. However, one could assume that in general (except at the end of a sentence!), a Roman numeral is not followed by a period or an apostrophe (as *D'ye want* for *Do you want*). We could add this rule to our graph, with a negative right context (see Section 1.7.3, page 55). In fact, we were lucky so far, because we could have had in the concordances, from the first graph, a confusion either with the letters *V* or *X*, possibly used in an abbreviation. Let us finish our complete recognition of all the Roman numerals and we will see later what we can do about ambiguities (Section 3.5, page 68).

3.4 Fourth step: from 2 to 3 999

The numbers greater than 3 999 are generally not written as Roman numerals because there are no symbols for 5 000 and 10 000.

3.4.1 The subgraph from 1 to 999

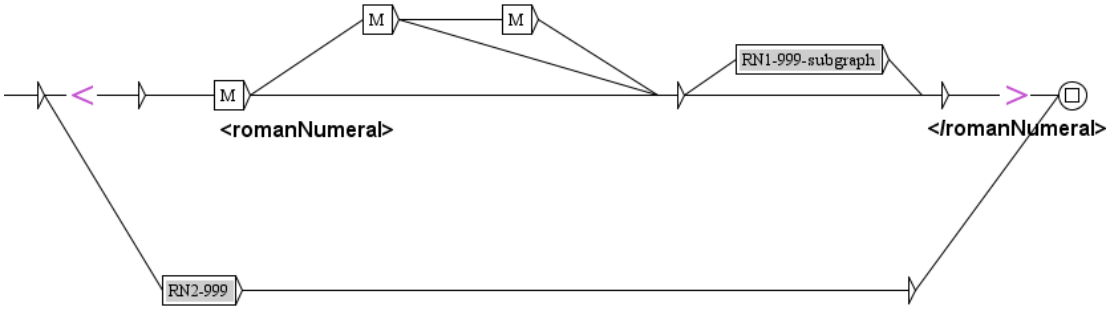
We will repeat the same procedure for the third time. Starting with the *RN2-999.grf* graph we will modify it by canceling the morphological mode and the insertion of tags. We should not forget to replace the call to the *RN2-99.grf* subgraph with *RN1-99-subgraph.grf* because the later subgraph does not insert tags, does not impose morphological mode and recognizes *I*.



3.4.2 The graph from 2 to 3 999

In order to recognize thousands (one, two and three thousand) it is enough to take the graph *RN2-999.grf*, to replace in it, *C* with *M*, and to delete all other paths. As before, we will replace

a call to the subgraph with *RN2-999.grf* outside the morphological mode (for numerals between 2 to 999), and a call to the subgraph with *RN1-999-subgraph.grf* (inside the morphological mode) for numerals between 1 to 999 when the are added to thousands. We will name this graph *RN2-3999.grf*.



The application of this graph to our text yields 81 matches. Five new false recognitions are added to those previously detected by the occurrence of "M" with abbreviations, i.e. *M.A.*.: *Master of Arts* (see the Wikipedia).[63]

The continuation of this exercise (Section 3.5) is only for confident users.

3.5 Ambiguities of Roman numerals (only for confident users)

> Before starting this section
>
> If you have not modified the *Norm.txt* file as explained in Chapter 2, Section 3.1.2, page 35, do so, or download it, then close and reopen Unitex.

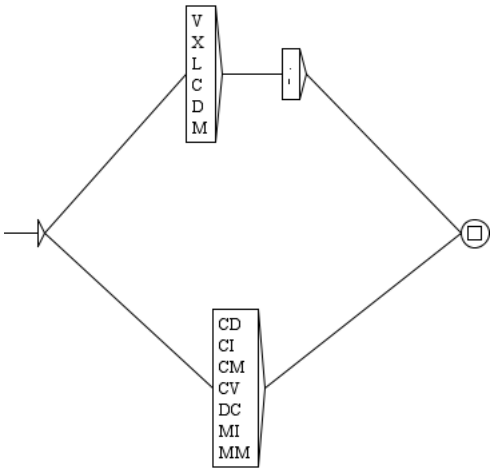
As we said in Section 3.3.2, page 66, Roman numerals are ambiguous with letters or abbreviations; and it is almost impossible to disambiguate them. We can reduce these ambiguities by assuming that a Roman numeral is not followed by a full stop or an apostrophe. However, false annotations cannot be completely avoided, because a Roman numeral can occur at the end of a sentence and thus be followed by a period. Let us also add a list of seven common words that are ambiguous with a number written in Roman numerals: *CD* (compact

[63] *the Reverend Alfred Feeder, M.A., who was to perform the ceremony*.

disc), *CI* (criminal investigation), *CM* (College of Medicine), *CV* (curriculum vitae), *DC* (District of Columbia), *MI* (Michigan) and *MM* (abbreviation of month or of millimeter).[64]

3.5.1 The subgraph of ambiguous Roman numerals

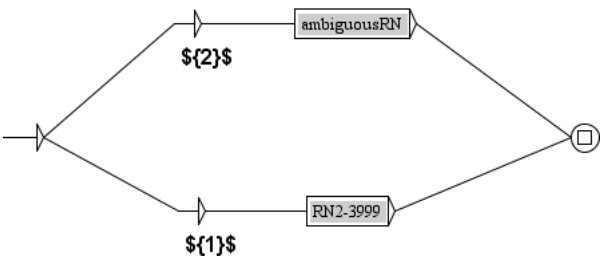
Let us create a graph based on the above remarks. This graph, named *ambiguousRN.grf* does not insert tags.



3.5.2 The use of weights

If we create a graph with two subgraphs, *RN2-3999* and *ambiguousRN*, and if we use it to recognize, for instance, the sequence *CD*, we do not know if the result will be <romanNumeral>CD</romanNumeral> (*:RN2-3999* path) or *CD* (*:ambiguousRN* path). We would not like to annotate *CD* as a Roman numeral. The *Longest matches* option cannot help here, because both annotations are equally long. In order to get what we want, we have to use weights. We will save this graph as *nonAmbiguousRN.grf*.

One can assign a weight to a box by inserting ${*weight*}$ in its output where *weight* is a positive integer or zero (*weight* ≥ 0). The path with the highest last weight has the priority. The use of weights is interesting for graphs that produce output because it enables choosing an output when there are several possibilities (as in our case). One must keep in mind that weights apply only to cases when two (or more) graph paths recognize exactly the same sequence of tokens; otherwise, the *Longest matches* option reduces the range of possibilities to those paths that recognize the longest sequence. If one path has several weights, only the last one is applied. A path with a weight always has priority over a path without any weight.



If you apply the *nonAmbiguousRN* graph to the text, you will obtain 81 concordance lines, but only 65 with the *romanNumeral* tags. Of these, only the seven capital letters are errors while

[64] This list is language dependent. For example, for French, DIX is the number ten, and so on.

the other tags correspond to 58 chapters (the first is not counted, but also the fifth, tenth and fiftieth, due to the period after the Roman numeral).

Chapter 4: CasSys

Before starting this chapter

1. Go to the folder: *MyUnitex\English\Corpus\UnitexGettingStarted\UnitexGettingStarted*
Create three new folders named (without space): *JubileePlainText* and *JubileeXmlText*.
Go to the folder:
MyUnitex\English\Corpus\UnitexGettingStarted\UnitexGettingStarted\JubileePlainText
Download the *jubileePlainText.txt* file.
Go to the folder: *MyUnitex\English\Corpus\UnitexGettingStarted\JubileeXmlText*
Download the *JubileeXmlText.xml* file.[65]

2. Go to the folder: *MyUnitex\English\Graphs\UnitexGettingStarted*
Create two new folders named *Entities* and *Measures*.
Go to the folder: *MyUnitex\English\Graphs\UnitexGettingStarted\Entities*
Create two new folders named *Analysis* and *Synthesis*.
Go to the folder: *MyUnitex\English\Graphs\UnitexGettingStarted\Measures*
Create two new folders named *Analysis* and *Synthesis*.

3. Go to the folder: *MyUnitex\English\CasSys*
Create a new folder named *UnitexGettingStarted* (without space).
Go to the folder: *MyUnitex\English\CasSys\UnitexGettingStarted*
Create three new folders named *PlainTextEntities*, *XmlTextEntities* and *Measures*.

1 First example: named entity recognition in a plain text

We will introduce cascades that analyze texts with aim to annotate named entities in a text (dates and places). First, we will use the text *jubileePlainText.txt*.

This week we celebrate the Platinum Jubilee of the Queen of England: seventy years of reign! The party will start on June 2, 2022 and end on June 5. The Queen ascended the British throne on February 6, 1952.
Our small town of Nearlondon will also celebrate this event. Nearlondon has always been a loyal subject of Her Majesty. Although the city of London is dear to us, on Thursday June 2 we will change the name of London Street to Queen Elizabeth Street.
Everyone will fondly remember 2022!
God Save the Queen!

We will open this text in Unitex and answer *Yes* to the question *Do you want to preprocess the text*. A window will open in which we will remove the first two checkmarks, leaving just the last one, *Apply All default Dictionaries*.

1.1 Analysis cascade

All graphs in the Section 1.1 will be saved in the folder:

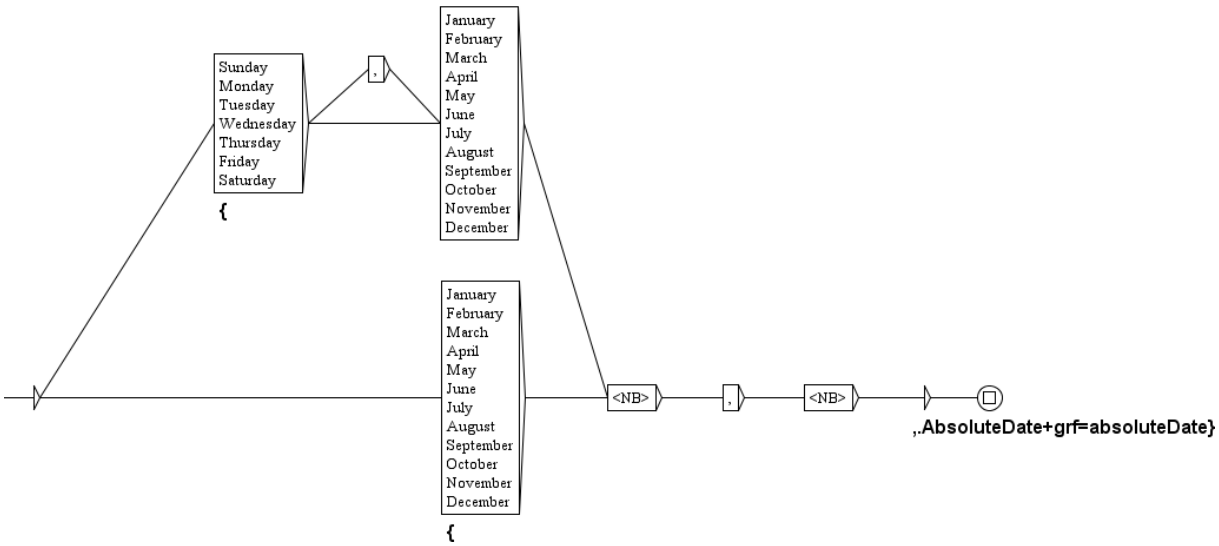
MyUnitex\English\Graphs\UnitexGettingStarted\Entities\Analysis

[65] These two texts were prepared specifically to illustrate this chapter.

1.1.1 The absoluteDate.grf *graph*

Let us create a first graph, which will recognize dates where the year is specified; we will call such dates *absolute*. In order to simplify this graph, we will not describe the days of months and years in detail but will instead use the lexical mask *<NB>*, which recognizes strings consisting of digits. We will put the recognized date between symbols *{* and *,.AbsoluteDate}*. By doing so, we are producing a lexical tag.[66] Assuming that a date in a text is *June 23, 2022*, the graph we are building will recognize it and replace it with a lexical tag {June 23\, 2022,.AbsoluteDate}. This text segment does not consist of eight tokens anymore, but just one, *June 23, 2022* and this token is classified as *AbsoluteDate*; in other words, the part-of-speech field in the lexical tag contains AbsoluteDate, in the same way as *June* was classified before as *N* (noun) by the application of dictionaries.[67]

In this graph, we will duplicate the box containing the names of months in order to avoid opening braces twice in the same path. We will add a tracing feature to the recognized sequence: the name of the graph itself *+grf=absoluteDate*, which can be useful in the debugging process. We will save this graph as *absoluteDate.grf*. In order to use the graph in a cascade (see Section 1.1.3, page 73), the saved graph must be compiled by clicking the *Compile graph* button at the left corner of the graph processing toolbar, next to the *Save graph* button.

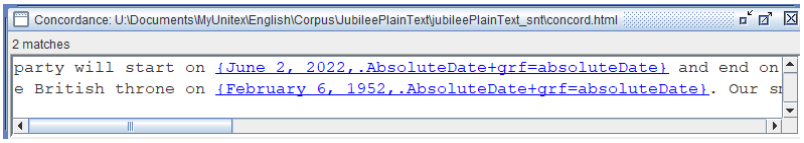


We will create a concordance with this graph. After clicking the *SEARCH* and *OK* buttons, the *Located sequences* window appears. At the bottom right of this window, we will open the *Sort according to* drop-down menu and choose the *Text order* option, instead of *Center, Left*. We will click on the *Build concordance* button and the concordance lines, in this case two lines, will be displayed in the order in which matched sequences appear in the text.

---

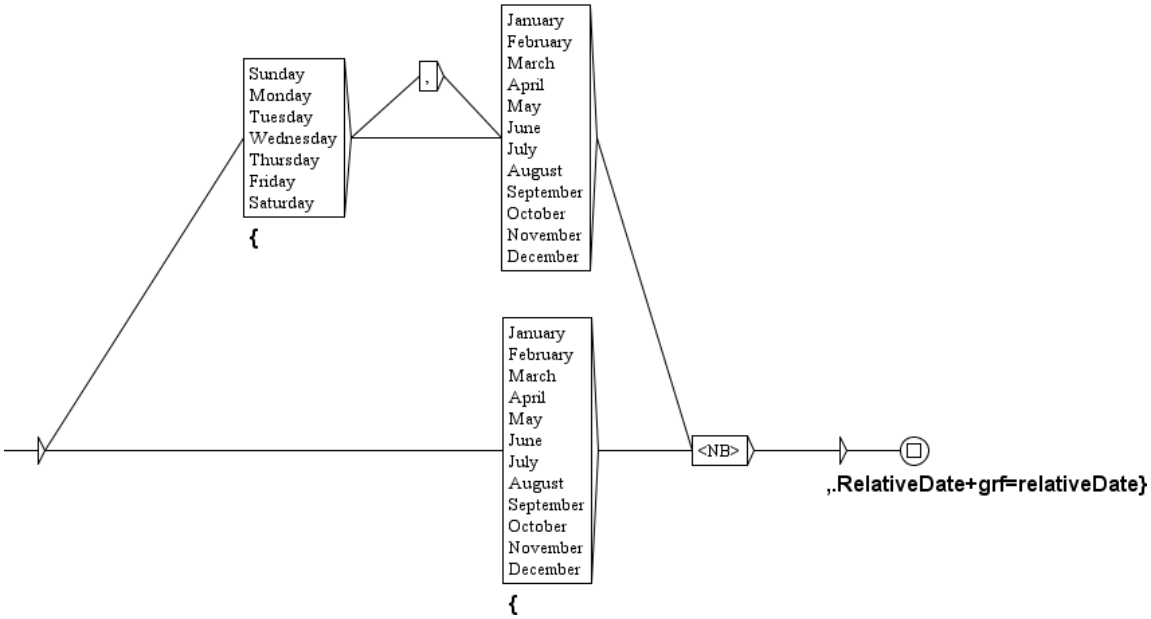
[66] See Chapter 2, Section 1.2.5, page 15. The common way to insert a *{* character is by the box that recognizes the beginning of the phrase you want to tag. If the *{* character is inserted just before, by a separate <E> box, the inserted *{* character can be followed by a space in the output, which is unsatisfactory because the braces are supposed to delimit a precise phrase.

[67] Remember that each alphabetic string is one token (*June* in this case), a punctuation one token (*,* in this case), and each digit is a separate token (*2*, *3*, *2*, *0*, *2*, *2* in this case).

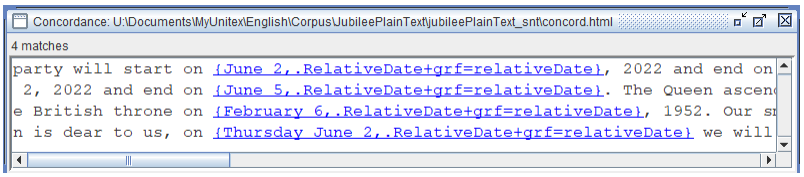


1.1.2 *The* relativeDate.grf *Graph*

We will now create the graph that recognizes relative, or incomplete, dates, when the year is not mentioned. We will use the previously constructed *absoluteDate.grf* graph, remove in it the boxes for year and separating comma and save it under the new name *relativeDate.grf* (we will use the *FSGraph/Save as…* menu). Do not forget to modify the output as well: *,.AbsoluteDate+grf=absoluteDate}* should be replaced by *,.RelativeDate+grf=relativeDate}*. After compiling the graph, we will apply it to the text.



Now we will obtain four concordance lines in which all dates will be annotated as relative dates, including two recognized by the previous graph (note that in these cases, the years are outside the scope of the lexical tags). In order to correct this, we will apply these two graphs one after the other, using what is known as a cascade of graphs.



1.1.3 *The* analysis.csc *cascade*

Now we will build a small cascade containing both graphs. We will open the *Text/Apply CasSys Cascade…* menu and click the *New* button. The Cascade configuration frame will open. Then we must find the location of the graphs that we want to put into the cascade, in this case:

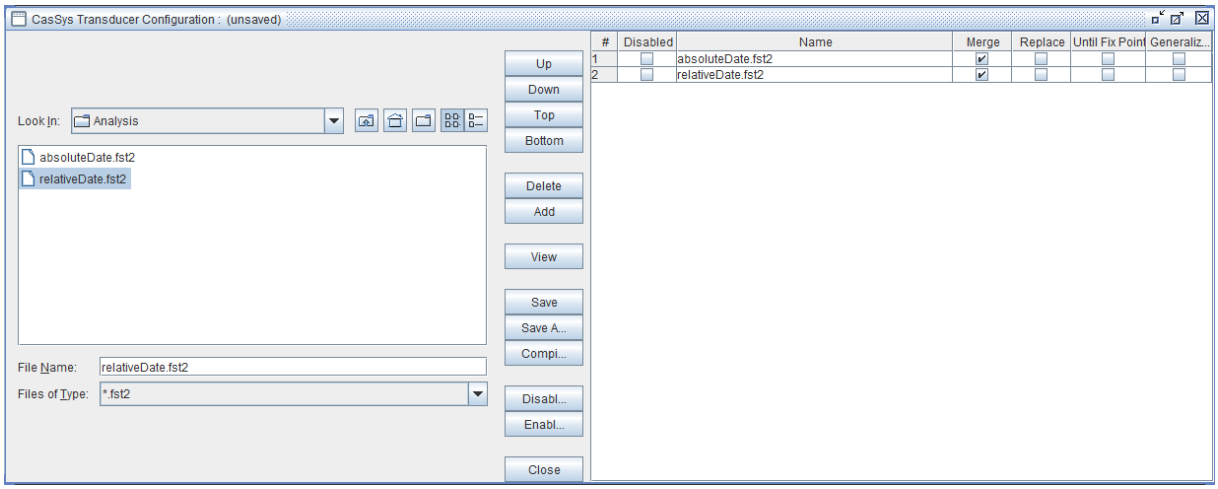
MyUnitex\English\Graphs\UnitexGettingStarted\Entities\Analysis

We will notice that the left window shows only files with the *.fst2* extension that is only graphs that have been compiled. It means that we can put into a cascade only already compiled graphs.[68]

With the mouse, we can drag and drop the *absoluteDate.fst2* graph into the right window; and we will repeat the same with the *relativeDate.fst2* graph. Alternatively, we can select a graph and click the *Add* button when we want to add it to a cascade. We will save this cascade as *analysis.csc* in the folder:

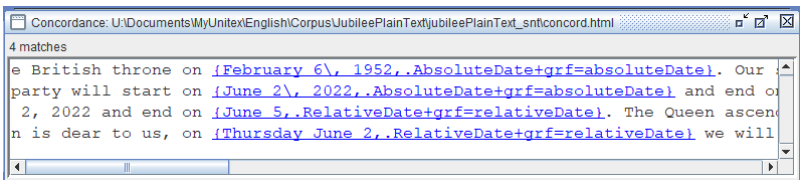
MyUnitex\English\CasSys\UnitexGettingStarted\PlainTextEntities

by using the *Save* button.



We will close this frame by clicking the *Close* button.

Now we will select our cascade and click the *Launch* button, which will initiate the analysis. When analysis is over, we can build concordances. We will obtain four concordance lines, but now the dates will be correctly annotated. Why is it so? Because the first graph in the cascade, *absoluteDate.grf,* has annotated all absolute dates and produced lexical tags. As we explained before, a lexical tag is treated as one token, and the subsequently applied graph *relativeDate.grf* cannot analyze inside its content.



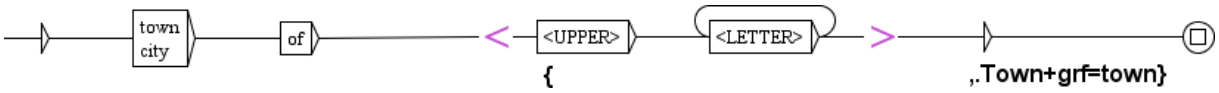
1.1.4 *The* town.grf *graph*

We will now prepare the graph that will recognize names of towns and cities and annotate them as *Town*.[69] We will presume that any sequence of letters of the English alphabet

[68] If we use subgraphs, only the main graph needs to be compiled, not the subgraphs.

[69] One has to be careful when choosing the annotation tags in order to avoid ambiguity with dictionary lemmas. For instance, if we had chosen *town* instead of *Town* as a tag for inhabited places, then the lexical mask *<town>* would correspond to recognized names of towns, but also the lemma *town*, that is, its forms *town* and *towns.* On the contrary, the lexical mask *<Town>* corresponds only to the annotated names of towns (remember that lexical masks are case sensitive).

(<LETTER>) starting with an uppercase letter (<UPPER>) and following the sequences *city of* or *town of* is the name we want to annotate. In order to recognize such a sequence, we must use the morphological mode in order to describe the structure of a single word.[70] We will save this graph as *town.grf* and compile it.



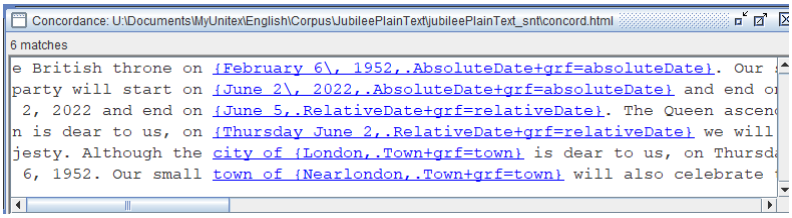
We will drag and drop this graph to the last position in the cascade. As a reminder: open the *Text/Apply CasSys Cascade…* menu, then, find the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Entities\Analysis

select the *analysis.csc* cascade, click the *Edit* button, move to the folder containing the graphs, then drag and drop the *town.fst2* graph to the list of graphs. Save the cascade.

#	Disabled	Name	Merge	Replace	Until Fix Point	Generaliz...
1		absoluteDate.fst2	✔			
2		relativeDate.fst2	✔			
3		town.fst2	✔			

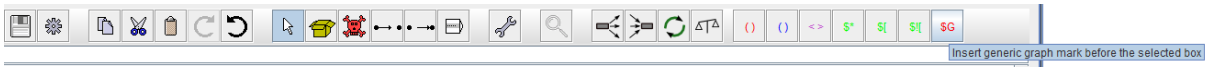
After launching this cascade two new concordance lines will be produced by the *town.grf* graph with *Nearlondon* and *London* annotated as *Town*.



1.1.5 Generalization graphs

If you look at the short text we are annotating, you will notice that *Nearlondon* in the sequence *Nearlondon has always been…* was not annotated as *Town* because it was preceded neither with *town of* nor *city of.* Moreover, *Nearlondon* is not even in the dictionary of the text, it is an unknown word.

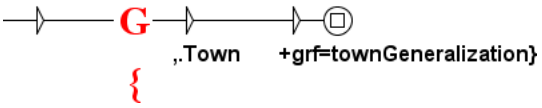
In order to solve this problem we will use a generalization graph which will annotate a word (or a sequence of words) even when it does not appear in a required context, if the same word appeared somewhere else in the same text in the required context, and was annotated by some graph in a cascade. We have such a situation with the word *Nearlondon* in our text: it was annotated once as *Town* by the *town.grf* graph, because it occurred in the required context, so its second occurrence will be annotated as *Town* by the *townGeneralization.grf* graph.



In order to create a simple generalization graph it is enough to introduce a box with the output consisting of the annotation that has to be generalized (in our case *,.Town*), select this box and

[70] See Chapter 3, Section 3.1, page 63. In CasSys cascades, the *<FIRST>* or *<UPPER>* lexical mask does not always detect if a word has an uppercase initial letter, because some words or expressions may have been replaced by lexical tags, as in the preceding examples. This is why we show another way of detecting words with an uppercase initial.

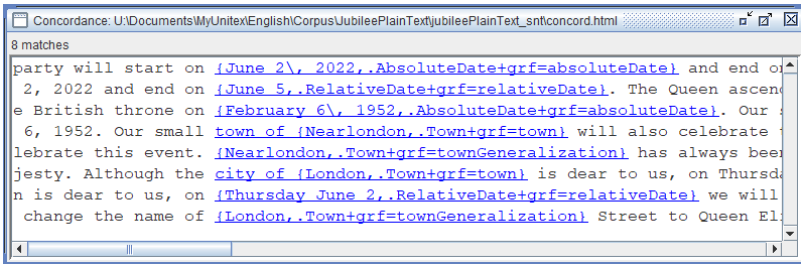
click to the *Insert generic graph mark before the selected box* button at the far right end of the toolbar.[71]



After compiling this graph, we will add it as the last one in the list of graphs in the *analysis.csc* cascade (remember that it has to be in the list after the *town.grf* graph) in the same way as we have done before.[72] We must do one more thing: the configuration panel offers options for the modes of graph application. By default, this option is *Merge* mode, which suited us so far for all the graphs in our list. However, for this last graph we have also to check the *Generaliz…* box in order to signal that this is a generalization graph.

#	Disabled	Name	Merge	Replace	Until Fix Point	Generaliz...
1	☐	absoluteDate.fst2	☑	☐	☐	☐
2	☐	relativeDate.fst2	☑	☐	☐	☐
3	☐	town.fst2	☑	☐	☐	☐
4	☐	townGeneralization.fst2	☑	☐	☐	☑

After launching our cascade, we will obtain a concordance with two new lines, one of them with *Nearlondon* annotated although it was preceded neither with *city of* nor *town of*. We can see that the second occurrence of *London* was annotated as well, although we did not want that, since the context is *change the name of London Street to Queen Elizabeth Street*.



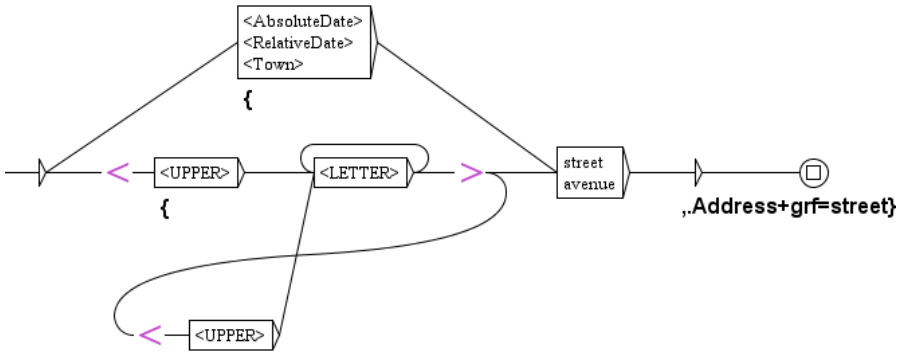
1.1.6 *The* street.grf *graph*

We will now create a graph that recognizes names of streets with the *Address* category. We will recognize two patterns before the words *street* and *avenue*: (1) a date or a town name already recognized by the cascade and tagged as *<AbsoluteDate>*, *<RelativeDate>* or *<Town>*;[73] (2) one or more upper-case words. In order to make the loop for uppercase words, we can duplicate two boxes (the big angular bracket and the lexical mask *<UPPER>*), but without the opening braces as output. Alternatively, we can add the new *<UPPER>* box, surround it with morphological mode angular brackets by clicking the *Surround box selection with morphological mode tags* button, and then delete the unnecessary closing bracket and connect the box with *<LETTER>*.

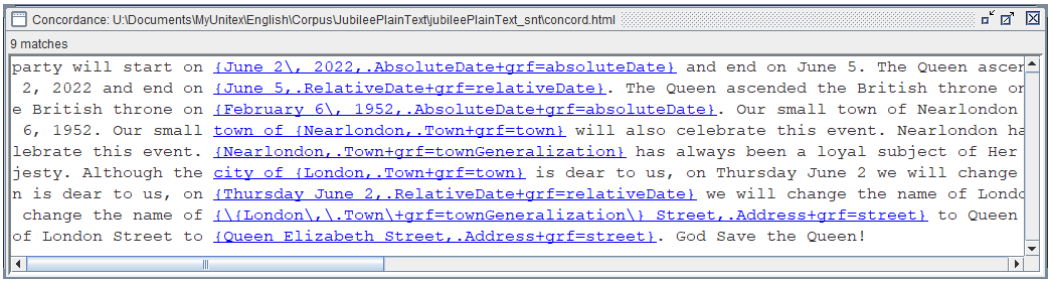
[71] Generalization graphs for annotation can be more sophisticated; see the Section 4.5.2, page 104.

[72] The *townGeneralization.grf* file must be compiled before we place it in the cascade, but the program uses also the *.grf* file which must therefore also be present in the folder, whereas this is not obligatory for other graphs.

[73] This enables recognition of *May 8, 1945 Avenue*.



After saving the graph with the name *street.grf* we will compile it and add it as the last graph in our *analysis.csc* cascade to be used in the *Merge* mode. After launching the cascade and producing a concordance, we will obtain nine concordance lines. One new line contains the match *Queen Elizabeth Street* recognized by the second rule in the *street.grf* graph (two upper-case words). The first rule is applied to *London Street* since *London* was already tagged as *Town*. If we look at that line in the concordance list (it is the second-to-last line in the *Text Order* option), we will see that there we have nested tagging, that is a lexical tag inside another lexical tag.[74]

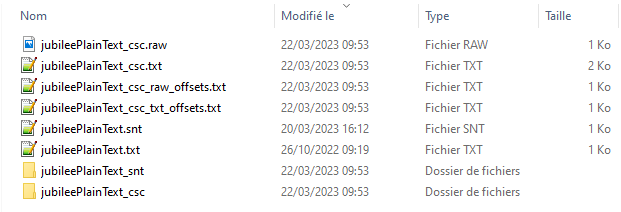


1.1.7 Created files

The role of the Cassys program is not only to produce concordances; it produces as output numerous files. If we open the folder:

MyUnitex\English\Corpus\UnitexGettingStarted\jubileePlainText

we will see in it, besides the original *jubileePlainText.txt* file, also *jubileePlainText.snt* and the *jubileePlainText_snt* folder, as for all texts processed by Unitex. Besides this, we will see four more files with the suffix *_csc* and one new folder.



Nom	Modifié le	Type	Taille
jubileePlainText_csc.raw	22/03/2023 09:53	Fichier RAW	1 Ko
jubileePlainText_csc.txt	22/03/2023 09:53	Fichier TXT	2 Ko
jubileePlainText_csc_raw_offsets.txt	22/03/2023 09:53	Fichier TXT	1 Ko
jubileePlainText_csc_txt_offsets.txt	22/03/2023 09:53	Fichier TXT	1 Ko
jubileePlainText.snt	20/03/2023 16:12	Fichier SNT	1 Ko
jubileePlainText.txt	26/10/2022 09:19	Fichier TXT	1 Ko
jubileePlainText_snt	22/03/2023 09:53	Dossier de fichiers	
jubileePlainText_csc	22/03/2023 09:53	Dossier de fichiers	

1.1.7.1 The _csc files

The *jubileePlainText_csc.raw* file contains the text modified by the cascade. Namely, all sequences recognized by cascade graphs are replaced by lexical tags enclosed in braces.

[74] In order to do that the delimiting characters of the internal lexical tag (*{.+}*) are protected by the backslash character.

```
This week we celebrate the Platinum Jubilee of the Queen of England: seventy years of reign! The
party will start on {June 2\, 2022,.AbsoluteDate+grf=absoluteDate} and end on {June
5,.RelativeDate+grf=relativeDate}. The Queen ascended the British throne on {February 6\,
1952,.AbsoluteDate+grf=absoluteDate}.
Our small town of {Nearlondon,.Town+grf=town} will also celebrate this event.
{Nearlondon,.Town+grf=townGeneralization} has always been a loyal subject of Her Majesty.
Although the city of {London,.Town+grf=town} is dear to us, on {Thursday June
2,.RelativeDate+grf=relativeDate} we will change the name of
{\{London\,\.Town\+grf=townGeneralization\} Street,.Address+grf=street} to {Queen Elizabeth
Street,.Address+grf=street}.
Everyone will fondly remember 2022!
God Save the Queen!
```

The *jubileePlainText_csc.txt* file is a quasi XML version of the same output in which the braces are replaced by XML tags *<csc></csc>*, the recognized text is surrounded by tags *<form></form>*, while the code in the POS field and its additional features appear in tags *<code></code>*.[75]

```
This week we celebrate the Platinum Jubilee of the Queen of England: seventy years of reign! The
party will start on <csc><form>June 2,
2022</form><code>AbsoluteDate</code><code>grf=absoluteDate</code></csc> and end on
<csc><form>June 5</form><code>RelativeDate</code><code>grf=relativeDate</code></csc>.
The Queen ascended the British throne on <csc><form>February 6,
1952</form><code>AbsoluteDate</code><code>grf=absoluteDate</code></csc>.
Our small town of
<csc><form>Nearlondon</form><code>Town</code><code>grf=town</code></csc> will also
celebrate this event.
<csc><form>Nearlondon</form><code>Town</code><code>grf=townGeneralization</code></cs
c> has always been a loyal subject of Her Majesty. Although the city of
<csc><form>London</form><code>Town</code><code>grf=town</code></csc> is dear to us, on
<csc><form>Thursday June
2</form><code>RelativeDate</code><code>grf=relativeDate</code></csc> we will change the
name of
<csc><form><csc><form>London</form><code>Town</code><code>grf=townGeneralization</c
ode></csc> Street</form><code>Address</code><code>grf=street</code></csc> to
<csc><form>Queen Elizabeth
Street</form><code>Address</code><code>grf=street</code></csc>.
Everyone will fondly remember 2022!
God Save the Queen!
```

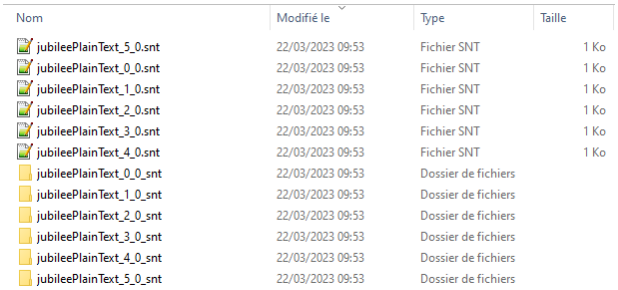
The remaining two files contain offsets corresponding to two previously described files.[76]

1.1.7.2 The _csc folder

The folder with the name ending with *_csc* contains all intermediary files, which are very useful in debugging cascades.

[75] *<lemma></lemma>* is one more pair of XML tag that can appear in the content of *<csc></csc>*. It does not appear in our case because in the description of all lexical tags the lemma field (between the comma and the period) is empty. If it were not the case, this field would be copied here enclosed in *<lemma></lemma>* tags.

[76] See the Unitex user manual, Section 15.13.10.



Nom	Modifié le	Type	Taille
jubileePlainText_5_0.snt	22/03/2023 09:53	Fichier SNT	1 Ko
jubileePlainText_0_0.snt	22/03/2023 09:53	Fichier SNT	1 Ko
jubileePlainText_1_0.snt	22/03/2023 09:53	Fichier SNT	1 Ko
jubileePlainText_2_0.snt	22/03/2023 09:53	Fichier SNT	1 Ko
jubileePlainText_3_0.snt	22/03/2023 09:53	Fichier SNT	1 Ko
jubileePlainText_4_0.snt	22/03/2023 09:53	Fichier SNT	1 Ko
jubileePlainText_0_0_snt	22/03/2023 09:53	Dossier de fichiers	
jubileePlainText_1_0_snt	22/03/2023 09:53	Dossier de fichiers	
jubileePlainText_2_0_snt	22/03/2023 09:53	Dossier de fichiers	
jubileePlainText_3_0_snt	22/03/2023 09:53	Dossier de fichiers	
jubileePlainText_4_0_snt	22/03/2023 09:53	Dossier de fichiers	
jubileePlainText_5_0_snt	22/03/2023 09:53	Dossier de fichiers	

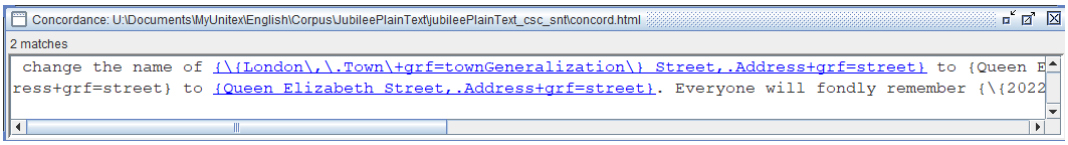
The original text is copied in this folder and renamed as *jubileePlainText_0_0.txt*. The result of the first graph in the cascade is given the name *jubileePlainText_1_0.txt*, and so on. If some graph in the cascade is used in the *until fix point* mode (this will be explained in the following sections) it means that it will be repeatedly used and the second number in a file's name tells which iteration produced the file (in our present example it is always *0*).

1.2 Synthesis cascade

We have seen in the previous section that there are two outputs of an analysis cascade.

1. The *_csc.raw*

In the *_csc.raw* version of the text the recognized matches are transformed into lexical tags that can be subsequently used in Unitex for further text exploration. It can be used in Unitex for searching in the usual way. We can open the text *jubileePlainText_csc.raw*, without preprocessing, and then launch with *Locate pattern* a search using classical as well as new categories.[77] For instance, a pattern *<Address>* retrieves two matches:



2. The *_csc.txt*

The *_csc.txt* version of the text is the resulting text where the lexical tags are replaced by XML tags. This format is used to transform the output of an analysis cascade into whatever other format needed.

We will now open (with the *Text/Open* menu) the file:

MyUnitex\MyUnitex\English\Corpus\UnitexGettingStarted\jubileePlainText\jubileePlainText_csc.txt

and reply *No* to the question *Do you want to preprocess the text*.

We will delete tags *<csc>*, *<form>* and so on, and replace them with tags that we have chosen, namely, *<AbsoluteDate></AbsoluteDate>*, *<RelativeDate></RelativeDate>*, *<Town></Town>* and *<Address></Address>*.

1.2.1 *The* tag.grf *graph*

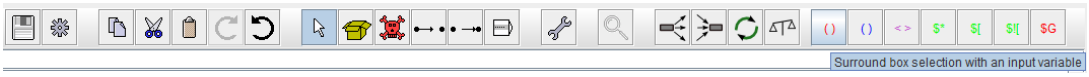
Let us create the first graph, which we will save in the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Entities\Synthesis

[77] That is to say *<N>*, *<V>* or *<ADV>*, but also, *<AbsoluteDate>*, *<Town>* or *<Address>*.

In this folder we will save the rest of the graphs for synthesis that we will introduce in this section. The aim of this graph that we will use in *Replace* mode is:

1. delete tags *csc*, *form* and *code*;
2. save the matched text in a variable *$text$* and its category in a variable *tag*;
3. replace all that with the tagged text: *<tag>$text$</tag>*; that is, everything recognized is replaced (thus the graph must be used in the *Replace* mode) by the recognized entity (*$text$*) enclosed in appropriate tags (*<tag></tag>*).



An input variable in Unitex stores the part of a text recognized by boxes inside (big red) parentheses. These parentheses are placed in the same way as context and morphological mode brackets by clicking on the *Surround box selection with an input variable* button (red parentheses at the right-hand side of the toolbar).

The user can freely choose the name of the variable. If the value of a variable is used in output, then it is surrounded by a pair of *$* character.

1.2.1.1 A first attempt

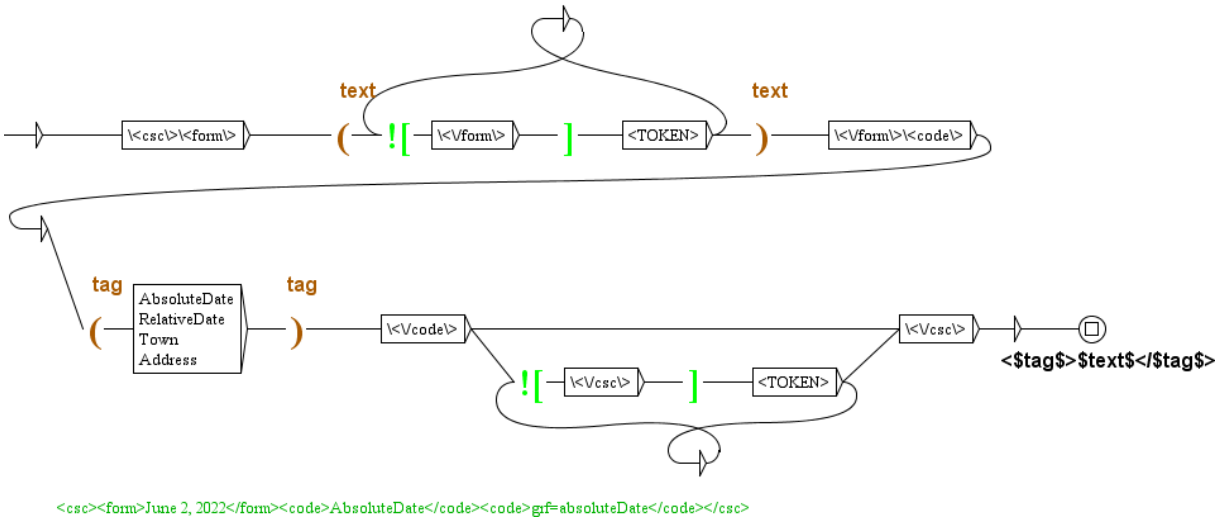
In order to collect the text we will use a loop that recognizes a continuous sequence of tokens (lexical mask: *<TOKEN>*) except that it will exclude *</form>* (the tag at the end of the recognized text).[78] After the loop, the graph will match one of the used categories, followed by everything until the end of the coded sequence that we are analyzing (*</csc>*).

A very important note: in Unitex queries, angle brackets have the special meaning, so if we want to recognize an angle bracket in a text, we have to *protect* it by preceding it with a backslash.[79] For instance, if we want to recognize *<csc>* in a text we have to write *\<csc\>* (otherwise it would be interpreted as a lexical mask recognizing all forms of the lemma *csc*). Similarly, since a slash also has a special meaning in Unitex graph boxes (it marks the beginning of an output) if we want to recognize it in a text, it also has to be preceded by a backslash. For instance, in order to recognize *</csc>* we must write *\<\/csc\>*.

In order to make this complex graph easier to understand, we will provide an example inside a comment box (a box starting with a slash and not connected with any other boxes). Text in such a box will appear in green.

[78] The negative right contexts (see Chapter 3, Section 1.7.3, page 55) are interpreted like this: *<TOKEN>* is recognized if it is not the *<* character, followed by */form>* (first negative right context), respectively */csc>* (second negative right context).

[79] See the Unitex user manual, Section 5.2.7. We do not have to take care about this in the output text - no protection is needed there.



We will now save the graph as *tag.grf*; compile it; create the new cascade in which we will add in the first position this graph; and choose the *Replace* mode by checking the appropriate box. We will call this cascade *synthesis.csc* and we will save it in the same folder as the *analysis.csc* cascade.

After launching this cascade, we will obtain a new text file:[80]

MyUnitex\MyUnitex\English\Corpus\UnitexGettingStarted\jubileePlainText\jubileePlainText_csc_csc.txt

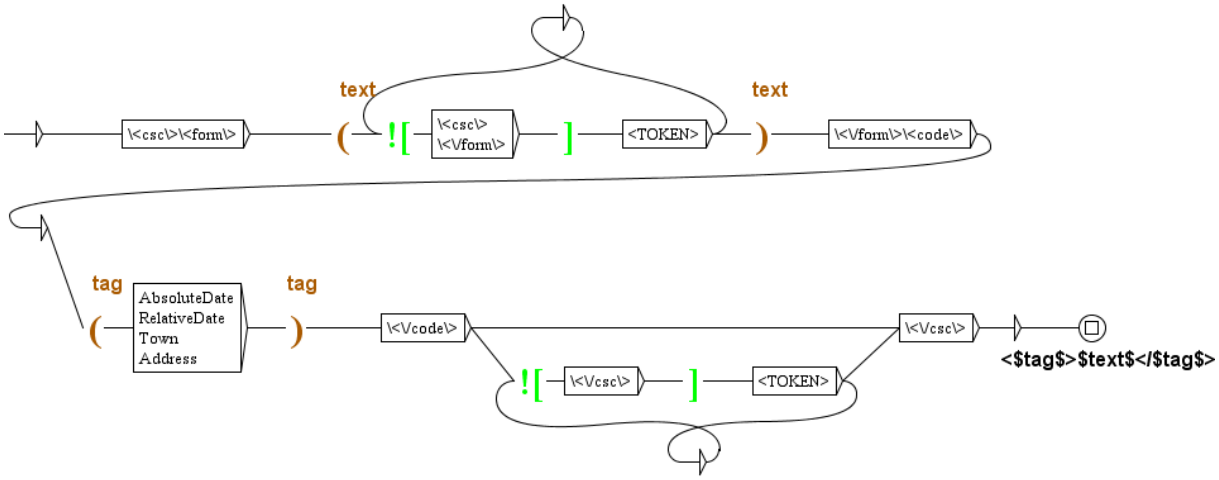
This week we celebrate the Platinum Jubilee of the Queen of England: seventy years of reign! The party will start on <AbsoluteDate>June 2, 2022</AbsoluteDate> and end on <RelativeDate>June 5</RelativeDate>. The Queen ascended the British throne on <AbsoluteDate>February 6, 1952</AbsoluteDate>.
Our small town of <Town>Nearlondon</Town> will also celebrate this event.
<Town>Nearlondon</Town> has always been a loyal subject of Her Majesty. Although the city of <Town>London</Town> is dear to us, on <RelativeDate>Thursday June 2</RelativeDate> we will change the name of <Town><csc><form>London</Town> Street</form><code>Address</code><code>grf=street</code></csc> to <Address>Queen Elizabeth Street</Address>.
Everyone will fondly remember 2022!
God Save the Queen!

We can observe in the resulting text that the transformation of annotations of non-nested elements was done successfully. For instance, the occurrence of the first date in the text became *<AbsoluteDate>June 2, 2022</AbsoluteDate>*. However, for nested elements, the result is surprising: *<Town><csc><form>London</Town> Street</form><code>Address</code><code>grf=street</code></csc>*! What happened is that the graph generated the opening tag using the internal markup *<Town>*.

[80] Because the graph in this cascade is used in the *Replace* mode, it is difficult to visualize the output as a concordance. We will therefore from now on display the file obtained as a result, which is in this case *jubileePlainText_csc_csc.txt*.

1.2.1.2 *A second attempt*

To avoid this, we will add *<csc>* (the beginning of a nested coded sequence) in the first negative context box in order to avoid deletion and replacement in cases when an entity has nested entities in its content.



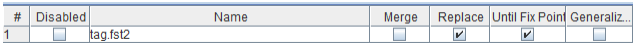
We will compile the graph and relaunch the cascade on *jubileePlainText_csc.snt*.

> This week we celebrate the Platinum Jubilee of the Queen of England: seventy years of reign! The party will start on <AbsoluteDate>June 2, 2022</AbsoluteDate> and end on <RelativeDate>June 5</RelativeDate>. The Queen ascended the British throne on <AbsoluteDate>February 6, 1952</AbsoluteDate>.
> Our small town of <Town>Nearlondon</Town> will also celebrate this event.
> <Town>Nearlondon</Town> has always been a loyal subject of Her Majesty. Although the city of <Town>London</Town> is dear to us, on <RelativeDate>Thursday June 2</RelativeDate> we will change the name of <csc><form><Town>London</Town> Street</form><code>Address</code><code>grf=street</code></csc> to <Address>Queen Elizabeth Street</Address>.
> Everyone will fondly remember 2022!
> God Save the Queen!

We can now observe in the resulting text that, this time, the nested element was well transformed: *<csc><form><Town>London</Town> Street</form><code>Address</code> <code>grf=street</code></csc>*.

1.2.1.3 *The* Until fix point *option*

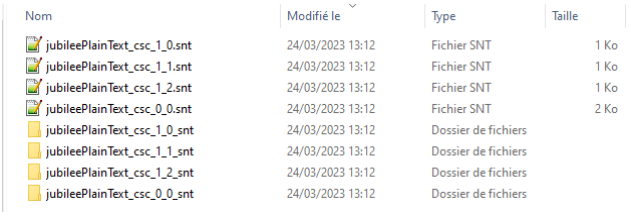
In order to transform the annotations of the external element it would be enough to launch the same graph for the second time (add it to the cascade). However, if we had a three-level nesting, then we would have to launch (and add to the cascade) the graph a third time. In order to avoid this, we will check the *Until Fix Point* box that will launch the *tag.fst2* graph as long as it makes some changes to the text.



After launching again this cascade (applying it to the text *jubileePlainText_csc.txt*) the resulting text *jubileePlainText_csc_csc.txt* becomes correct, and all annotations have been correctly transformed: *<Address><Town>London</Town> Street</Address>*.

This week we celebrate the Platinum Jubilee of the Queen of England: seventy years of reign! The party will start on <AbsoluteDate>June 2, 2022</AbsoluteDate> and end on <RelativeDate>June 5</RelativeDate>. The Queen ascended the British throne on <AbsoluteDate>February 6, 1952</AbsoluteDate>.
Our small town of <Town>Nearlondon</Town> will also celebrate this event.
<Town>Nearlondon</Town> has always been a loyal subject of Her Majesty. Although the city of <Town>London</Town> is dear to us, on <RelativeDate>Thursday June 2</RelativeDate> we will change the name of <Address><Town>London</Town> Street</Address> to <Address>Queen Elizabeth Street</Address>.
Everyone will fondly remember 2022!
God Save the Queen!

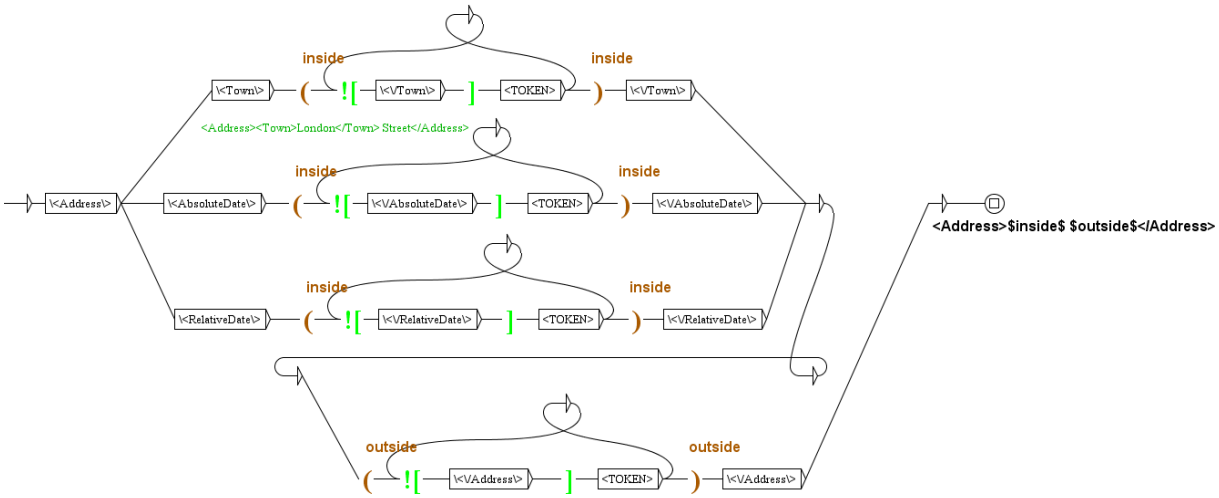
If we open the *JubileePlainText_csc_csc* folder, we will find in it four files (and four corresponding folders): the original text *jubileePlainText_csc_0_0.snt*, the result of the first run of the graph (resolves outer-level and one-level annotations), *jubileePlainText_csc_1_0.snt*, the result of the second run of the graph (resolves second-level annotations), *jubileePlainText_csc_1_1.snt*, the result of the third run of the graph, *jubileePlainText_csc_1_2.snt*, that establishes that the text was not modified (the last two files are identical).



1.2.2 The internalDeletion.grf *graph*

In some cases, we do not want nested annotations. For instance, annotating a date or a city occurring in an address is not obligatorily relevant. While recognizing dates and inhabited places has been essential for recognizing addresses, we would like to remove the date and town tags now that we have used them.

The *internalDeletion.grf* graph that will remove nested tags in the *<Address>* element begins with the recognition of the start tag *<Address>*, which is followed by the recognition of a date tag or a city tag; the content of this second element (start tag and end tag excluded) becomes the value of the variable *$inside$.* The end tag *</Address>* is then searched for and everything found between the end tag of the nested annotation and the end tag *</Address>* is remembered in the variable *$outside$*. The output of this graph, which works in the *Replace* mode, are tags *<Address></Address>* surrounding the content of the nested element (*$inside$*) followed by the content of the *<Address>* element itself (*$outside$*).



After saving and compiling the graph, we will add it as the second graph in the *synthesis.csc* cascade (do not forget to check the *Replace* box).

#	Disabled	Name	Merge	Replace	Until Fix Point	Generaliz...
1	☐	tag.fst2	☐	☑	☑	☐
2	☐	internalDeletion.fst2	☐	☑	☐	☐

After launching this cascade, we will get the desired result.

> This week we celebrate the Platinum Jubilee of the Queen of England: seventy years of reign! The party will start on <AbsoluteDate>June 2, 2022</AbsoluteDate> and end on <RelativeDate>June 5</RelativeDate>. The Queen ascended the British throne on <AbsoluteDate>February 6, 1952</AbsoluteDate>.
> Our small town of <Town>Nearlondon</Town> will also celebrate this event.
> <Town>Nearlondon</Town> has always been a loyal subject of Her Majesty. Although the city of <Town>London</Town> is dear to us, on <RelativeDate>Thursday June 2</RelativeDate> we will change the name of <Address>London Street</Address> to <Address>Queen Elizabeth Street</Address>.
> Everyone will fondly remember 2022!
> God Save the Queen!

2 Second example: named entity recognition in an XML text

We will work now with the same text but prepared in the XML format, *jubileeXmlText.xml*. This text is in the folder:

MyUnitex\English\Corpus\UnitexGettingStarted\jubileeXmlText

Please note that despite being an XML text it should not have the *.xml* extension because Unitex automatically removes in preprocessing all XML tags from a text having an*.xml* extension. In order to avoid this, we will rename our text to *jubileeXmlText.xml.txt*.

```
<xml>
<title>The municipal newspaper of the city of Nearlondon</title>
<date>Posted on Wednesday June 1, 2022</date>
<body>
<p>This week we celebrate the Platinum Jubilee of the Queen of England: seventy years of reign! The party will start on June 2, 2022 and end on June 5. The Queen ascended the British throne on February 6, 1952.</p>
<p>Our small town of Nearlondon will also celebrate this event. Nearlondon has always been a loyal subject of Her Majesty. Although the city of London is dear to us, on Thursday June 2 we will change the name of London Street to Queen Elizabeth Street.</p>
<p>Everyone will fondly remember 2022!</p>
<p>God Save the Queen!</p>
</body>
</xml>
```

We will open the text *jubileeXmlText.xml.txt* in Unitex (*Text/Open*) and answer *Yes* to the question *Do you want to preprocess the text*. A window will open in which we will remove the first two checkmarks, leaving just the last one *Apply All default Dictionaries*. We will launch the *analysis.csc* cascade (*Text/Apply CasSys Cascade*), then open the resulting text, *jubileeXmlText.xml_csc.txt*, and launch the *synthesis.csc* cascade. We will get as a result the *jubileeXmlText.xml_csc_csc.txt* file. Let us look at it:

```
<xml>
<title>The municipal newspaper of the city of <Town>Nearlondon</Town></title>
<date>Posted on <AbsoluteDate>Wednesday June 1, 2022</AbsoluteDate></date>
<body>
<p>This week we celebrate the Platinum Jubilee of the Queen of England: seventy years of reign! The party will start on <AbsoluteDate>June 2, 2022</AbsoluteDate> and end on <RelativeDate>June 5</RelativeDate>. The Queen ascended the British throne on <AbsoluteDate>February 6, 1952</AbsoluteDate>.</p>
<p>Our small town of <Town>Nearlondon</Town> will also celebrate this event. <Town>Nearlondon</Town> has always been a loyal subject of Her Majesty. Although the city of <Town>London</Town> is dear to us, on <RelativeDate>Thursday June 2</RelativeDate> we will change the name of <Address>London Street</Address> to <Address>Queen Elizabeth Street</Address>.</p>
<p>Everyone will fondly remember 2022!</p>
<p>God Save the Queen!</p>
</body>
</xml>
```

The result of annotating the element *<body>* was the same as before. However, we see that *Nearlondon* and *Wednesday June 1, 2022* appearing in the header were annotated as well although they are not part of the text. We would like to avoid this, because we do not want to annotate the header.

We will use the cascades developed before to which we will add some new graphs. We will copy two cascades (*analysis.csc* and *synthesis.csc*) from the folder:

MyUnitex\English\CasSys\UnitexGettingStarted\PlainTextEntities

and paste them in the folder:

MyUnitex\English\CasSys\UnitexGettingStarted\XmlTextEntities

2.1 Analysis cascade

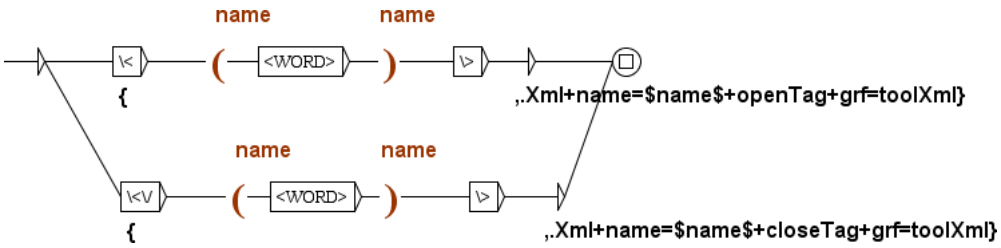
We will open the text *jubileeXmlText.xml.txt* in Unitex (*Text/Open*) and answer *No* to the question *Do you want to preprocess the text*.

We will build two new graphs, saved in the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Entities\Analysis

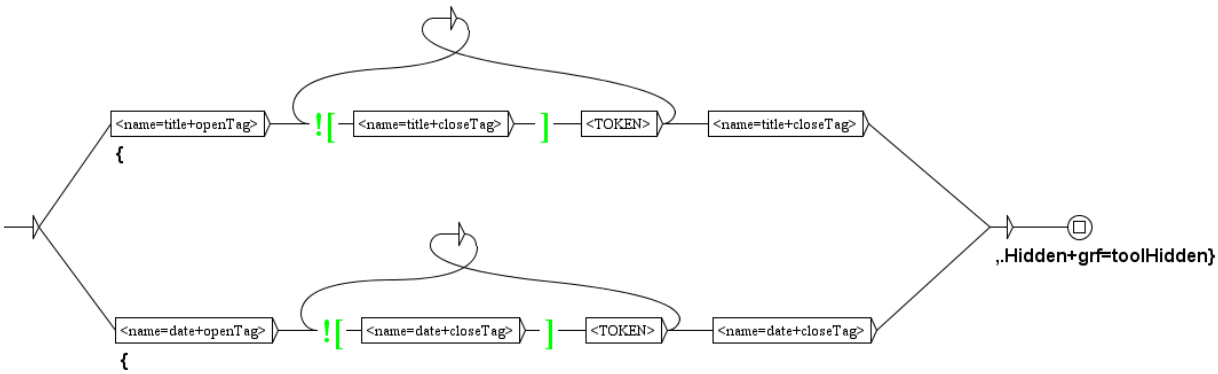
2.1.1 *The* toolXml.grf *graph*

We will assume that the name in an XML tag is a sequence of letters that can be recognized in Unitex with the lexical mask *<WORD>*.[81] The *toolXml.grf* graph recognizes XML tags, transforms them into brace-enclosed lexical tags, with *Xml* in the part-of-speech (POS) field and with as features the name of the recognized tag and the indication whether it was an opening or closing tag.[82]



2.1.2 *The* toolHidden.grf *graph*

The role of the *toolHidden.grf* graph is to hide and protect the XML elements that can be found in the text header.[83] By doing this, we disable the annotation of entities in the content of hidden elements. This graph transforms the recognized content, together with the opening and closing tags, into a lexical tag categorized as *Hidden* in the POS field.



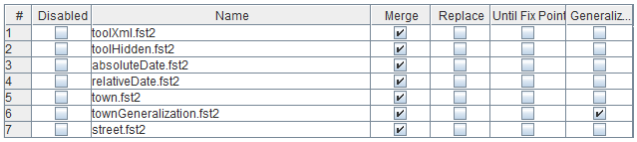
We will put these two new graphs at the beginning of the *analysis.csc* cascade, which will now consist of seven graphs. These two graphs will work in the *Merge* mode like the rest of the analysis graphs. Do not forget to apply the *analysis.csc* cascade to the original *jubileeXmlText.xml.snt* file.

---

[81] Of course, XML names can be more complex, but we want to keep our example simple. It would be possible to call in this box a subgraph that describes possible names more precisely.

[82] This graph is designed for the simple XML document that we are using in our examples. A general graph that would recognize all possible XML tags, including their attributes, would be more complex.

[83] This graph only works if the text to hide is not too long. Otherwise, see Section 4.4.1, page 98 (for confident users).



#	Disabled	Name	Merge	Replace	Until Fix Point	Generaliz...
1	☐	toolXml.fst2	☑	☐	☐	☐
2	☐	toolHidden.fst2	☑	☐	☐	☐
3	☐	absoluteDate.fst2	☑	☐	☐	☐
4	☐	relativeDate.fst2	☑	☐	☐	☐
5	☐	town.fst2	☑	☐	☐	☐
6	☐	townGeneralization.fst2	☑	☐	☐	☑
7	☐	street.fst2	☑	☐	☐	☐

The result of the first graph in the cascade (*toolXml.grf*), *jubileeXmlText.xml_1_0.snt* file, is recorded in the folder:

MyUnitex\English\Corpus\UnitexGettingStarted\JubileeXmlText\jubileeXmlText.xml_csc

We can see that all XML tags in the text have been transformed into lexical tags with *Xml* in the POS field (only the first line is displayed here).

```
{<xml>,.Xml+name=xml+openTag+grf=toolXml}
```

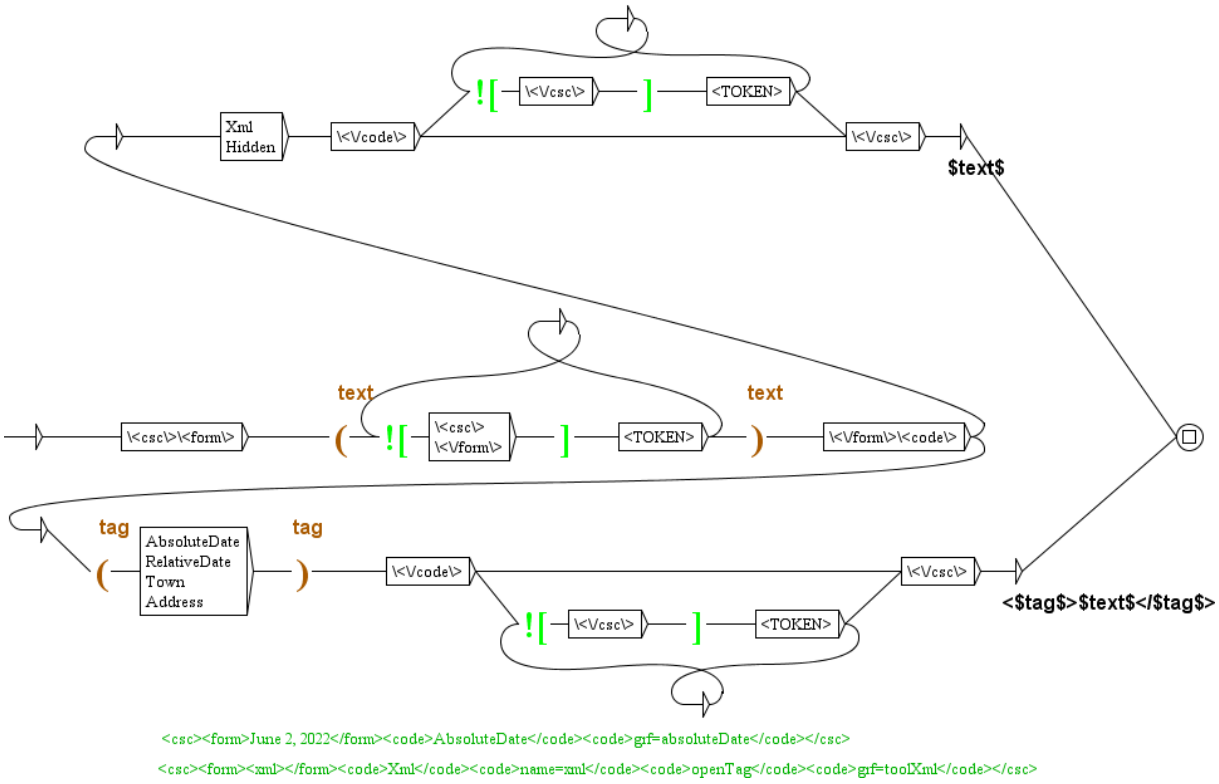
The result of the second graph is recorded in the *jubileeXmlText.xml_2_0.snt* file. We can see that two tag-enclosed XML elements, *<title>/</title>* and *<date>/</date>*, which appear in the header, have been transformed into lexical tags *Hidden* (only the second and third lines are displayed here). Note that in order to preserve the literal meaning of the special characters (period, comma, parentheses and so on), they are protected by an escape character backslash when they appear inside the first field of a lexical tag. These escape characters are inserted before the special characters only if the application of the graph has been launched as an item in a cascade.

```
{\{<title>\,\.Xml\+name=title\+openTag\+grf=toolXml\}The municipal newspaper of the city of
Nearlondon\{</title>\,\.Xml\+name=title\+closeTag\+grf=toolXml\},.Hidden+grf=toolHidden}
{\{<date>\,\.Xml\+name=date\+openTag\+grf=toolXml\}Posted on Wednesday June 1\,
2022\{</date>\,\.Xml\+name=date\+closeTag\+grf=toolXml\},.Hidden+grf=toolHidden}
```

2.2 Synthesis cascade

Now, we will open the text *jubileeXmlText.xml_csc.txt* in Unitex (*Text/Open*) and answer *No* to the question *Do you want to preprocess the text*.

We will open (*Graphs/Open* menu or *Graphs/Open Recent* menu) the existing *tag.grf* graph that lists all annotation categories that must remain (the output is *<tag>$text$</$tag>*). We will modify this graph by adding a path where we list the annotation categories to be removed (*Xml* and *Hidden*); the output is *$text$*.



We will save the *tag.grf* graph, compile it and launch the *synthesis.csc* cascade. The resulting *jubileeXmlText.xml_csc_csc.txt* file will contain exactly what we expected.

```
<xml>
<title>The municipal newspaper of the city of Nearlondon</title>
<date>Posted on Wednesday June 1, 2022</date>
<body>
<p>This week we celebrate the Platinum Jubilee of the Queen of England: seventy years of reign!
The party will start on <AbsoluteDate>June 2, 2022</AbsoluteDate> and end on
<RelativeDate>June 5</RelativeDate>. The Queen ascended the British throne on
<AbsoluteDate>February 6, 1952</AbsoluteDate>.</p>
<p>Our small town of <Town>Nearlondon</Town> will also celebrate this event.
<Town>Nearlondon</Town> has always been a loyal subject of Her Majesty. Although the city of
<Town>London</Town> is dear to us, on <RelativeDate>Thursday June 2</RelativeDate> we will
change the name of <Address>London Street</Address> to <Address>Queen Elizabeth
Street</Address>.</p>
<p>Everyone will fondly remember 2022!</p>
<p>God Save the Queen!</p>
</body>
</xml>
```

3 Third example: measure recognition

We will now go back to the text *DombeyAndSon* in the folder:

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon

with the idea of recognizing measures in it. We will do that in two phases: first, we will recognize numbers and then units. In order to keep things simple, we will produce graphs that recognize only units that occur in the novel: currencies (pence, sixpence, shilling and pound), duration (minute, hour, day, week, month and year), lengths (mile, foot and yard), speed (mile an hour), temperature (degree) and weight (ton).

Sixpence is the name of an old coin. We will check its presence in the Unitex distribution dictionary. We will open the *Dela/Lookup…* menu, select the *dela-en-public.bin* dictionary and type *sixpence* in the *Word* field. Unitex displays:

sixpence,.N:s:p

Unitex considers *sixpence* as both singular and plural, but Charles Dickens uses a plural form with an *s*. Sure enough, if we use the *Dela/Lookup…* menu again for the form *sixpences*, we will not find it.[84]

Therefore, in the graph of Section 3.1.3, page 90, we cannot use the *<sixpence>* mask, but *sixpence+sixpences*.[85]

3.1 Analysis cascade

Most probably, we will be able to use the *Text/Open Recent* menu and open directly *DombeyAndSon.snt*. If this file is not in the offered list, we will use the *Text/Open* menu, find the *DombeyAndSon* folder, and choose the *All Files* option in the *Files of Type* drop-down menu. If *DombeyAndSon.snt* exists in it, we will open it because it means that it has already been preprocessed. Otherwise, we will open *DombeyAndSon.txt* and answer *Yes* to the question *Do you want to preprocess the text*. A window will open in which we will remove the first two checkmarks, leaving just the last one *Apply All default Dictionaries*.

3.1.1 The number.grf *graph*

We will copy the *NB2-999999.grf* graph and all the subgraphs (**-subgraph.grf*) from the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Numbers

and paste them in the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Measures\Analysis

Then we will rename *NB2-999999.grf* as *number.grf*. We should remember that this graph recognizes all numbers between 2 and 999 999.[86]

We must make a final modification to the *number.grf* graph.

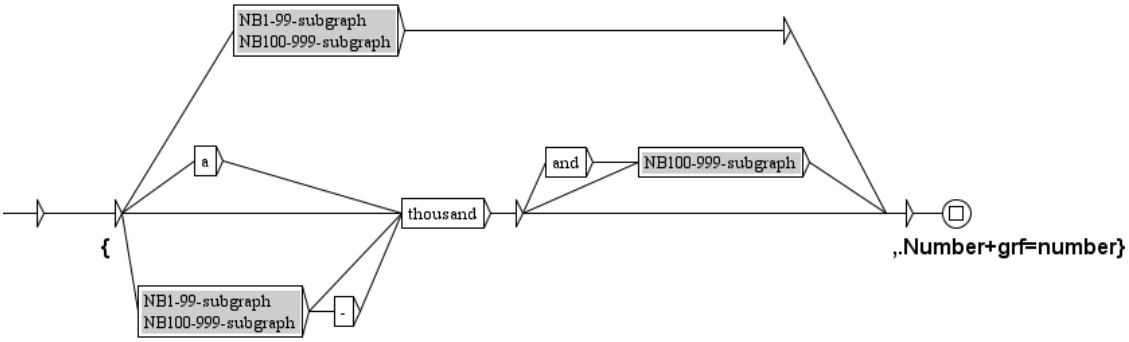
1. We replace the call to the *NB2-999* graph by a call to the two subgraphs, *NB1-99-subgraph* and *NB100-999-subgraph* (these two subgraphs produce no output and the first one recognizes number one).
2. We modify the two paths leading to and from this box (because the subgraphs don't produce output).
3. We remove the box *:NB100-999-subgraph* after the box *thousand* and replace the subgraph *NB1-99-subgraph* by the subgraph *NB100-999-subgraph*.
4. Since we are going to use it in a cascade, we will change its output to produce lexical tags belonging to the category *Number*: the opening brace will replace the opening

[84] The Merriam-Webster Dictionary records that this noun has two forms of plural: *sixpence* and *sixpences* (https://www.merriam-webster.com/dictionary/sixpence). In the Unitex distribution dictionary only the first possibility is recorded.

[85] The *<sixpence>* mask only matches the *sixpence* form present in the dictionary.

[86] See Chapter 3, Section 2.4.2, page 62.

XML tag *<number>* and the rest of the lexical tag including the closing brace will replace the closing XML tag *</number>*.



This graph recognizes all numbers between 1 and 999 999. We will compile it.

3.1.2 Cascade

We will now produce a new analysis cascade that will initially contain only one graph, *number.grf*. Remember that in order to add it to the cascade this graph must be compiled. We will save the cascade in the folder

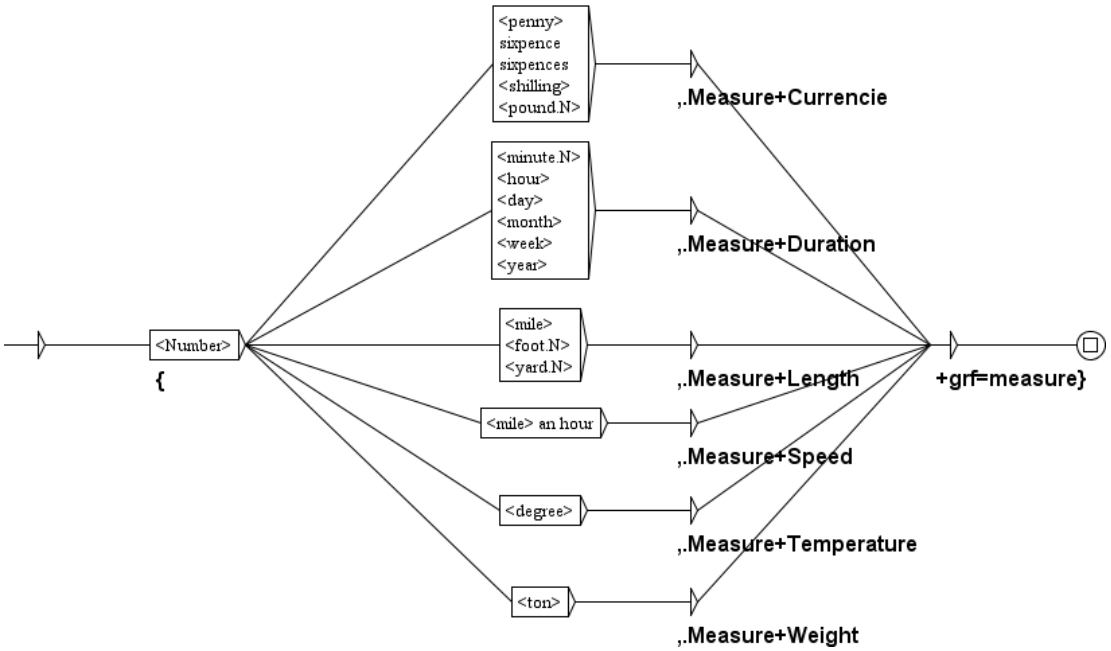
MyUnitex\English\CasSys\UnitexGettingStarted\Measures

and name it *analysis.csc*. The graph will work in the *Merge* mode.

When we launch this cascade, we will obtain 1 652 lines: the same 736 matches that we obtained with the *NB2-999999.grf* graph (the only difference is the output, the matches are in lexical tags now) and 916 matches for the ambiguous word *one*, which we previously omitted.

3.1.3 The measure.grf *graph*

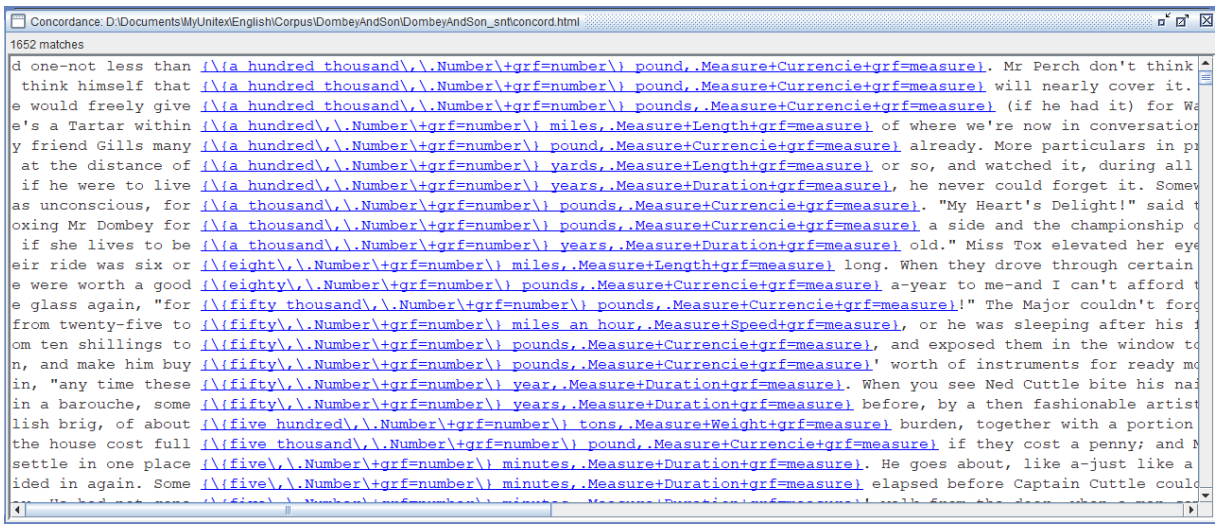
We will now build a graph that recognizes and classifies measurement entities, that is numbers followed by measurement units. As explained before, it will recognize only those units that occur in the novel. The graph classifies the recognized entities as *Measure* and sub-classifies them as *Currencies*, *Durations*, *Lengths*, *Speed*, *Temperature* and *Weight*. Note that we have added the Part-Of-Speech code *N* in the lexical masks for the units when corresponding lemmas are ambiguous with verbs or adjectives.



We will now compile this graph and add it as the second graph in the analysis cascade. This graph will also be applied in the *Merge* mode.

#	Disabled	Name	Merge	Replace	Until Fix Point	Generaliz...
1	☐	number.fst2	☑	☐	☐	☐
2	☐	measure.fst2	☑	☐	☐	☐

After launching this cascade, the result will be again a list of 1 652 concordance lines, some of which contain measurement entities and the other just numbers or the word *one* (tagged as a number as well).



3.2 Synthesis cascade

Our next step is to produce the cascade that will transform lexical masks into XML tags. We will open the text *DombeyAndSon_csc.txt* from the folder:

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon

without preprocessing.

3.2.1 The tag.grf graph

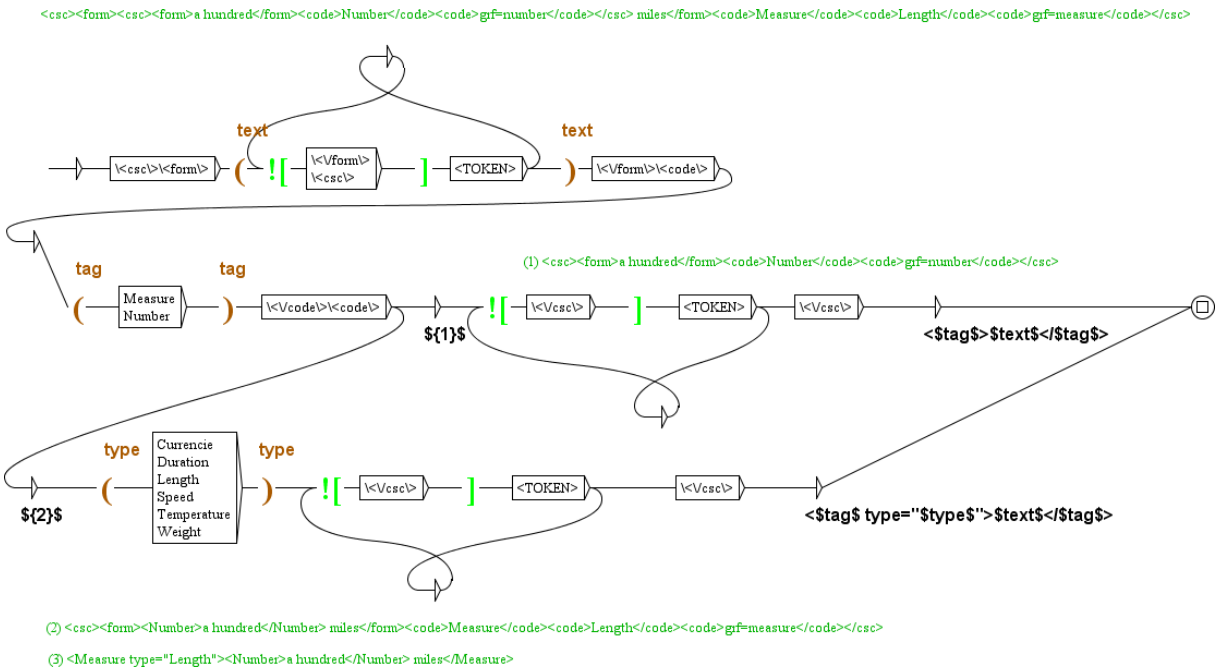
We will create a *tag.grf* graph that will be similar to the graph with the same name that we produced for the named entity recognition problem. So, we can copy this graph from the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Entities\Synthesis

and paste it in the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Measures\Synthesis

We will delete the upper part of the previous *tag.grf* (the part used to recognize *Hidden* and *Xml* codes) and modify the rest. We will replace the old names of the tags (*Measure* and *Number* will replace *AbsoluteDate*, *RelativeDate*, *Town* and *Street*) and add the recognition of subclasses that will be values of the attribute *type*. Note that subclasses are optional (class *Measure* have them, while class *Number* does not) and that these codes are also inside tags *<code>* and *</code>* produced by the *analysis.csc* cascade. Two paths that lead to the ending node after the recognition of codes for classes (*Measure* and *Number*) need to have weights, otherwise the recognition may just randomly skip everything up to the closing tag of a matched *</csc>*.[87] That is why the path for the recognition of subclasses has the higher priority *2*.



3.2.2 Cascade

We will now create a new cascade containing only the *tag.grf* graph that we have just built and compiled. This graph will work in the *Replace* mode and it will iterate until it comes to a fixed point (no change made). Therefore, we must check these two options. We will save this cascade in the folder:

MyUnitex\English\CasSys\UnitexGettingStarted\Measures

[87] See the Chapter 3, Section 3.5.2, page 69.

giving it the name *synthesis.csc*.

After launching this cascade, we will get the resulting *DombeyAndSon_csc_csc.txt* file in the folder:

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon

Here is an excerpt from this file in which we see that numbers are tagged both by themselves and inside *<Measure></Measure>* tags. In addition, *Measure* tags have the attribute *type* with a value indicating the type of measurement unit.

We will take the following sentence as an example:

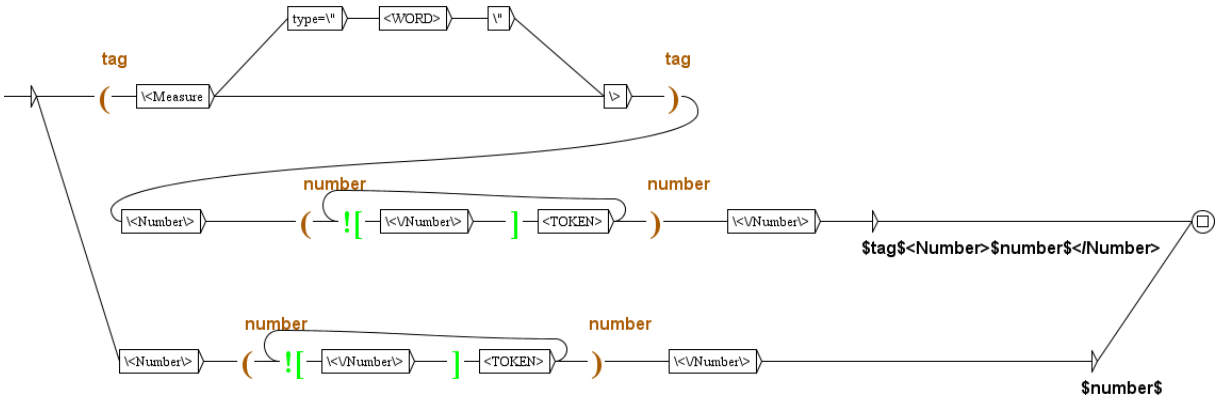
```
Dombey was about <csc><form>eight</form><code>Number</code><code>grf=number</code>
</csc>-and-<csc><form><csc><form>forty</form><code>Number</code>
<code>grf=number</code></csc> years</form><code>Measure</code>
<code>Duration</code><code>grf=measure</code></csc> of age.
```

After applying the cascade, this sentence becomes:

```
Dombey was about <Number>eight</Number>-and-<Measure type="Duration"><Number>forty
</Number> years</Measure> of age.
```

3.2.3 *The* numberDeletion.grf *graph*

We would like to annotate only measurement entities and not numbers unless they are part of them. We will thus create a *numberDeletion.grf* graph that removes *Number* tags found outside *Measure* tags.



We will compile this graph and add it as the second in the *synthesis.csc* cascade to be used in the *Replace* mode.

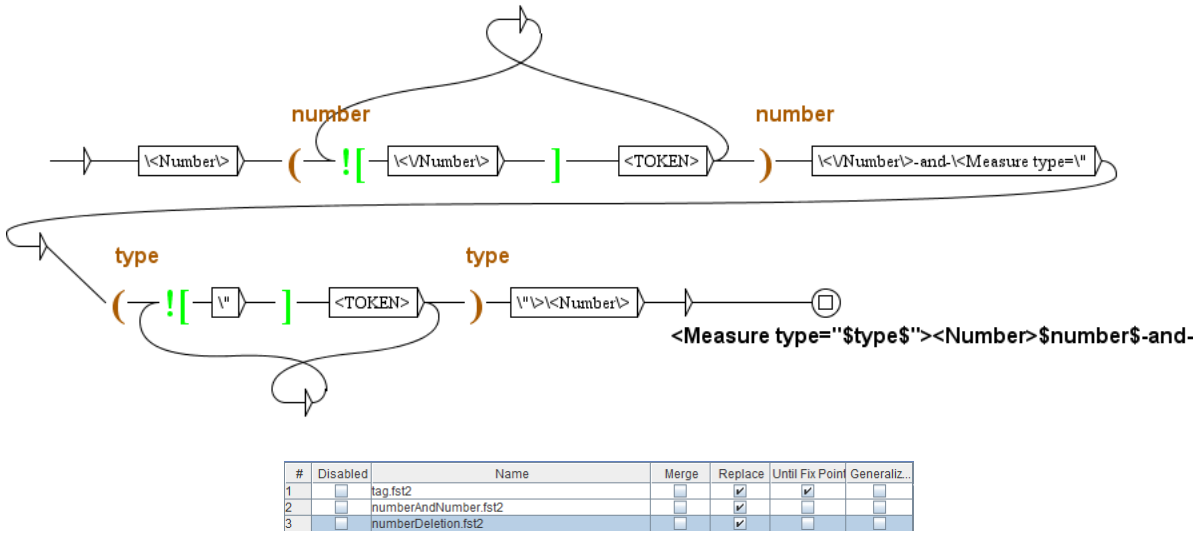
This cascade applied to the text *DombeyAndSon_csc.txt* gives us the desired result. Now, the annotation of the first number is deleted.

```
Dombey was about eight-and-<Measure type="Duration"><Number>forty</Number> years
</Measure> of age.
```

However, we see that a *number-and-number* sequence actually represents one number, while only the second number has been included in the Measure entity. We need to correct this.

3.2.4 *The* numberAndNumber.grf *graph*

We will just add a new graph, *numberAndNumber.grf*, between the two previous graphs of the *synthesis* cascade, to include the whole sequence in *Measure* tags. We have to take care to recognize the type of the *Measure* entity.



#	Disabled	Name	Merge	Replace	Until Fix Point	Generaliz...
1		tag.fst2		✔	✔	
2		numberAndNumber.fst2		✔		
3		numberDeletion.fst2		✔		

The previous example is now correctly annotated. The text obtained as a result of the *tag.fst2* graph:

> Dombey was about <Number>eight</Number>-and-<Measure type="Date"><Number>forty</Number> years</Measure> of age.

becomes now:

> Dombey was about <Measure type="Duration"><Number>eight-and-forty</Number> years </Measure> of age.

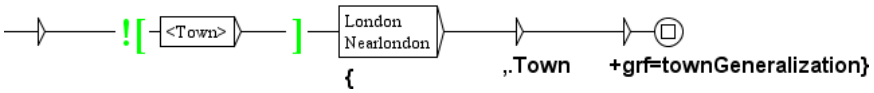
4 Additional possibilities (only for confident users)

4.1 How does a generalization graph work?

The generalization graph developed in Section 1.1.5, page 75, enables the automatic creation of a graph which will replace it in the cascade and which is constructed on the basis of occurrences found in a text. This graph, having the same name *townGeneralization.grf*, is in the folder (this graph is the fourth graph in the *analysis* cascade):

MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\jubileePlainText_csc\jubileePlainText_4_0_snt

After rearrangement, this graph looks like this:



As compared to the generalization graph, the automatically generated graph contains an additional box with the list of all words that were previously tagged as *Town* (*London* and *Nearlondon*). The G box is replaced by a negative right context in order to avoid tagging again as *Town* words that were already tagged *Town*.

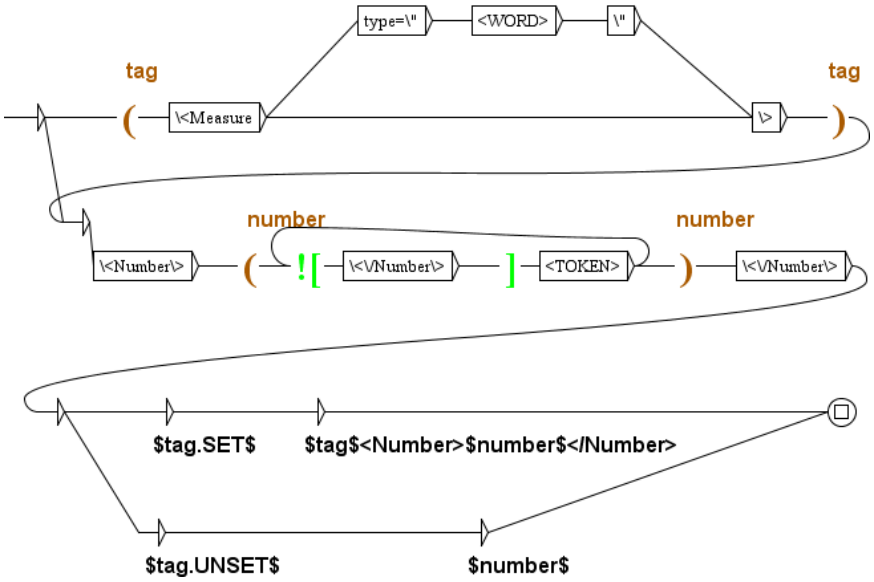
4.2 Testing the variables

In the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Measures\Synthesis

we can observe in the *numberDeletion.grf* graph, Section 3.2.3, page 93, that some part is written twice (recognition of the element tags and the content). In order to avoid this, we could create a subgraph. However, we will instead explore another solution, the possibility to test variables.

We will create a new graph named *numberDeletionWithTest.grf* from the previous *numberDeletion.grf* graph. In this graph we recognize the element only once, but we decide what output will be produced based on the variable *tag* (value of the open tag of a measurement entity with its attribute). In this graph the *$tag.SET$* box is successful only if the *tag* variable is not empty, that is, the *Number* tag is inside a *Measure* tag. On the other hand, the path through the *$tag.UNSET$* box is successful only if the *tag* variable is empty, that is, the *Number* tag is outside a *Measure* tag.



If we compile this graph and replace the second graph in the *synthesis.csc* cascade with it, the cascade will produce the same result as before when applied to the text *DombeyAndSon_csc.txt*. To do the substitution, we can use the *Disabled* column that enables switching on and off some graphs in a cascade.

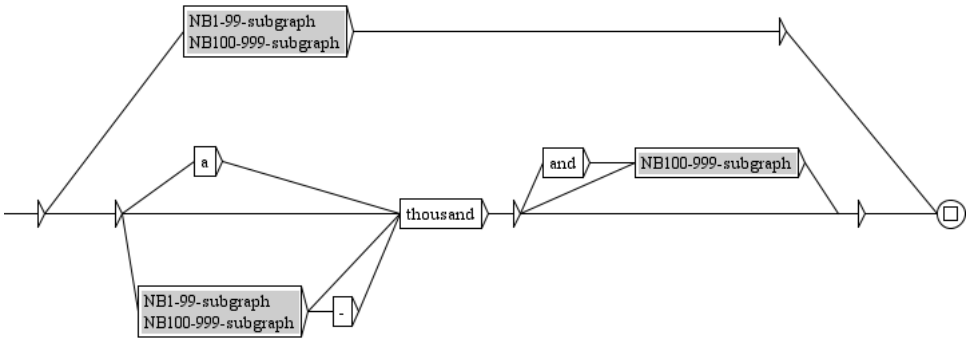
#	Disabled	Name	Merge	Replace	Until Fix Point	Generaliz...
1	☐	tag.fst2	☐	☑	☑	☐
2	☐	numberAndNumber.fst2	☐	☑	☐	☐
	☑	~~numberDeletion.fst2~~	☐	☑	☐	☐
3	☐	numberDeletionWithTest.fst2	☐	☑	☐	☐

4.3 Reuse the same graph

Redears can recall that we have constructed the *number.grf* graph from the *NB2-999999.grf* graph (see Section 3.1.1, page 89). We will now propose another solution, a single graph that can be used to annotate both with XML tags and with lexical tags. We will copy the *number.grf* graph and paste it in the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Numbers

We will name it *numberXmlOrLt.grf* and we will remove the insertion of an opening tag *<E>/{* and replace it with an empty box (*<E>*); and similarly, we will remove the insertion of a closing tag *<E>/,.Number+grf=number}* and replace it by an empty box (*<E>*).



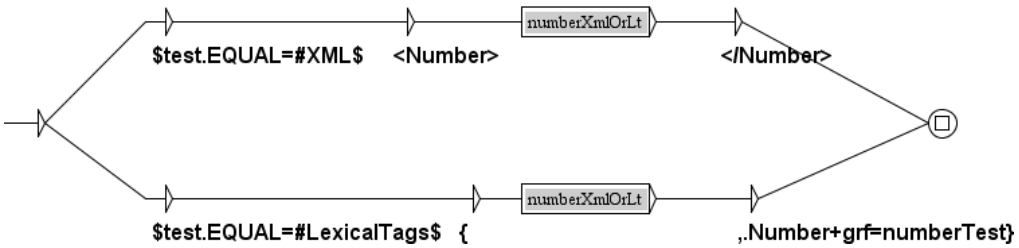
We will now describe how to use this graph for both types of annotation: We will use output variables.

4.3.1 Output variable

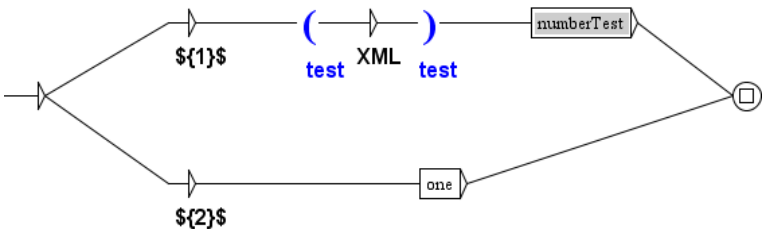
We will create a new graph, *numberTest.grf*, and place it in the same folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Numbers

In this graph, the call to the *numberXmlOrLt.grf* graph follows a test on an output variable *$test$*.

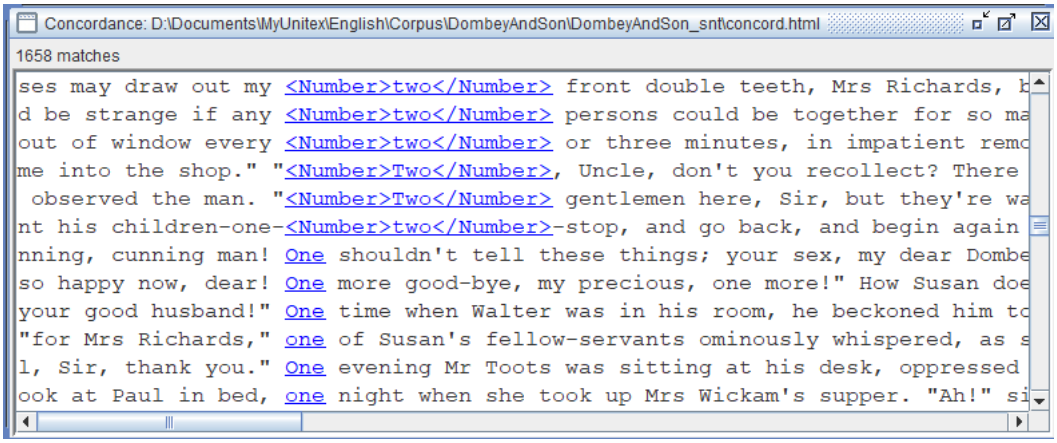


A new graph named *numberXml.grf* will replace the *NB2-999999.grf* graph created in Chapter 3, Section 2.4.2, page 62. In this graph, the *<E>/XML* empty box is surrounded by special blue output parentheses, which you can insert by clicking the *Surround box selection with an output variable* button. The value of the output variable *$test$* is the output of the path enclosed in the blue parentheses, here *XML*.[88] To tag only numbers greater than 1, as we saw in Chapter 3, Section 3.4.2, page 67, we will use weights.

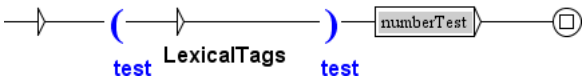


If we apply this graph to the novel text, we will obtain 1658 concordance lines, but only 736 lines with <Number></Number> tags, as in the precedent chapter.

[88] See Unitex user manual, Section 6.8.



In the same way, a new graph named *numberLt.grf* will replace the *number.grf* graph, created in Section 3.1.1, page 89.

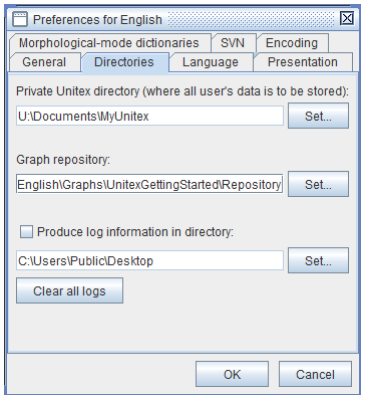


4.3.2 Graph repository

This solution can be improved by using a graph repository. For this, let us create a new folder, *Repository*, inside the folder:[89]

MyUnitex\English\Graphs\UnitexGettingStarted

In order to make this folder our *graph repository*, we have to open the *Info/Preferences/Directories* menu, click the *Set* button next to the *Graph repository* option in order to choose this specific folder, and finish the action by clicking the *OK* button.[90]



We will copy the *numberXmlOrLt.grf* graph and its three subgraphs[91] from the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Numbers

and paste them in the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Repository

---

[89] This folder is not necessarily located in the *MyUnitex* folder. It can be common to several projects, each corresponding to a particular private working directory. See Chapter 2, Section 1.1.4, page 11.

[90] This action adds a line to the configuration file located in the folder:

MyUnitex\English

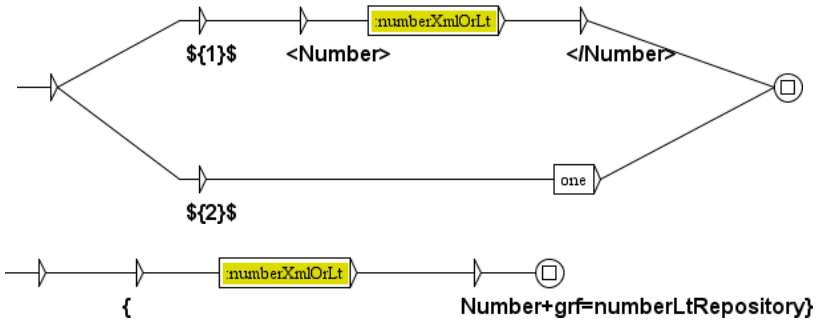
[91] *NB1-9-subgraph.grf*, *NB1-99-subgraph.grf* and *NB100-999-subgraph.grf*.

A call to a subgraph that is in the repository is obtained by preceding its name with two colons; the box will have yellow background. It doesn't matter in which folder the main graph is, since it calls a subgraph that is in the repository.

We will create two new graphs in the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Numbers

The first graph, named *numberXmlRepository.grf*, annotates numbers with *<Number></Number>* tags, while the second graph named *numberLtRepository.grf* transforms numbers in *lexical tags*.



4.4 Hiding or deleting part of a text

Before starting this section

Go to the folder: *MyUnitex\English\Corpus\UnitexGettingStarted*
Create a new folder named (without space): *JubileeLoremIpsum*.
Go to the folder:*MyUnitex\English\Corpus\UnitexGettingStarted\JubileeLoremIpsum*
Download the *JubileeLoremIpsum.xml* file.

4.4.1 Hiding part of a text

As mentioned in Section 2.1.2, page 86, the *toolHidden.grf* graph causes an error if the text to be hidden is too long. Actually, a Unitex graph analyzes a text by traversing all the possible paths (between the initial box and the final box) before deciding which path to choose. However, the maximum length of this run is set at 1,000 tokens. This value can be increased by using a script (see Chapter 7, page 149), but there is still a limit.

We will use a text that we prepared specifically for this exercise, *JubileeLoremIpsum.xml*. You will find it in the folder:

MyUnitex\English\Corpus\UnitexGettingStarted\JubileeLoremIpsum

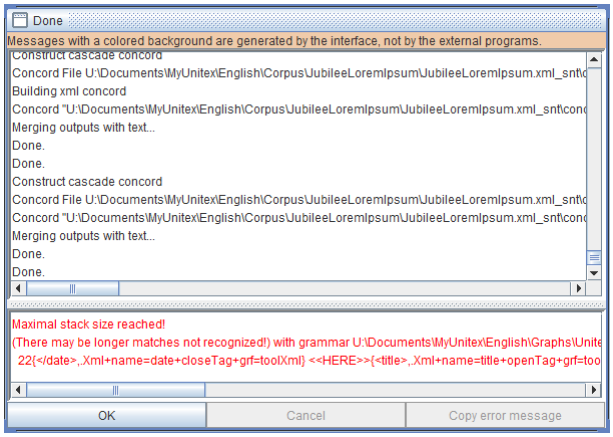
This file contains 596 sequences of letters and 750 other tokens between tags *<title></title>*.

As we explained before, we will rename this file as *JubileeLoremIpsum.xml.txt*, open it in Unitex and answer *Yes* to the question *Do you want to preprocess the text*. A window will open in which we will remove the first two checkmarks, leaving just the last one *Apply All default Dictionaries*.

Now we will open the *Text/Apply CasSys Cascade…* menu and select the *analysis.csc* file in the folder:

MyUnitex\English\CasSys\UnitexGettingStarted\XmlTextEntities

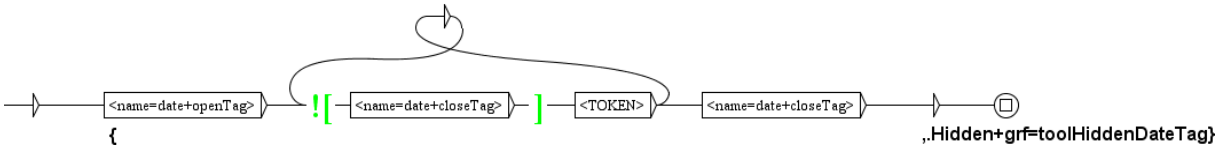
When we click the *Launch* button, we obtain an error message.



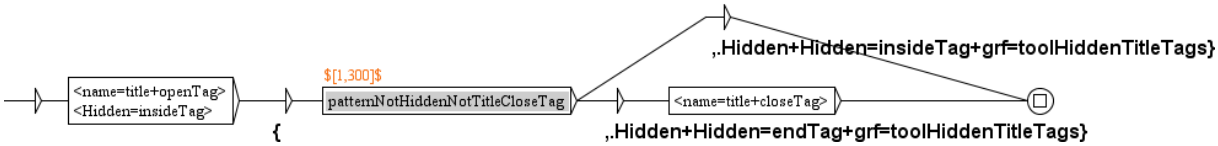
We have to modify the *toolHidden.grf* graph that caused the error. We will open it (*Graphs/Open* menu) in the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Entities\Analysis

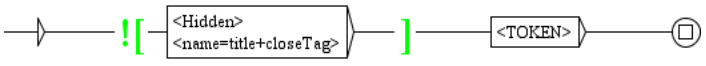
and save it under the new name *toolHiddenDateTag.grf* (we will use the *FSGraph/Save as…* menu). We should not forget to modify the tracing feature as well: *,.Hidden+grf=toolHidden* should be replaced by *,.Hidden+grf=toolHiddenDateTag*. We will delete the first path, leaving only the path that hides the content between the *<date></date>* tags. We will compile this graph.



We will create another new graph, named *toolHiddenTitleTag.grf*, to hide the content between the *<title></title>* tags.



The notation $[1,300]$ above the box that calls this subgraph controls the number of iterations (from one to 300 tokens).[92] The *patternNotHiddenNotTitleCloseTag.grf* subgraph uses the negative right context to recognize a token only if it is neither a hidden text (token classified as "Hidden"), nor the *title* closing tag.[93]

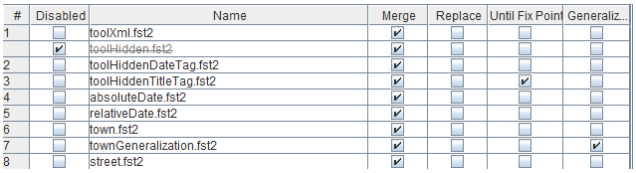


We will modify the analysis cascade in which we will replace the previous *toolHidden.grf* graph with the two graphs: the *toolHiddenDateTag.grf* graph followed by the *toolHiddenTitleTag* graph. The latter graph will be applied until the fixed point is reached.

---

[92] Like all output, these interval boundaries are written after a slash character. The box actually contains *:patternNotHiddenNotTitleCloseTag/$[1,300]$*.

[93] See Chapter 3, Section 1.7.3, page 55, for more details about negative right context.



#	Disabled	Name	Merge	Replace	Until Fix Point	Generaliz...
1	☐	toolXml.fst2	☑	☐	☐	☐
	☑	~~toolHidden.fst2~~	☑	☐	☐	☐
2	☐	toolHiddenDateTag.fst2	☑	☐	☐	☐
3	☐	toolHiddenTitleTag.fst2	☑	☐	☑	☐
4	☐	absoluteDate.fst2	☑	☐	☐	☐
5	☐	relativeDate.fst2	☑	☐	☐	☐
6	☐	town.fst2	☑	☐	☐	☐
7	☐	townGeneralization.fst2	☑	☐	☐	☑
8	☐	street.fst2	☑	☐	☐	☐

We will explain in detail how the *toolHiddenTitleTag.grf* graph works: Unitex detects the opening *title* tag, then it iterates and recognizes 300 tokens at most. If it does not come upon the closing *title* tag, as will be the case with our example text, it tags these 300 tokens as a text to be hidden (*Hidden*) with an additional feature (*Hidden=insideTag*) stating that the end of a text was not yet reached. The graph is launched again. In the second run, the graph detects the new token that we created in the previous step, followed by some regular tokens and after that it loops again over at most 300 tokens.[94] When the closing *title* tag is recognized before reaching the upper limit of iterations 300, the text collected as a title is categorized as *Hidden* with the additional atribute *Hidden=endTag*.

As we saw in Section 1.1.7.2, page 78, the cascade created a new folder:

MyUnitex\English\Corpus\UnitexGettingStarted\JubileeLoremIpsum\JubileeLoremIpsum.xml_csc

containing a new file for each graph or iteration of graph.

The *toolHiddenTitleTag.grf* graph is the third graph of the cascade and it is iterated four times. So the cascade created four new files named:

1. *JubileeLoremIpsum.xml_3_0.snt*,
2. *JubileeLoremIpsum.xml_3_1.snt*,
3. *JubileeLoremIpsum.xml_3_2.snt*,
4. *JubileeLoremIpsum.xml_3_3.snt*.

To check the result, we will open these files. If we search with *name=title* to gain insight only into interesting categories, we will find:

JubileeLoremIpsum.xml_3_0.snt	*{name=title+openTag}{Hidden=insideTag} Aliquam convallis sollicitudin purus…{name=title+closeTag}*
JubileeLoremIpsum.xml_3_1.snt	*{name=title+openTag}{Hidden=insideTag}{Hidden=insideTag} augue. Integer id felis…*
JubileeLoremIpsum.xml_3_2.snt	*{name=title+openTag}{Hidden=insideTag}{Hidden=insideTag}{Hidden=endTag}{name=title+closeTag}*[95]

The *JubileeLoremIpsum.xml_3_2.snt* and *JubileeLoremIpsum.xml_3_3.snt* files are identical, which means that the fixed point has been reached and that the iteration of the *toolHiddenTitleTag* graph stops.

The synthesis cascade (the same as before, see Section 2.2, page 87) will rewrite this text without any internal annotation, whereas, if the hiding graphs had not been used, *Wednesday*

[94] Actually, the 300 tokens surrounded with braces have become a single token of a category *Hidden*.

[95] Which means that at this point our text, inside *<title>/</title>* tags, consists of only 5 tokens.

June 1, 2022 would be tagged as an *AbsoluteDate* inside the *<date></date>* tags and *Nearlondon* would be tagged as a *Town* inside the *<title></title>* tags.[96]

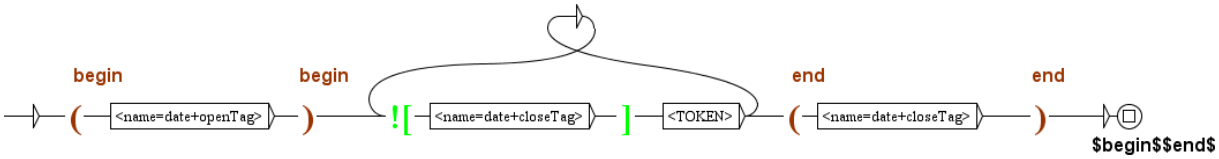
```
<xml>
<date>Posted on Wednesday June 1, 2022</date>
<title><p>From Nearlondon:</p><p>Lorem ipsum…
```

4.4.2 Deleting part of a text

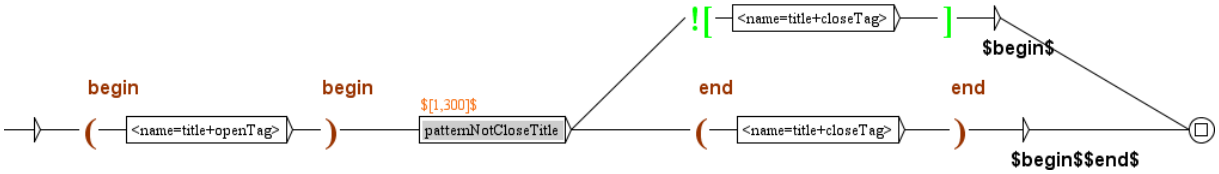
4.4.2.1 The deleting graphs

Instead of hiding the *date* and *title* tags, suppose we want to remove their content.

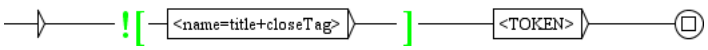
For the *date* tag, presumably with a short content, we can write a graph that replaces everything recognized with <date></date>. We will name this graph *toolEraseDateTag.grf*.



For the *title* tag, which can have a very long content in the number of tokens, we will use a loop limited to 300 tokens. We will name this graph *toolEraseTitleTag.grf*.



This graph uses a subgraph that we will name *patternNotCloseTitle.grf*.



These two graphs are similar to graphs developed in the previous section; they differ only in the produced output. We will build now the cascade. The *toolEraseDateTag* graph and the *toolEraseTitleTag* graph are in *Replace* mode; the *toolEraseTitleTag* graph is repeatedly used until a fixed point.

#	Disabled	Name	Merge	Replace	Until Fix Point	Generaliz...
1	☐	toolXml.fst2	☑	☐	☐	☐
	☑	~~toolHidden.fst2~~	☑	☐	☐	☐
	☑	~~toolHiddenDateTag.fst2~~	☑	☐	☐	☐
2	☐	toolEraseDateTag.fst2	☐	☑	☐	☐
	☑	~~toolHiddenTitleTag.fst2~~	☑	☐	☑	☐
3	☐	toolEraseTitleTag.fst2	☐	☑	☑	☐
4	☐	absoluteDate.fst2	☑	☐	☐	☐
5	☐	relativeDate.fst2	☑	☐	☐	☐
6	☐	town.fst2	☑	☐	☐	☐
7	☐	townGeneralization.fst2	☑	☐	☐	☑
8	☐	street.fst2	☑	☐	☐	☐

We will open the *JubileeLoremIpsum.xml.snt* text and launch the analysis cascade; we will open the *JubileeLoremIpsum.xml_csc.txt* text (without applying dictionaries) and launch the

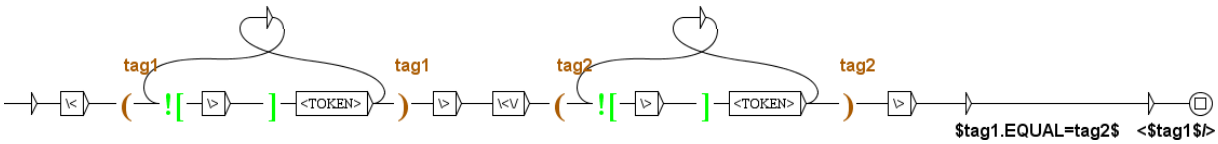
[96] Now that we no longer have to care about the length of the text, we can directly hide the whole preamble, between the *<xml>* and *<body>* tags, instead of hiding inside the *<date>* and *<title>* tags separately.

synthesis cascade. The beginning of the resulting text *JubileeLoremIpsum.xml_csc_csc.txt* is now:

```
<xml>
<date></date>
<title></title>
<body>…
```

4.4.2.2 Comparing the variables

To stick to common XML practice, we will add to the synthesis cascade a new graph that transforms two successive tags, opening and closing, into a self-closing tag. We will name this graph *selfClosing.grf*. We will use two variables *$tag1$* and *$tag2$* and compare them with the formula *$tag1.EQUAL=tag2$*.[97] A self-closing tag is output only if two successive opening and closing tag contain the same element name.



The new graph is added as the third in the synthesis cascade and marked to be used in *Replace* mode.

#	Disabled	Name	Merge	Replace	Until Fix Point	Generaliz...
1	☐	tag.fst2	☐	☑	☑	☐
2	☐	internalDeletion.fst2	☐	☑	☐	☐
3	☐	selfClosing.fst2	☐	☑	☐	☐

After applying the synthesis graph to *JubileeLoremIpsum.xml_csc.snt,* we will get what we wanted.

```
<xml>
<date/>
<title/>
<body>…
```

4.5 Sub-categorization with cascades

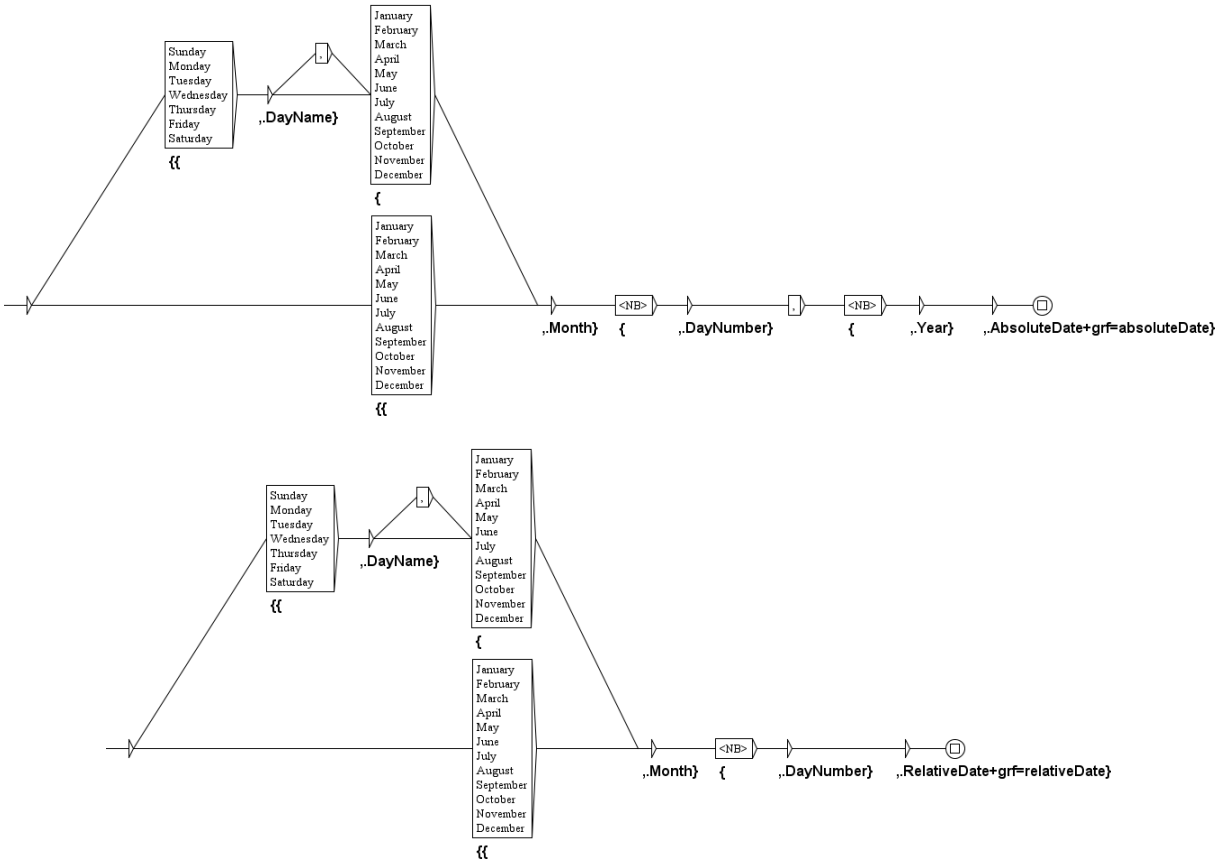
We will work here on the *JubileePlainText* text.

4.5.1 Three modified graphs

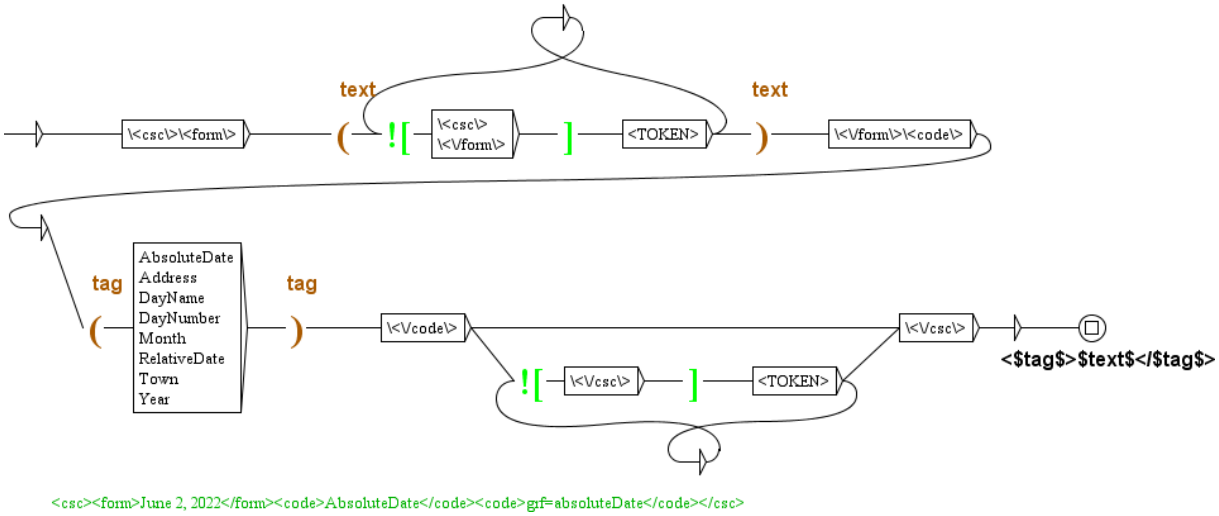
We will take the first two graphs of Section 1.1, page 71, named *absoluteDate.grf* and *relativeDate.grf*; we will modify them by introducing sub-categories. Either DayName, DayNumer (as in graphs) or name of a day, number of a day.[98]

[97] It is also possible to compare a variable to a character string either in a case-sensitive or case-insensitive way. See the Unitex user manual, Section 6.9.2.

[98] Care must be taken that, on each path, a closing brace corresponds to an opening brace. In a complex graph, we can verify this by using the *FSGraph/Tools/Verify braces* option.



Then, we must also modify the *tag* graph of the synthesis cascade, adding the new categories.[99]



When we run the analysis and synthesis cascades with these three new graphs (don't forget to compile the modified graphs!), we get the desired result, a subcategorization of dates.

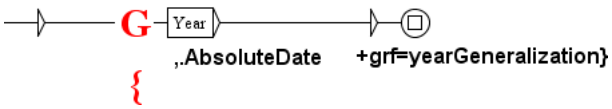
[99] When the categories have been added in the corresponding box, it is possible to sort them in alphabetical order by the *FSGraph/Tools/Sort Node Label* option.

This week we celebrate the Platinum Jubilee of the Queen of England: seventy years of reign! The party will start on <AbsoluteDate><Month>June</Month> <DayNumber>2</DayNumber>, <Year>2022</Year></AbsoluteDate> and end on <RelativeDate><Month>June</Month> <DayNumber>5</DayNumber></RelativeDate>. The Queen ascended the British throne on <AbsoluteDate><Month>February</Month> <DayNumber>6</DayNumber>, <Year>1952</Year></AbsoluteDate>.
Our small town of <Town>Nearlondon</Town> will also celebrate this event.
<Town>Nearlondon</Town> has always been a loyal subject of Her Majesty. Although the city of <Town>London</Town> is dear to us, on <RelativeDate><DayName>Thursday</DayName> <Month>June</Month><DayNumber>2</DayNumber></RelativeDate> we will change the name of <Address>London Street</Address> to <Address>Queen Elizabeth Street</Address>.
Everyone will fondly remember 2022!
God Save the Queen!

4.5.2 Generalization graphs with restrictions

4.5.2.1 Without substitution of a category

A single number should not be recognized as a year, even if it has 4 digits. This could cause too many errors. This is why *2022* in the penultimate sentence *Everyone will fondly remember 2022* was not tagged as a *Year* (nor as a date). We will use a generalization graph to decide that since *2022* has already been recognized as a date, it has to be recognized again as a date.[100] We will name this graph *yearGeneralization.grf.* Note that the subcategory *Year* is in the box, while category *AbsoluteDate* is the output of the box.



We will place this graph in the third position of the *analysis* cascade. Don't forget to check the *Generaliz*... box.

#	Disabled	Name	Merge	Replace	Until Fix Point	Generaliz..
1	☐	absoluteDate.fst2	☑	☐	☐	☐
2	☐	relativeDate.fst2	☑	☐	☐	☐
3	☐	yearGeneralization.fst2	☑	☐	☐	☑
4	☐	town.fst2	☑	☐	☐	☐
5	☐	townGeneralization.fst2	☑	☐	☐	☑
6	☐	street.fst2	☑	☐	☐	☐

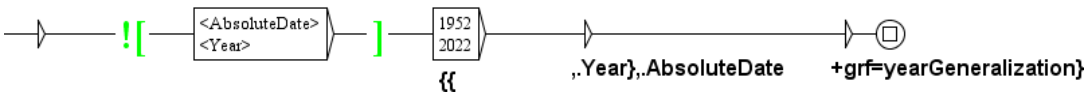
When we run this new analysis cascade and the same synthesis cascade, the desired result is as expected (We display only the penultimate line).

Everyone will fondly remember <AbsoluteDate><Year>2022</Year></AbsoluteDate>!

As explained in Section 4.1, page 94, the generalization graph is automatically replaced in the cascade by a generated graph, having the same name *yearGeneralization.grf*, which is in the folder:

MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\jubileePlainText_csc\jubileePlainText_3_0_snt

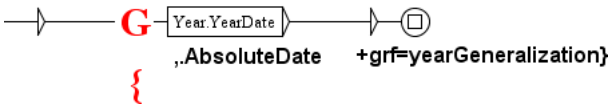
After rearrangement, this graph looks like this:



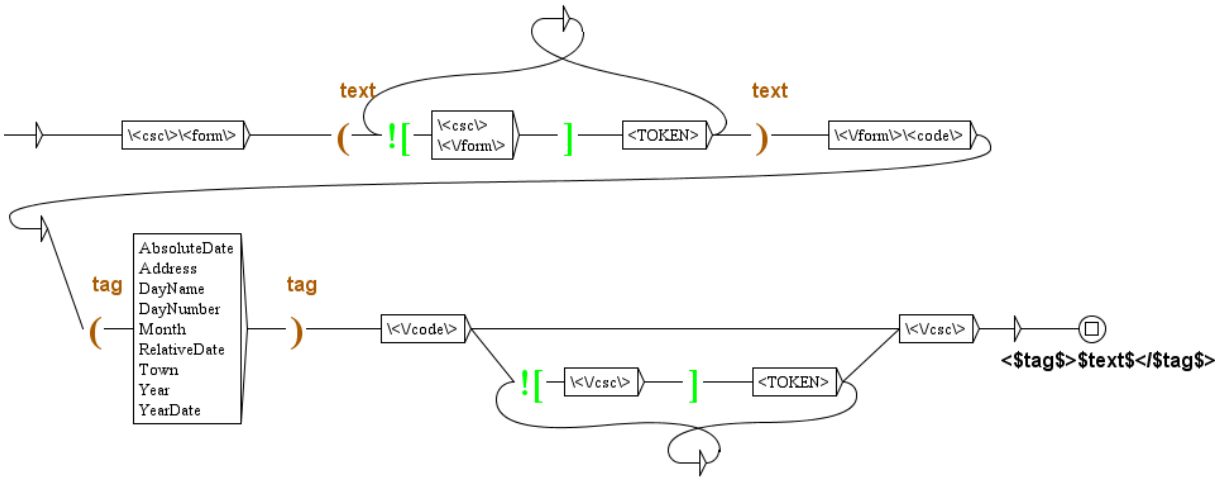
[100] See Section 1.1.5, page 75. Remember that the .grf file must stay in the same folder as the .fst2 file.

4.5.2.2 With substitution of a category

It is also possible to modify the category obtained. For example, if we will want to categorize *YearDate* years recognized in this way, we must write in the box *Year.YearDate/,.AbsoluteDate*. Save this graph.



Of course, this will only work if the *tag.grf* graph of the synthesis cascade is also modified to recognize the new tag *YearDate*.

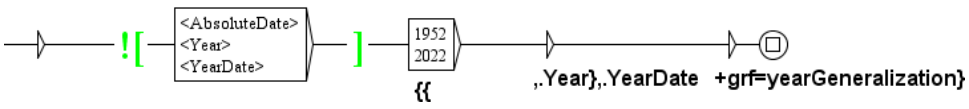


<csc><form>June 2, 2022</form><code>AbsoluteDate</code><code>grf=absoluteDate</code></csc>

The result is now different (don't forget to compile the modified graphs!).

Everyone will fondly remember <YearDate><Year>2022</Year></YearDate>!

After rearrangement, the replaced generalization graph looks like this:



Chapter 5: dictionary creation

Before starting this chapter

Go to the folder: *MyUnitex\English\Inflection*
Select and copy the two files named *Equivalences.txt* and *Morphology.txt*.
Create a new folder named *UnitexGettingStarted* (without space).
Go to the folder: *MyUnitex\English\Inflection\UnitexGettingStarted*
Paste the two files *Equivalences.txt* and *Morphology.txt*.[101]

1 Introduction

We have already discussed the use of the dictionaries from the Unitex distribution in Chapter 2, Section 1.2.4, page 14.

Before we begin, we will open in Unitex our novel, *DombeyAndSon.txt*, located in the folder:

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon

We will answer *No* to the question *Do you want to preprocess the text*.

In this chapter, we will create a new dictionary following these three steps:

1. We will manually create a dictionary of representative (principal) forms (or lemmas) and rules for their inflection;
2. We will create automatically a dictionary of inflected forms, that is, all forms that can occur in texts, like singular and plural forms of nouns;
3. We will automatically create a dictionary in format *.bin* that will appear in the *User resources* field.

For the first step, we will use the Unitex editor. We will open the *File Edition/New File* menu and save the file in the *Dela* folder using the name *myDictionary.dic*. We will make a difference between the inflection of monolexical words and polylexical words.[102]

2 Inflection of monolexical words

2.1 Inflection by simple suffixation

2.1.1 The nouns with s in the plural

We will enter the following three lines in the *myDictionary.dic* file. Note that these entries should not contain spaces. In addition, we will finish the third line (the last line) by clicking the *Enter* key.

[101] These two files define the morphological codes used by Unitex for a specific language. It is not recommended to modify them, except if a user wants to use different morphological codes or wants to create resources for a new language.

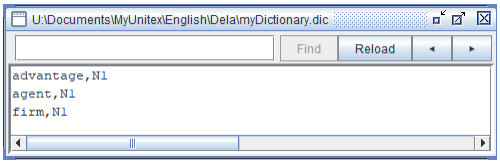
[102] It is also possible to create the inflection rules based on word stems, which is convenient for the Semitic languages; see the Unitex user manual, Section 3.5.4.

```
advantage,N1
agent,N1
firm,N1
```

Each line represents a dictionary entry consisting of a lemma (representative form), followed after a comma by the associated inflectional graph (in the case of our entries *N1.grf*). In the name of this graph, *N* represents the grammatical category (Part-Of-Speech) noun. We will now create this graph and save it in the folder:

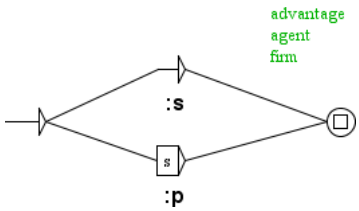
MyUnitex\English\Inflexion\UnitexGettingStarted

We will save and close the *myDictionary.dic* file.



The *N1* graph enables the inflection of nouns for which the plural is obtained by adding an *s* to the singular form. We will use, although it is not mandatory, the same codes used by the dictionary provided with the Unitex distribution *dela-en-public.bin*. One can see these codes in the *Word List* window: *s* is used for the singular forms and *p* for the plural forms. We will add a comment (a box starting with the slash /) with some examples of nouns that can be inflected by this graph.

This inflectional graph is read like this: in the upper part, there is an empty box, signifying that a word (lemma) to which it applies does not change; in the lower part, there is a box containing an *s*, signifying that it is concatenated to the word to which it applies. The output of these boxes are values of grammatical categories associated with inflected forms.



This graph must be compiled, so we will click the *Compile graph* button. We will obtain an error message: *Main graph matches epsilon! Error: the main graph N0 recognize <E>*. The meaning of this message is that we cannot use this graph for search (the *Locate pattern* option) because it recognizes an empty symbol.[103] Since we have not developed this graph for search, we can ignore this message. We will simply click the *OK* button.

We will now launch the inflection program. This program will create a dictionary of all inflected forms, named *myDictionaryflx.dic* (*flx* is automatically added to the file name). In order to do this, we will open the *myDictionary.dic* file in the *DELA* menu, which gives access to programs

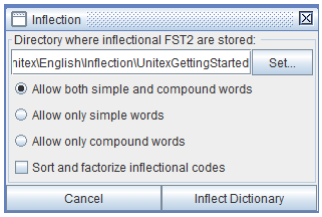
[103] Which would cause an infinite loop and crash the computer.

dealing with dictionaries. Note that now we will not use the *File Edition* menu, because we are not going to create or modify a dictionary of lemmas.[104]

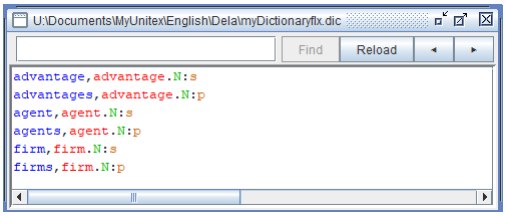
We will now inflect the *myDictionary.dic* dictionary by using the *DELA/Inflect* menu option. First, we will set the folder in which the inflectional graphs are stored. Click the *SET* button, select the folder:[105]

MyUnitex\English\Inflexion\UnitexGettingStarted

and click the *Open* button. Next, click the *Inflect Dictionary* button.



The *myDictionaryflx.dic* dictionary is created and displayed on the screen.



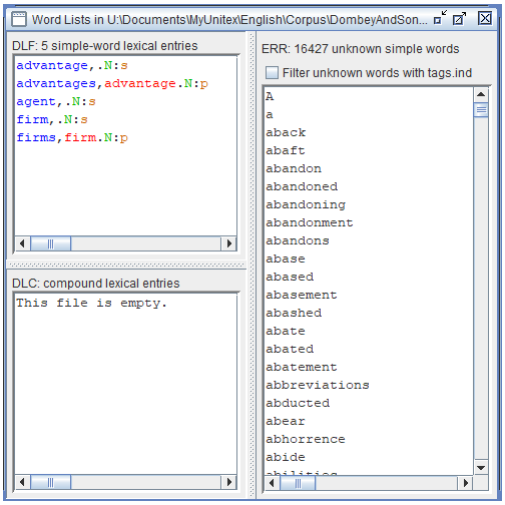
If we want to use this dictionary on our novel (or any other text), one more step is necessary. We must choose the *DELA/Compress into FST* menu option and confirm by clicking *OK*. This will create two more files: *myDictionaryflx.bin* and *myDictionaryflx.inf*.[106] Note that if you want to use this dictionary on some other computer you need to copy both files.

We will now close both windows containing the *myDictionary.dic* and *myDictionaryflx.dic* dictionaries. We want to apply our newly created dictionary to the novel (already open). We will open the *Text/Apply Lexical Resources* menu and click the *Clear* button (in order to cancel all previous selections). We will choose the *myDictionaryflx.bin* file (appearing in the left field) and click the *Apply* button. As a result, the *Word List* window will be displayed.

[104] If you detect an error in your *myDictionary.dic* file, you have to close it in the *DELA* menu and reopen it in the *File Edition* menu.

[105] If you are not going to go through this entire section in one session, remember to set the folder containing the inflectional graphs when you start to work again.

[106] The *myDictionaryflx.bin* file contains words, but it is a binary file and a text editor cannot read it. The *myDictionaryflx.inf* file contains the factorization of tags associated with entries: it is readable.



Since five of our six inflected forms (two for each lemma) appear in this novel, they will be listed in the *DLF* area. *DLC* area is empty, while the *ERR* area lists 16 427 unknown simple words. This is natural since we applied only our tiny dictionary to the text.

We will close the *Word List* window.

2.1.2 Two more examples

We will open the *myDictionary.dic* file again, using the *File Edition/Open…/Dictionaries* menu and answer *OK* to the warning *This is not necessarily the text being processed by Unitex* stating that this file cannot be used for the text analysis.[107] We will add to our dictionary of lemmas some more illustrative examples:

- for the inflection of the nouns *branch*, *bus*, *box* having plural forms produced by adding *es* to the singular form we will use the *N2* graph;
- for the inflection of the verbs *accept*, *connect*, *work* having forms produced by adding *s*, *ed* and *ing* to the infinitive form we will use the *V1* graph.[108]

We will use the same format as before – remember that you should not use spaces and to press the *Enter* key after the last entry.

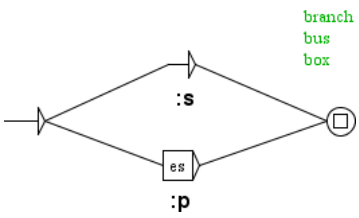
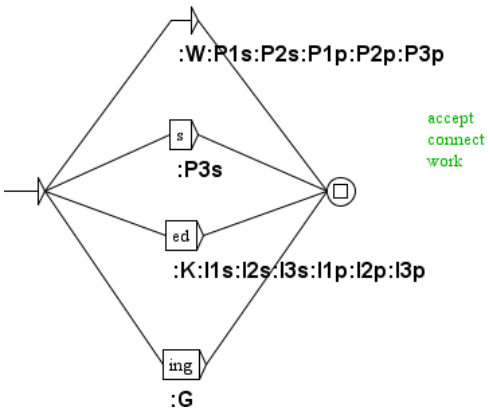
```
advantage,N1
agent,N1
firm,N1
branch,N2
bus,N2
box,N2
accept,V1
connect,V1
work,V1
```

We will save and close the *myDictionary.dic* file.

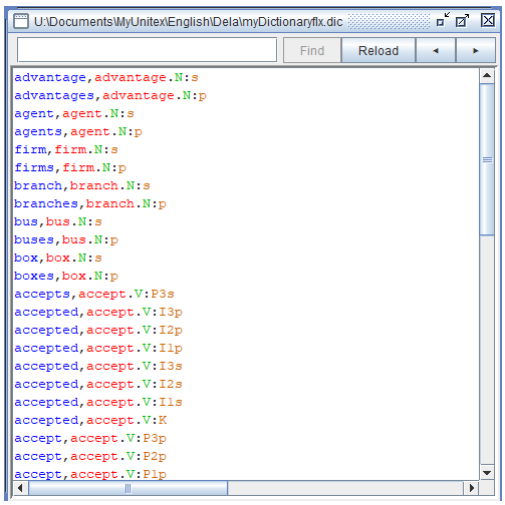
[107] Except if we close it and reopen using the menu *Text/Open*.

[108] We saw in the Chapter 2, Section 1.2.5, page 15, that *G* codes the gerund of a verb. We saw in the Chapter 3, Section 1.2, page 47, that *K* codes the past participle and *W* codes the infinitive of a verb. We also need to know that the code *P* is used for the present tense, *I* for the perfect tense, while *1*, *2* and *3* are codes for persons.

We will create and compile the *N2* et *V1* graphs. We can notice that the *N2* graph is very similar to the previous *N1* graph (the content of the lower box is *es* instead of *s*); so we can facilitate our work by opening *N1.grf*, using the *FSGraphs/Save as…* menu to rename it as *N2.grf*; and then changing the content of this box. Do not forget to replace the comment as well by typing new examples and to compile graphs.

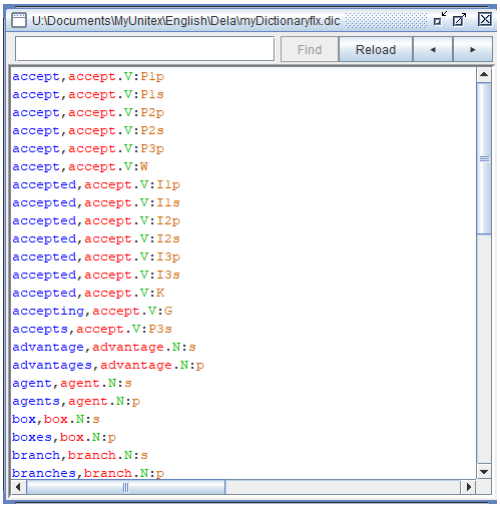
	
N2	V1

We can now open the *myDictionary.dic* dictionary by using the *DELA/Open Recent* menu and inflect it using the *DELA/Inflect* option. The new dictionary named *myDictionaryflx.dic* will be displayed in a separate window with its 57 entries. You can see below the content of this window with some of the produced inflected forms.

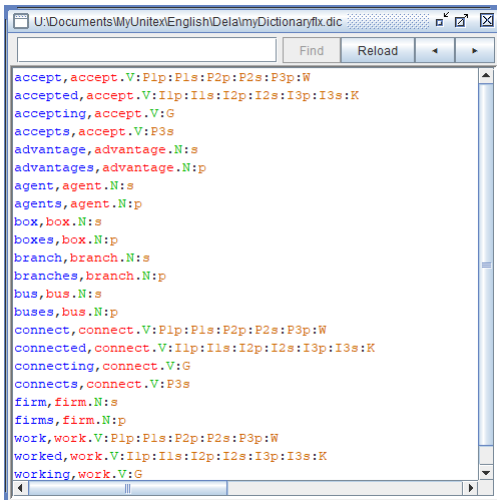


We will verify the inflected forms and, if we find them correct, we can sort them by using the *DELA/Sort Dictionary* option.[109]

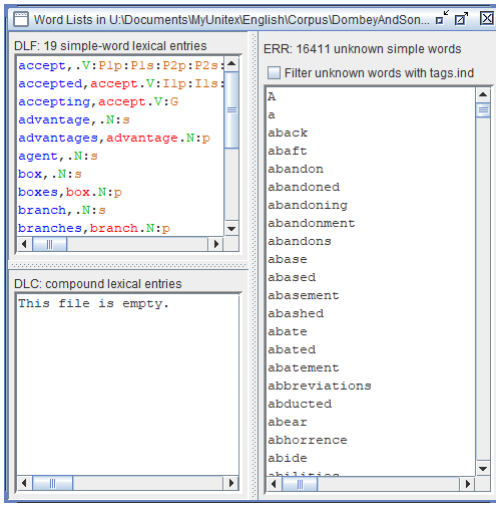
[109] Note that we could have inflected this dictionary after saving the first graph, *N1.grf*. The inflection program issue error notifications for all entries using graphs that do not exist and produces a dictionary of inflected forms only for those entries for which inflectional graphs exist.



When inflecting a dictionary of lemmas, we can check the *Sort and factorize inflectional codes* option in the *DELA/Inflect* menu. A more compact dictionary of inflected forms will be produced. Namely, there will be only one entry for each form of a lemma, followed by a sequence of all possible values of grammatical codes. If we apply this to our *myDictionary.dic* dictionary, we will get the *myDictionaryflx.dic* dictionary of inflected forms consisting of 24 entries, some of which are displayed in the figure below.



In order to make this dictionary functional, we must apply the *DELA/Compress into FST* option. We will now close the windows containing *myDictionary.dic* and *myDictionaryflx.dic.* We will test this new dictionary by applying it to our novel (do not forget to click the *Clear* button in the *Text/ApplyLexical Resources*) menu. In the *DLF* zone of the *Word List* window, 19 entries will be listed (the word forms *accepts*, *bus* and *buses* do not occur in the novel) compared to the five entries obtained by the previous version of the dictionary. Therefore, there are less entries in the *ERR* zone: 16 411. Close the *Word List* window.



2.2 The *L* operator

In order to concatenate the suffix to a substring of a word, that is, not after its last letter (e.g. *compan* from *company*, see below), we use the *L* operator (standing for *Left*) in order to move from right to left starting from the end of a word.

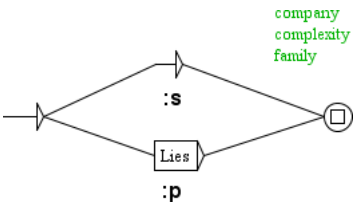
2.2.1 The nouns with final y

Plural forms of the nouns *company*, *complexity*, *family* are produced by replacing the final *y* with the suffix *ies*. Starting from the end of the word, we move one letter to the left (operator *L*) and then we add *ies*. This rule will be written as *Lies* in the box for plural forms in the *N3* graph.[110]

We will now add three more lines to the *myDictionary.dic* file (having now 12 entries). Remember to open it in the Unitex editor by using the *File Edition/Open…/Dictionaries* menu.

```
…
company,N3
complexity,N3
family,N3
```

We will save this dictionary and close the editor. Now we will create the *N3* graph, using as a base the *N1.grf* graph, modifying it and saving with the *Save as…* menu option under the name *N3.grf.* Remember to replace the comment with new examples and compile the graph.



In order to use this new version of the dictionary we have to inflect and compress it. These are the steps to follow: open the *myDictionary.dic* file (the *DELA/Open Recent* menu) and inflect it (the *DELA/Inflect* menu, *Sort and factorize inflectional codes* option). The new version of the *myDictionaryflx.dic* dictionary will be displayed containing 30 lines. If inflected forms

[110] Since the inflection endings are written in lower-case letters, the upper-case letters are reserved for operators.

are correct, we can proceed with the production of the dictionary version that can be used in text analysis: the *DELA/Compress into FST* menu. We will close the windows displaying *myDictionary.dic* and *myDictionaryflx.dic.* We can now apply the new dictionary to our text: the *Text/ApplyLexical Resources* menu (do not forget to click the *Clear* button). The *DLF* zone of the *Word List* window shows that 23 words have been recognized by the *myDictionaryflx.dic* dictionary. We can also see that the word *complexity* does not occur in our text. Before proceeding with new examples, we will close the *Word List* window.

2.2.2 Several more examples

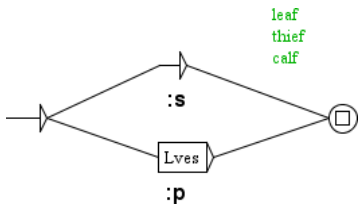
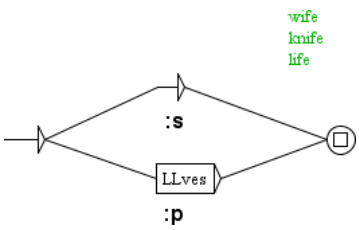
We will again open the *myDictionary.dic* file in the Unitex editor (*File Edition/Open…/Dictionaries*). We will add to the dictionary some more examples of inflections requiring the *L* operator once or several times.

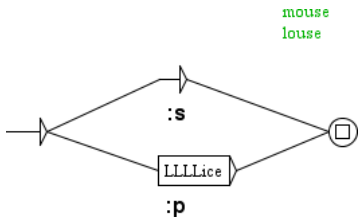
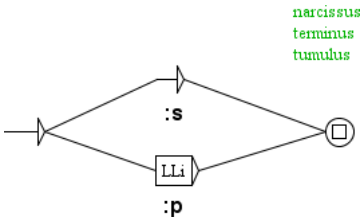
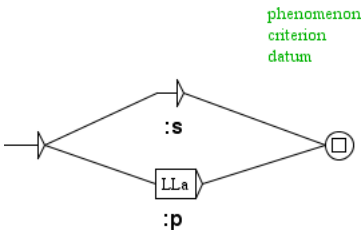
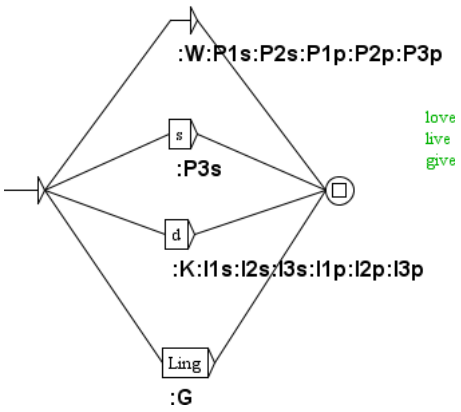
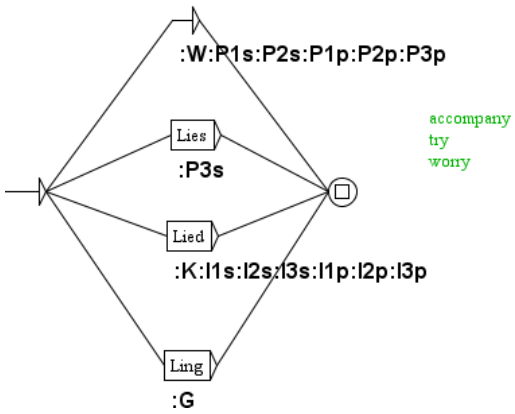
- *leaf*, *thief*, *calf* are nouns in which the final *f* is replaced by *ves* in plural (*N4.grf* graph).
- *wife*, *knife*, *life* are nouns in which the final *fe* is replaced by *ves* in plural (*N5.grf* graph, the *L* operator is applied twice).
- *mouse*, *louse* are nouns in which the final *ouse* is replaced by *ice* in plural (*N6.grf* graph, the *L* operator is applied four times).
- *narcissus*, *terminus*, *tumulus* are nouns in which the final *us* is replaced by *i* in plural (*N7.grf* graph, the *L* operator is applied twice).
- *phenomenon*, *criterion*, *datum* are nouns in which the two final letters are replaced by *a* in plural (*N8.grf* graph, the *L* operator is applied twice).
- *love*, *live*, *give* are verbs for which the final *e* is deleted before adding the ending *ing* (*V2.grf* graph, the *L* operator is applied once for this inflected form).
- *accompany*, *try*, *worry* are verbs for which the final letter *y* is replaced by *i* in all inflected forms (*V3.grf* graph, the *L* operator is applied once for all inflected forms).

We will now add entries for these nouns and verbs into the *myDictionary.dic* dictionary (do not forget to press the *Enter* key after the last entry). This dictionary now has 32 entries.

```
…
leaf,N4
…
worry,V3
```

We will save the dictionary and close the Unitex editor. After that, we will create and compile new graphs for the inflection of nouns and verbs.

	
N4	N5

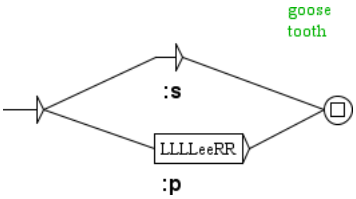
<table>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>N6</td><td>N7</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>N8</td><td>V2</td></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td colspan="2">V3</td></tr>
</table>

As the last step, we will open the *myDictionary.dic* file using the *DELA/Open Recent* option and inflect it choosing *DELA/Inflect*. The new *myDictionaryflx.dic* dictionary contains 82 lines (if the *Sort and factorize inflectional codes* option is on). After compressing this dictionary (*DELA/Compress into FST* menu), we will close both windows (*myDictionary.dic* and *myDictionaryflx.dic*). We will open the *Text/ApplyLexical Resources* menu (do not forget to clear all preselected dictionaries by clicking the *Clear* button and select the *myDictionary.bin* dictionary). The *Word List* now displays 57 recognized words. We will close *Word List*.

2.3 Operators *R*, *C* and *D*

We use the *R* operator (*Right*) in combination with the *L* operator, when we need to substitute part of a lemma while preserving another part on the right.

What do the nouns *goose* and *tooth* have in common? In the plural, the last two letters remain as they are while the string *oo* is replaced by *ee*. It means that starting from the end of a lemma, we have to move to the left by four letters (*LLLL*), then write *ee*, and then move to the right by two letters (*RR*). This procedure is used by the *N9.grf* graph.

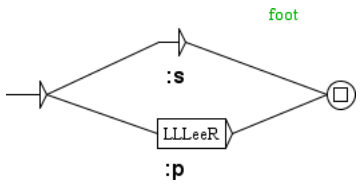
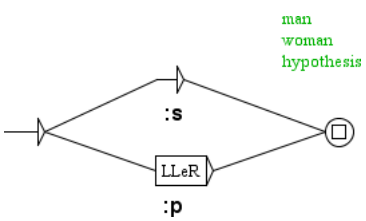


Let us see how it works. The vertical bar indicates the current position in the processed word (the position of the cursor) at each step in the sequence of operations.

|LLLLeeRR	goose|	tooth|
L|LLLeeRR	goos|e	toot|h
LL|LLeeRR	goo|se	too|th
LLL|LeeRR	go|ose	to|oth
LLLL|eeRR	g|oose	t|ooth
LLLLe|eRR	ge|ose	te|oth
LLLLee|RR	gee|se	tee|th
LLLLeeR|R	gees|e	teet|h
LLLLeeRR|	geese|	teeth|

Other examples:

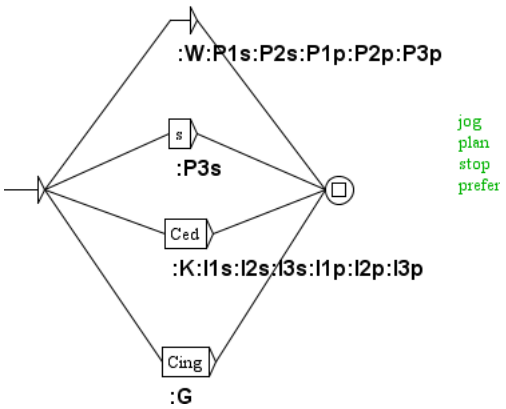
- *foot* is a noun for which the last letter remains in the plural while the string *oo* is replaced by *ee* (*N10.grf* graph).
- *man*, *woman*, *hypothesis* are nouns for which the penultimate letter becomes *e* in the plural (*N11.grf* graph).

	
N10	N11

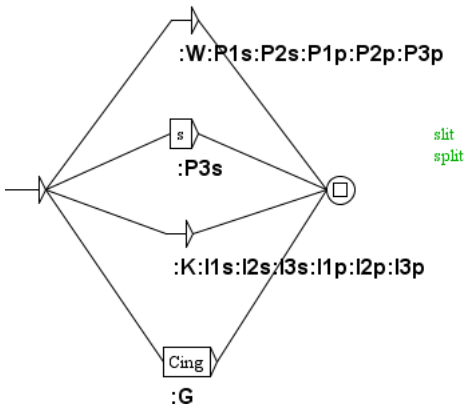
The *C* operator (*Copy*) duplicates the current letter, that is the letter situated on the left of the cursor, which permits the inflection with a single graph of the verbs *jog*, *plan*, *stop*, *prefer* for which the final consonant is duplicated in the past tense, the past participle and the gerund (*V4.grf* graph).

Other examples:

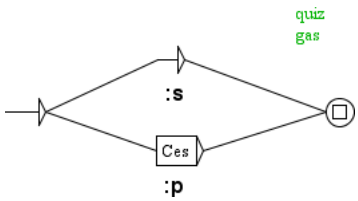
- past tense is identical to the present tense for verbs *slit*, *split* while their final consonant is duplicated in the gerund (*V5.grf* graph).
- the final consonant is duplicated in the plural before adding *es* for the nouns *quiz* or *gas* (*N12.grf* graph).



V4



V5



N12

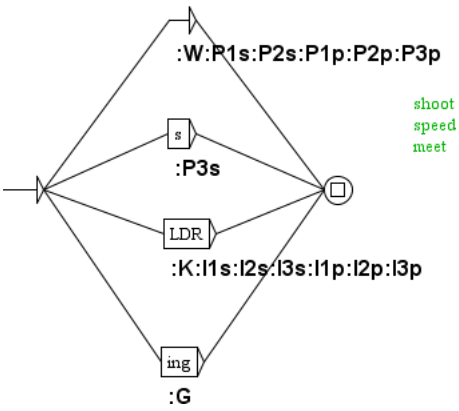
The *D* operator (*Delete*) deletes the current letter (and shifts the right-hand part of the word to the left), that is the letter situated to the left of the cursor, which permits the inflection with a single graph of the verbs *shoot*, *speed*, *meet* for which the double letter (*oo*, *ee*) is simplified in the past tense and the past participle (*V6.grf* graph).[111] We will demonstrate step by step how the pattern *LDR* used in the *V6.grf* graph works.

[111] Note that some other operators exist that are described in the Unitex user manual, Section 3.5.1.

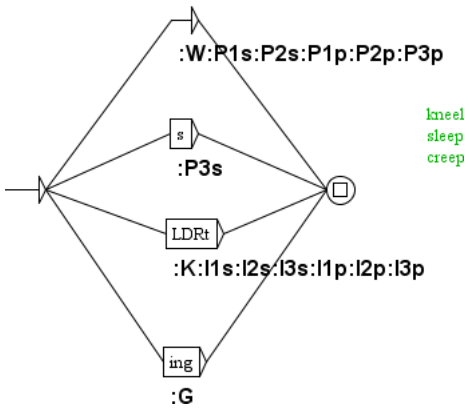
|LDR	shoot|	meet|
L|DR	shoo|t	mee|t
LD|R	sho|t	me|t
LDR|	shot|	met|

Other examples:

- the verbs *kneel*, *sleep*, *creep* for which the double letter (*oo*, *ee*) is simplified in the past tense and the past participle and the letter *t* is added (*V7.grf* graph).



V6



V7

We will now add these eight nouns and twelve verbs into the *myDictionary.dic* dictionary (do not forget to press the *Enter* key after the last one). This dictionary now has 52 entries.

```
...
goose,N9
...
creep,V7
```

We will save the dictionary and close the Unitex editor.

After opening the *myDictionary.dic* file, using the *DELA/Open Recent* menu, and inflecting it, choosing the *DELA/Inflect* menu, the new *myDictionaryflx.dic* dictionary contains 144 lines (if the *Sort and factorize inflectional codes* option is on). After compressing this dictionary (*DELA/Compress into FST* menu), we will close both windows (*myDictionary.dic* and *myDictionaryflx.dic).* We will apply the new *myDictionaryflx.bin* dictionary (*Text/ApplyLexical Resources* menu). The *Word List* now displays 103 recognized words. We will close the *Word List*.

3 Inflection of multi-word units

Before starting this section

If you have not modified the *Norm.txt* file as explained in Chapter 2, Section 3.1.2, page 35, do so, or download it, then close and reopen Unitex.

In order to inflect multi-word units (MWU), or compound words, we will create graphs that have as many boxes as there are tokens in an MWU with the addition of one box for the result. One has to remember that processing units, or *tokens*, for Unitex are strings of alphabetic characters (letters) and any other individual character, including the space.[112]

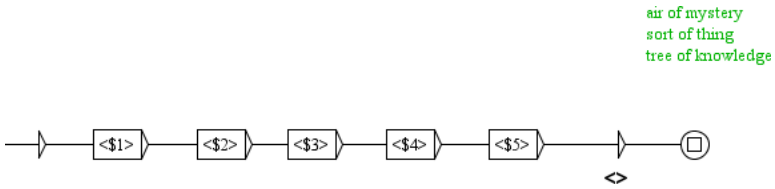
3.1 Example of the MWU *air of mystery*

MWUs *air of mystery*, *sort of thing* and *tree of knowledge*, each comprised of five tokens (three alphabetic strings and two spaces), of which only the first one is inflected to obtain the plural form. We will inflect these MWUs with the graph named *NC_XWW1*.[113]

We will now add three entries to the *myDictionary.dic* file (and click *ENTER* after the third line) using the Unitex editor (*File Edition/Open…/Dictionaries*; answer *OK*). These entries contain the information about the inflection of the MWU as a whole (in this case *NC_XWW1*) and between parentheses (without any space) information about the inflection of each word token (if it inflects): lemma, inflectional graph and inflectional information of a simple word form used in a MWU being described.

```
…
air(air.N1:s) of mystery,NC_XWW1
sort(sort.N1:s) of thing,NC_XWW1
tree(tree.N1:s) of knowledge,NC_XWW1
```

The *NC_XWW1.grf* graph consists of six boxes: the first five boxes refer to the five tokens in a MWU, while the sixth box is empty, it is used only for output.

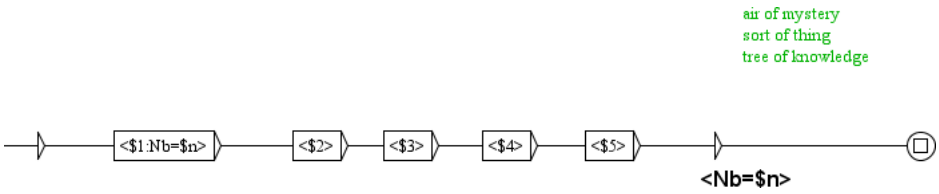


The second (<$2>) and the fourth (<$4>) token correspond to spaces, so they do not inflect. The third (<$3>) and the fifth (<$5>) token refer to word tokens that do not inflect in described MWUs. Only the first token inflects in number. In order to indicate this, we will use the keyword *Nb* representing the grammatical category of number.[114] The formula *Nb=$n* means that the variable *$n* is instantiated to all possible values of the keyword *Nb*, in this case, *s* and *p*. This means that the single path in the graph corresponds to both singular and plural forms of a MWU being inflected. This formula in the first box indicates that all forms (actually two) of the first word token will be taken in turn, while its use in the sixth box indicates that the corresponding values will be assigned to the inflected MWU forms in the output.

[112] See Chapter 2, Section 1.2.3, page 13.

[113] The prefix NC_ in this name indicates that a noun to be inflected is a compound. The suffix is free.

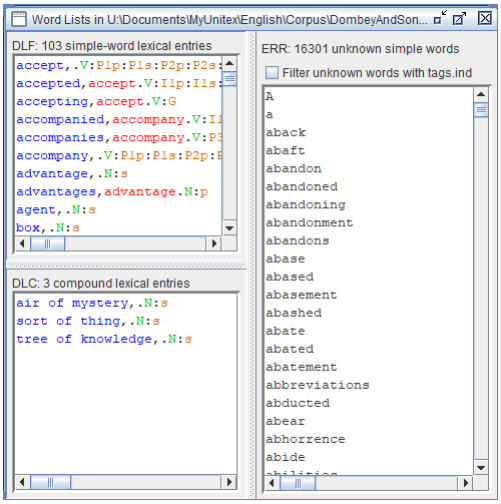
[114] The keywords and their possible values are given for each language in the *Morphology.txt* file that has to be in the same folder as the inflectional graphs.



We will close this graph after compiling it. We will open the *myDictionary.dic* file using the *DELA/Open Recent* menu and inflect it using the inflectional graphs in the folder:

MyUnitex\English\Inflection\UnitexGettingStarted

The new *myDictionaryflx.dic* dictionary has 150 entries, two for each of the newly added MWU lemmas. After its transformation (*DELA/Compress into FST* menu), we will close the windows with the *myDictionary.dic* and *myDictionaryflx.dic* dictionaries; and open the *Text/ApplyLexical Resources* menu, click the *Clear* button, select the *myDictionaryflx.dic* dictionary and then click the *Apply* button. The number of recognized simple words listed in *Word List* is still 103 (*DLF*), but there are also three MWUs (*DLC*).



We will close the *Word List*.

3.2 Some other examples

We will open for the last time the *myDictionary.dic* file using the *File Edition/Open…/Dictionaries* menu, and clicking the *OK* button. We will add to it lemmas for some other illustrative examples of MWUs.

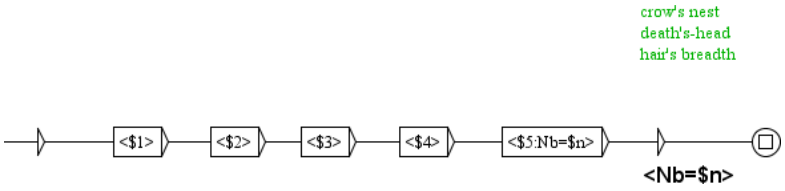
- *crow's nest, death's-head, hair's breadth;* these are MWUs composed, like before, of five tokens, but this time only the last one is inflected (*NC_WWX1.grf* graph); the only difference between this graph and the *NC_XWW1.grf* graph is that now the fifth instead of the first box contains the formula *Nb=$n* indicating the inflection in number.
- *boa constrictor*, *prince regent*, *secretary general*; these are three token MWUs in which only the first word token inflects (*NC_XW1.grf* graph).
- *altar rail*, *Anglo-Norman*, *bad boy;* only the last word token inflects (*NC_WX1.grf* graph).
- *after-life*, *eye-cup*, *sky-blue*; these MWUs inflect like the previous except that the first and the third word tokens can be written together (*NC_WX2.grf* graph); the fact that two words can be written without the hyphen is indicated by the new path that connects directly the first and the third token.

- *three quarter*, *blood red*, *self glorification*;these MWUs inflect like the previous except that the first and the third token can be connected with a hyphen (*NC_WX3.grf* graph); the new path is added to the *NC_WX3.grf* graph which connects the *<$1>* and *<$3>* boxes with a hyphen in between.
- *carry out*, *take off*, *wash out* are MWU which unites two previous cases, namely, two word tokens can be either written together or separated by a hyphen or a space, while only the last word inflects (*NC_WX4.grf* graph).
- *attorney general, notary public* are MWUs that allow inflection of either the first or the second word (but not both!) (*NC_XX1.grf* graph); this graph has two paths, the upper taken from the *NC_XW1.grf* graph and the lower from the *NC_WX1.grf* graph. [115]

We will get the final version of the *myDictionary.dic* dictionary by adding these twenty entries:

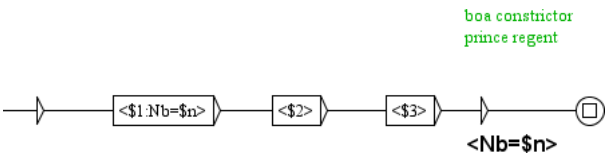
```
…
crow's nest(nest.N1:s),NC_WWX1
death's-head(head.N1:s),NC_WWX1
hair's breadth(breadth.N1:s),NC_WWX1
boa(boa.N1:s) constrictor,NC_XW1
prince(prince.N1:s) regent,NC_XW1
secretary(secretary.N3:s) general,NC_XW1
altar rail(rail.N1:s),NC_WX1
Anglo-Norman(Norman.N1:s),NC_WX1
bad boy(boy.N1:s),NC_WX1
after-life(life.N5:s),NC_WX2
eye-cup(cup.N1:s),NC_WX2
sky-blue(blue.N1:s),NC_WX2
three quarter(quarter.N1:s),NC_WX3
blood red(red.N1:s),NC_WX3
self glorification(glorification.N1:s),NC_WX3
carry out(out.N1:s),NC_WX4
take off(off.N1:s),NC_WX4
wash out(out.N1:s),NC_WX4
attorney(attorney.N1:s) general(general.N1:s),NC_XX1
notary(notary.N3:s) public(public.N1:s),NC_XX1
```

We will construct and compile these seven inflectional graphs.

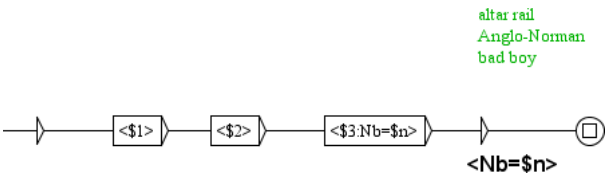


NC_WWX1

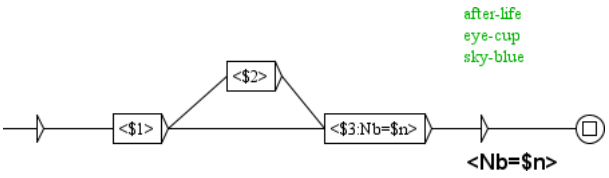
[115] We thank Agata Savary for these three examples: *carry out*, *take off* and *attorney general.*



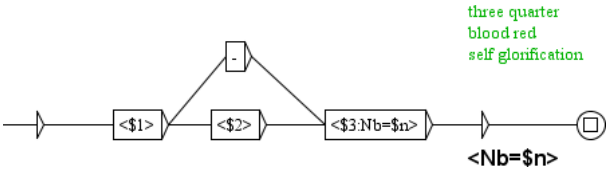
NC_XW1



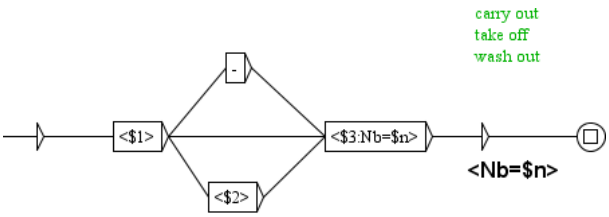
NC_WX1



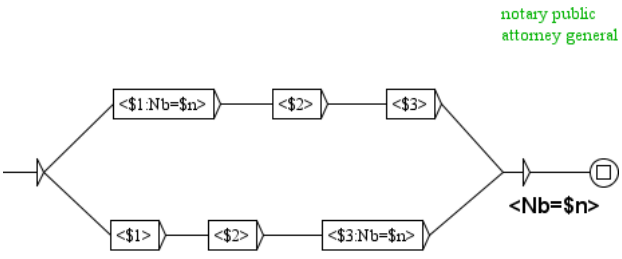
NC_WX2



NC_WX3



NC_WX4



NC_XX1

We will now close all graphs using the *Graphs/Close all* option. Like before, we will open the *myDictionary.dic* file using the *DELA/Open Recent* menu and inflect it (*DELA/Inflect* menu). The final *myDictionaryflx.dic* dictionary consists of 216 entries. After transforming it (*DELA/Compress into FST* menu), we will close both *myDictionary.dic* and *myDictionaryflx.dic* windows. We will apply our dictionary to the text (*Text/ApplyLexical Resources*, clear all pre-selections and select *myDictionaryflx.dic*). The number of recognized MWUs listed in the *Word List* is now 21 (*DLC*).

4 Some additional remarks

4.1 Adding features in dictionary entries

Features with additional information can be added to Unitex dictionaries. In the Unitex user manual, these features are referred to as *semantic codes.* In a dictionary of lemmas these features follow an inflectional class (a name of an inflectional graph) and are preceded by the *+* character. Features that are added to the entries in a dictionary of lemmas are transferred to the entries of a dictionary of inflected forms.

Let us turn again to our dictionary of lemmas (*myDictionary.dic*) that we will modify by adding semantic codes: *abstract* for abstract nouns, *animal* for nouns denoting animal, *concrete* for concrete inanimate nouns and *human* for nouns denoting human.

```
advantage,N1+abstract
agent,N1+human
firm,N1+concrete
…
wash out(out.N1:s),NC_WX4+abstract
notary(notary.N3:s) public(public.N1:s),NC_XX1+human
attorney(attorney.N1:s) general(general.N1:s),NC_XX1+human
```

If you add one of these semantic codes to all noun entries in the lemma dictionary, then our form dictionary (*myDictionaryflx.dic*) will have 133 semantically described entries.

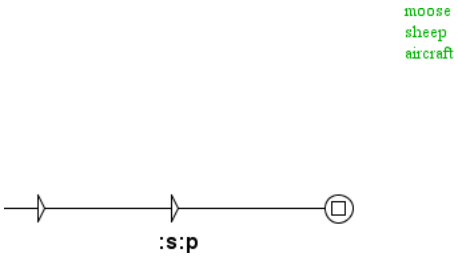
```
Anglo-Norman,Anglo-Norman.N+human:s
Anglo-Normans,Anglo-Norman.N+human:p
…
advantage,advantage.N+abstract:s
advantages,advantage.N+abstract:p
after-life,after-life.N+abstract:s
after-lifes,after-life.N+abstract:p
afterlife,after-life.N+abstract:s
afterlifes,after-life.N+abstract:p
agent,agent.N+human:s
agents,agent.N+human:p
…
```

These semantic codes can be used to refine a lexical mask, for instance by writing *<N+abstract>*.

4.2 Direct creation of inflected entries

It is possible to add entries directly into the *myDictionaryflx.dic* dictionary, which can be useful, for instance, for words that do not inflect. However, one has to be careful because if

the *myDictionaryflx.dic* dictionary is recreated from *myDictionary.dic*, these manually added entries would be lost. It is better thus to have a new inflectional graph, named N13.*grf*, describing nouns that do not change in plural, like *moose*, *sheep* or *aircraft*.




```
…
moose,N13+animal
sheep,N13+animal
aircraft,N13+concrete
```

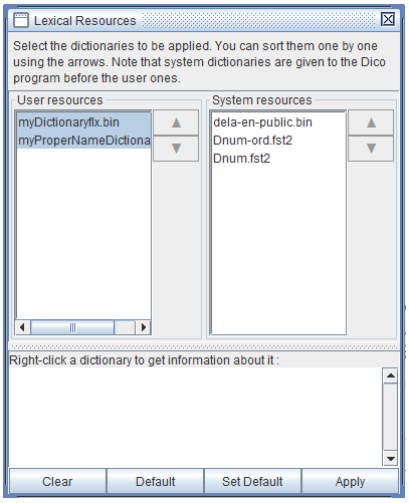
On the other hand, one can directly create a dictionary of inflected forms, which can be useful, for instance, for a dictionary of proper names that usually do not inflect in number. For instance, a *myProperNameDictionary.dic* dictionary can be created directly containing the following entries, collected on the text as you go.

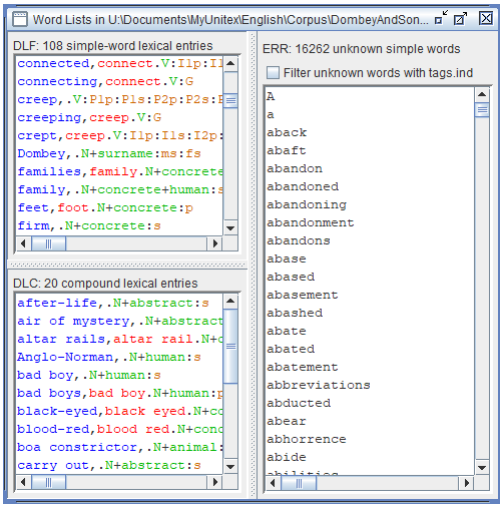
```
Dombey,.N+surname:ms:fs
Paul,.N+forename:ms
Florence,.N+forename:fs
Peps,.N+surname:ms:fs
```

This dictionary can then be sorted by the *Dela/Sort Dictionary* menu.

```
Dombey,.N+surname:ms:fs
Florence,.N+forename:fs
Paul,.N+forename:ms
Peps,.N+surname:ms:fs
```

After transforming it (*DELA/Compress into FST* menu), we will close the *myProperNameDictionary.dic* window. We will apply our two dictionaries to the text (*Text/ApplyLexical Resources*, clear all pre-selections and select *myDictionaryflx.dic* and *myProperNameDictionary.dic* by *CTRL*-clicking the mouse button). The number of recognized simple word listed in the *Word List* is now 108 (*DLF*).





4.3 Adding comments

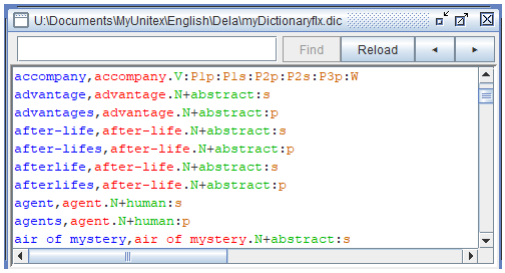
4.3.1 Adding comments to entries

It is possible to add comments to entries in a dictionary of inflected forms. They are placed at the end of a line, after a slash "/". For instance, we can supply some explanatory information in the *myProperNameDictionary.dic* dictionary. This information does not appear in the *Word List*.

```
Dombey,.N+surname:ms:fs
Florence,.N+forename:fs/the daughter of Mr Dombey
Paul,.N+forename:ms/the son of Mr Dombey
Peps,.N+surname:ms:fs/the doctor
```

It is also possible to add comments to the dictionary of lemmas, but these comments are not transferred to the inflected dictionary. It can be sometimes important to memorize who added an entry and when. If we use instead of *myDictionary.dic* the following dictionary of lemmas with added comments, we would still obtain the same *flx* dictionary. Let us modify, for example, the first three lines, then inflect the dictionary. The comments are not in the dictionary of inflected forms.

```
advantage,N1+abstract/ Cvetana, 2022/07/12
agent,N1+human/ Denis, 2022/07/13
firm,N1+concrete/ Cvetana, 2022/07/14
…
```



4.3.2 Documenting dictionaries

Finally, we can provide a description of our two dictionaries in two files, *myDictionaryflx.txt* and *myProperNameDictionary.txt*, which placed in the same folder as dictionaries themselves. For instance (for both):

```
Unitex Getting Started
Denis Maurel
University of Tours
Cvetana Krstev
University of Belgrade
```

This information can be displayed by accessing the *Text/ApplyLexical Resources* menu and right-clicking the name of the dictionary.

4.4 Automatically verifying the format of a dictionary

4.4.1 Inflected dictionary

After adding new entries to a dictionary, we can verify if it is well formed, that is, in the format expected by Unitex. Let us take, for example, the *myDictionaryflx.dic* dictionary and open it using the *DELA/Open Recent* menu. If we select the *DELA/Check Format* menu and click the *Check Dictionary* button, the file named *CHECK_DIC.TXT* will appear in the *DELA* folder and its content will be displayed in a window.

The content of this file in the case of our dictionary will be:

1. Quantitative information: 219 lines read, 159 simple entries for 61 distinct lemmas, 60 compound entries for 23 distinct lemmas.
2. The list of used characters and their UTF8 code.
3. The used grammatical and semantic codes: *6 grammatical/semantic codes used in dictionary: N*, *human, V, abstract*, *concrete* and *animal*.
4. The used morphological codes: *17 inflectional codes used in dictionary, s*, *p, P1p*, *P1s*…

However, if we have not manually added anything to the *myDictionaryflx.dic* dictionary, that is, if it has been automatically produced from *myDictionary.dic*, there is no need to verify its format, since it must be correct.

4.4.2 Lemma dictionary

What we can do is to check the correctness of the dictionary of lemmas (*myDictionary.dic*). After opening it in the *DELA* menu and choosing *DELA/Check Format* we can verify its format by checking the *DELAS/DELAC* option.[116] After applying this option to our dictionary of lemmas, the content of the *CHECK_DIC.TXT* file displayed on the screen will contain the similar information (and replace the previous one):

1. Quantitative information: 78 lines read, 55 simple entries for 55 distinct lemmas, 23 compound entries for 23 distinct lemmas.
2. The list of used characters and their UTF8 code.
3. The used grammatical and semantic codes: 32 grammatical/semantic codes used in dictionary: *N1*, *abstract*, *human*, *concrete*…

[116] This option corresponds to dictionaries of lemmas, while the *DELAC/DELACF* option that is selected by default corresponds to dictionaries of inflected forms.

4. The used morphological codes: *0 inflectional codes used in dictionary*.

4.5 Morphological-mode dictionary (only for confident users)

> Before starting this section
>
> Go to the folder: *MyUnitex\English\Graphs\UnitexGettingStarted*
> Create a new folder named *Dictionaries*.

Dictionary information can be inserted into a text by a graph, for example, to tag it. To do this, you must know how to insert the morphological mode (see Chapter 3, Section 3, page 63) and use variables (see Chapter 4, Section 1.2, page 79).

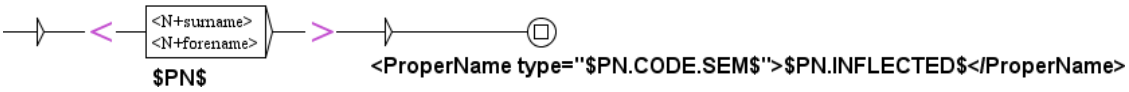
We will use the *myProperNameDictionary.dic* dictionary to markup proper names.

First (don't forget this point!), this dictionary must be allowed for use in the morphological mode: open the *Info/Preferences…/Morphological mode dictionaries* menu and click the *Add* button to select the *myProperNameDictionary.bin* dictionary.

We will also use the *myProperNameDictionaryTag.grf* graph below, saved in the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Dictionaries

PN is a dictionary-entry variable, placed as output text. In the second output, *$PN.CODE.SEM$* is the semantic code corresponding to the dictionary entry (*forename* or *surname*); *$PN.INFLECTED$* is the inflected form corresponding to the dictionary entry (i.e. the proper noun itself).[117]



This graph will produce, in *Replace* mode and with the *Text Order* option, the concordance with 3 493 proper names displayed below. If graphs that you will produce and use later no longer use this dictionary in morphological mode, remember to remove it from the list.

4.6 Priorities

If we use multiple dictionaries, we can set up three priority levels. To do this, we will add the symbols - (maximum priority) or + (minimum priority) to the name of the dictionary file.[118]

[117] See the Unitex user manual, Section 6.4.4, for more information on dictionary-entry variables.

[118] We have to change the name of two files (*.bin* and *.inf*). If it exists, we have to do the same for the *.txt* file.

For instance, if we want to use our two dictionaries, *myDictionary.dic* and *myProperNameDictionary.dic*:

With the dictionary selection:	A word present in both dictionaries will be labeled
myDictionaryflx.bin *myProperNameDictionary.bin*	by each dictionary
myDictionaryflx.bin *myProperNameDictionary+.bin*	only by the *myDictionary.bin* dictionary
myDictionaryflx.bin *myProperNameDictionary-.bin*	only by the *myProperNameDictionary-.bin* dictionary

4.7 Additional notes

The inflectional morphology of English is relatively simple. In French, common nouns inflect, not only in number, but some of them also in gender, and adjectives inflect too. In Serbian, nouns have gender, animacy, number and case, while all these categories inflect for adjectives, in addition to definiteness and comparison. All this can be modeled using the inflectional graphs introduced before.

When inflection also touches the prefix, then the more complex rules introduced in Section 6, page 132 must be used.

For Semitic languages, like Arabic, Hebrew and so on, in which inflection often operates between consonants in the root, specific rules were developed (see the Unitex user manual, Section 3.5.4).

5 Dictionary graphs

5.1 Example of Roman Numeral dictionary

You will recall that we have constructed the *RN2-3999.grf* graph to tag Roman numerals (see Chapter 3, Section 3.4.2, page 67). We now propose to create a dictionary graph for Roman numerals. We will name it *RomanNumeralDictionary.grf* and we will base it on the *RN2-3999.grf* graph. So, we can copy the four graphs: *RN1-9-subgraph.grf*, *RN1-99-subgraph.grf*, *RN1-999-subgraph.grf* and *RN2-3999.grf* from the folder:

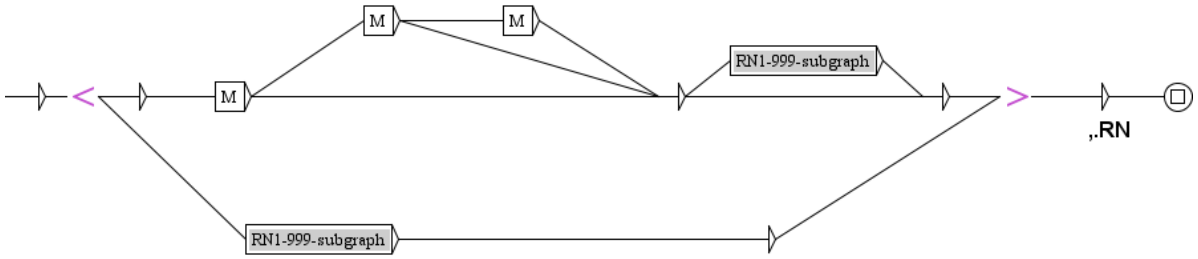
MyUnitex\English\Graphs\UnitexGettingStarted\RomanNumerals

and paste them in the folder:

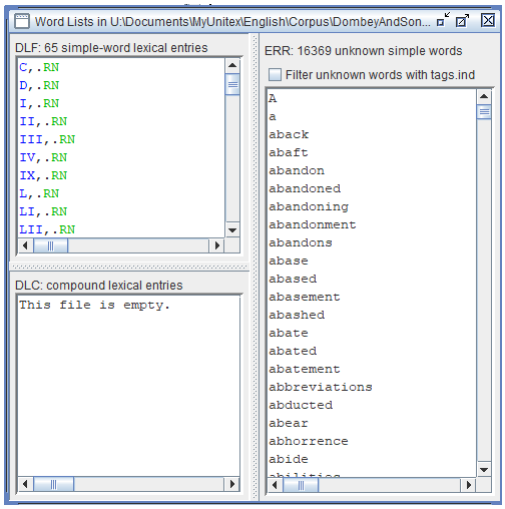
MyUnitex\English\Dela

then rename the *RN2-3999.grf* graph as *RomanNumeralDictionary.grf*.

We are going to remove the insertions of an opening tag *<romanNumeral>* and a closing tag *</romanNumeral>*; and we replace them with empty boxes (*<E>*). We will also replace the call to *:RN2-999* with a call to*:RN1-999-subgraph* and place this call inside morphological tags as well. We will add a new box at the end of the graph with the output *,.RN*.



We will compile this graph, open the *Text/Apply Lexical Resources* menu and click the *Clear* button (in order to cancel all previous selections). We will choose the *RomanNumeralDictionary.fst2* file (appearing in the left field) and click the *Apply* button.[119] As a result the *Word List* window will be displayed.



5.2 Morphological extension of the used dictionaries (only for confident users)

5.2.1 Morphological-mode dictionary declaration

It is also possible to extend the used dictionaries by creating morphological extensions of some entries. The root dictionary must be declared as a morphological-mode dictionary (don't forget this point!). See Section 4.5, page 127.

If we consider that the default English dictionary, *dela-en-public*, is the root dictionary, we have to open the *Info/Preferences…/Morphological mode dictionaries* menu and click the *Add* button to select the *dela-en-public.bin* file. This file is located in your installation folder (see Chapter 2, Section 1.1.2, page 7):

C:\Users\userName\AppData\Local\Unitex-GramLab[120]

or

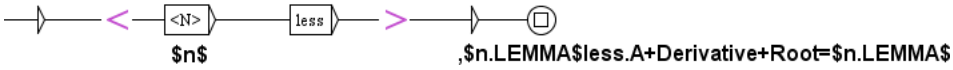
C:\Program Files (x86)\Unitex-GramLab\App

[119] We can also create a text file to write a documentation of the dictionary graph, *RomanNumeralDictionary.txt*, see Section 4.3.2, page 126.

[120] *userName* is your name under the Windows system.

5.2.2 Example of the -less suffix

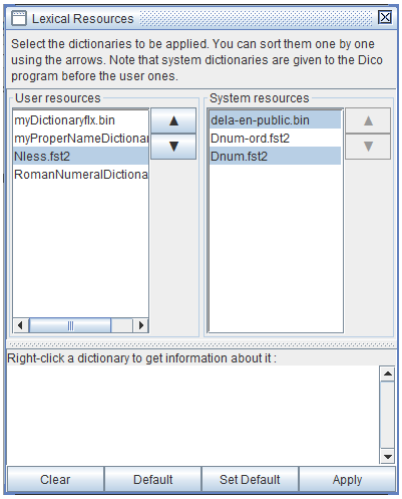
Generally, a noun can generate an adjective with the addition of the *-less* suffix. For instance, in our novel some such adjectives occur: *remorseless* from *remorse*, *restless* from *rest*, *colourless* from *colour* and so on. Many of these adjectives are entries in the default English dictionary, *dela-en-public*, as we can see if we use the *Dela/Lookup…* menu (see Chapter 4, Section 3, page 88). However, in our novel, three adjectives derived from nouns with *-less* suffix occur that are not in this dictionary: *brotherless*, *nevyless*[121] and *whiskerless*. The *Nless.grf* dictionary graph recognizes and tags them appropriately. In addition to the POS tag *A*, it adds two tags: *Derivative* to indicate that this is an input constructed by a graph and *Root=$n.LEMMA$* to specify the root.[122]

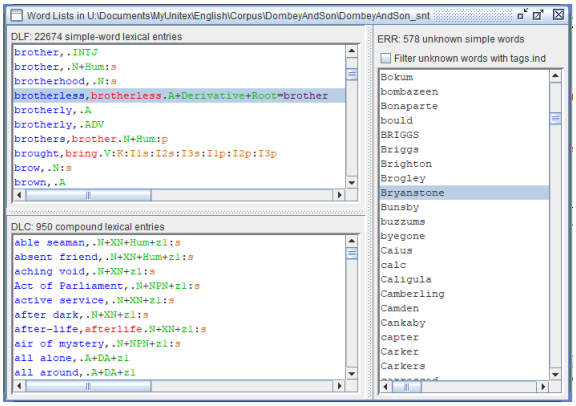


This graph should be added in the list of resources used to analyze the text: we must save and compile it in the folder:

MyUnitex\English\Dela

It recognizes adjectives composed of a noun (in a morphological-mode dictionary, not necessarily present in the applied dictionaries) and the *-less* suffix. We will now apply lexical resources to our text, this time adding to preselected dictionaries the *NLess.fst2* dictionary graph.[123] The number of simple-word lexical entries is now 22 674 (compared to 22 604 obtained before).[124]



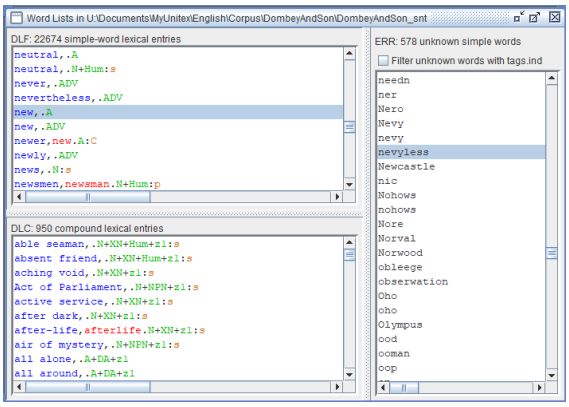


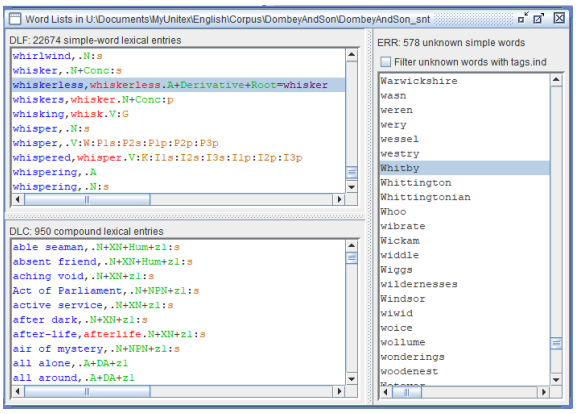
[121] The word *nevy* is an old form of *nephew*.

[122] The dictionary-entry variables are explained in Section 4.5, page 127.

[123] We can also create a text file to document the dictionary graph, *NLess.txt*, see Section 4.3.2, page 126.

[124] See Chapter 2, Section 1.2.3, page 13.





We notice that our graph recognized two of the three afore-mentioned adjectives. It could not recognize the adjective *nevyless* because the noun *nevy* itself is not in the applied dictionary.

This dictionary graph adds seventy entries to the *Word List*. We found two, what are the other sixty eight?

First, two errors, from the *b,.N* entry.

```
bless,bless.A+Derivative+Root=b
Bless,bless.A+Derivative+Root=b
```

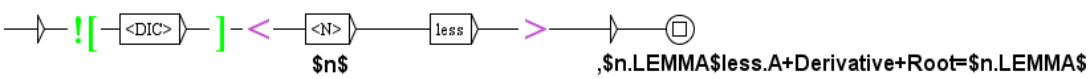
Second, four entries starting with an uppercase letter, although already present in lowercase.[125]

```
Homeless,homeless.A+Derivative+Root=home
Restless,restless.A+Derivative+Root=rest
Speechless,speechless.A+Derivative+Root=speech
Supperless,supperless.A+Derivative+Root=supper
```

Third, sixty four duplications of entries already present.

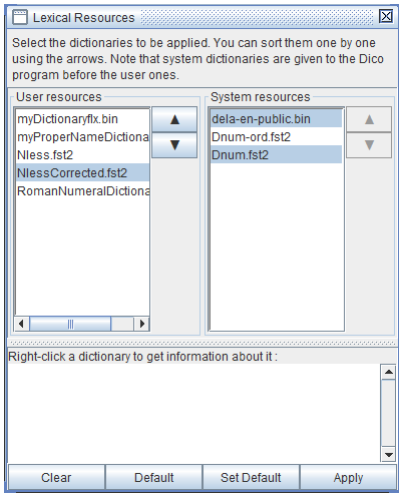
```
artless,.A
artless,artless.A+Derivative+Root=art
…
voiceless,.A
voiceless,voiceless.A+Derivative+Root=voice
```

We will now make a small addition to our graph. The *NlessCorrected.grf* dictionary graph only recognizes the two adjectives ending with *-less* that are not already in applied dictionaries (*brotherless* and *whiskerless*). It uses a negative right context (see Chapter 3, Section 1.7.3, page 55) and a <DIC> mask (see Chapter 2, Section 2.4.3, page 34).



We will now apply lexical resources to our text, this time adding to preselected dictionaries the *NlessCorrected.fst2* dictionary graph.

[125] If the graph does not add a lemma (the output *,.A* instead of the output *,$n.LEMMA$less.A*), the graph creates four new lemmas, *Homeless*, *Restless*, *Speechless* and *Supperless*, which is best avoided.



The number of simple-word lexical entries is now 22 606 entries.

6 Additional possibilities (only for confident users)

6.1 A test on a suffix

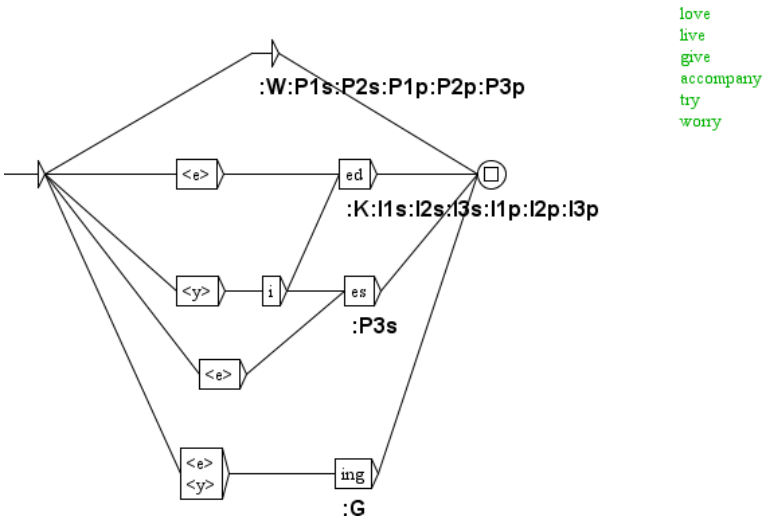
By testing the last letter of a verb, we can join the *V2.grf* and *V3.grf* graphs into one graph, named *V101.grf*. In order to illustrate this, we will create the *myAdvancedDictionary.dic* file containing the following six entries (remember that there should be no spaces, and that you should press the *Enter* key at the end of the last line). You will remember that the first three of these verbs were inflected with the *V2.grf* graph in our *myDictionary.dic* dictionary, while the last three verbs were inflected with the *V3.grf* graph.

```
love,V101
live,V101
give,V101
accompany,V101
try,V101
worry,V101
```

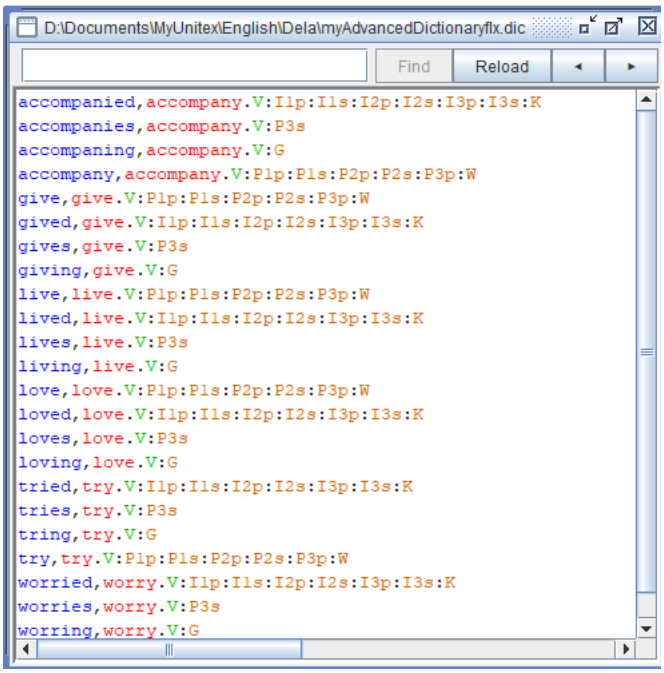
In the new *V101.grf* graph, we will use tests on the verb suffixes and inflect all six verbs. The occurrence of *<e>* in an inflectional graph means that the test is made on the verb suffix; if this suffix is *e,* it is removed and the inflection proceeds further. For instance, in order to produce forms of the third person singular of the present tense:

- the *<e>* box reads the suffix *e*, removes it and the next box adds *es:P3s*
- the *<y>* box reads the suffix *y*, removes it and the following boxes write first *i* and then *es:P3s*

which yields, for instance, *lov~~e~~es:P3s* and *accompan~~y~~ies:P3s*.



Now we can inflect the *myAdvancedDictionary.dic* dictionary by choosing the *DELA* menu and the *Inflect* option. The file containing the *myAdvancedDictionaryflx.dic* inflected dictionary will be created and displayed on the screen.



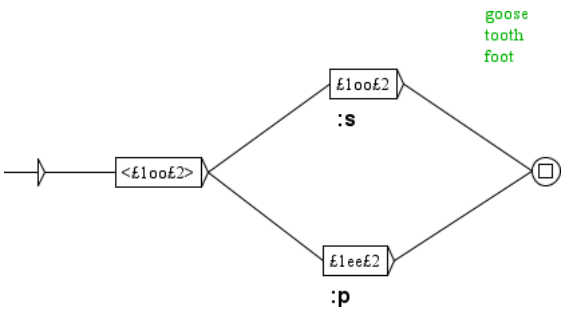
6.2 Use of variables

6.2.1 Example of goose, tooth, foot

Let us look at these three nouns: *goose*, *tooth*, *foot*. With the *N9.grf* graph we can inflect the first two nouns, but not the third for which a new graph should be written in which the rule *LLLLeeRR* would be replaced by *LLLLeeR* (there is one consonant, instead of two at the end of the noun). There is another way to formulate the rules that is more complex but also more powerful. It is based on two variables, *£n* and *$n*, that memorize the longest and the shortest strings, respectively, that occur where they are placed in the pattern (put in angular brackets)

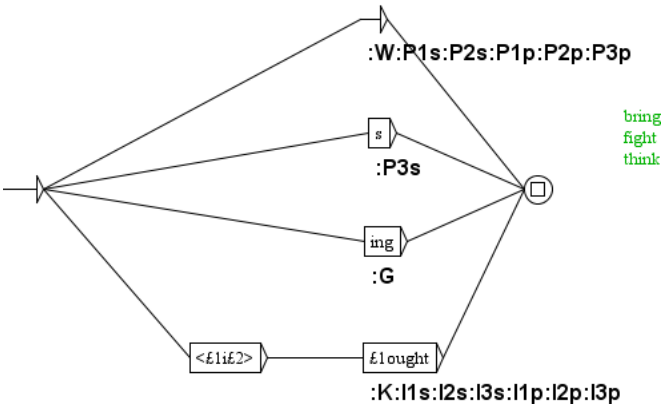
in a box of the graph. This pattern is matched with the input word, starting from the right end.[126] Several such variables can be used and numbered from n=1 to n=9.

In the *N101.grf* graph the first box contains the pattern *<£1oo£2>* which means that the variable *£2* will be assigned the longest string preceded by *oo* in the processed word (that is, *se*, *th*, *t*, respectively), then the variable *£1* will be assigned the longest string before *oo* (that is, *g*, *t*, *f,* respectively). In the case of the singular, the word is rewritten, while for the plural *oo* is replaced by *ee* keeping everything before and after it as it was, which seems as the most natural way to express this rule.[127]



6.2.2 Example of bring, fight, think

The use of variables can be further illustrated with the inflection of the verbs *bring*, *fight*, *think* for which the letters before *i* are suffixed by *ought* in the past tense and the past participle (*V102.grf* graph).



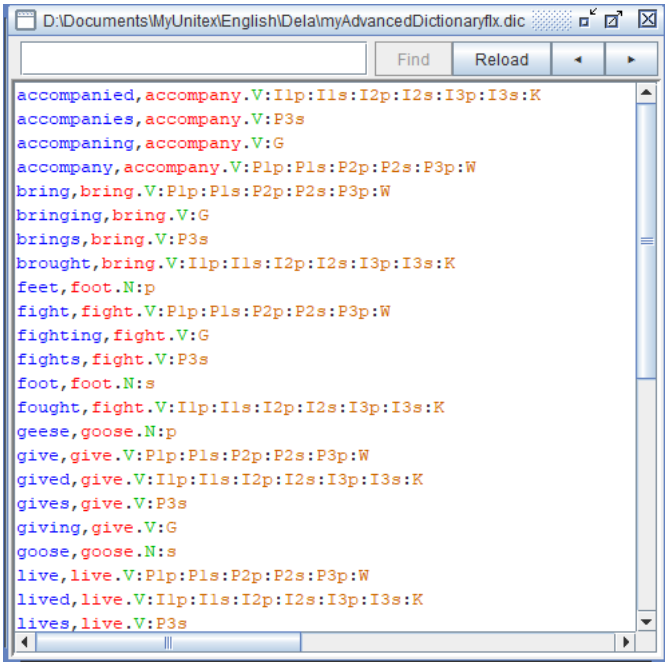
[126] For detailed explanation of the meaning and use of these variables, see the Unitex user manual, Section 3.5.2.

[127] Note that in the string £1oo£2, which is not inside angular brackets, variables £1 and £2 are not assigned values, but their previously assigned values are used.

We will add to our *myAdvancedDictionary.dic* dictionary the following six entries (without spaces and with clicking the *Enter* key at the end of the last line):

```
goose,N101
tooth, N101
foot, N101
bring,V102
fight,V102
think,V102
```

We will follow the preceding steps to inflect the *myAdvancedDictionary.dic* dictionary. The created file named *myAdvancedDictionaryflx.dic* will be displayed on the screen.



Chapter 6: other tools

1 Lexicon-grammar

Before starting this section

Go to the folder: *MyUnitex\English\Graphs\UnitexGettingStarted*
Create a new folder named *LexiconGrammar* (without space).
Go to the folder: *MyUnitex\English\Graphs\UnitexGettingStarted\LexiconGrammar*
Create a new folder named *ParticleVerbs* (without space).
Go to the folder:*MyUnitex\English\Graphs\UnitexGettingStarted\LexiconGrammar\ParticleVerbs*
Download the *particleVerbTable.xlsx* file.

1.1 Example: particle verbs

We will click the *Text/Open…* menu to open the file:

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSon\DombeyAndSon.txt

Then answer *Yes* to the question *Do you want to preprocess the text?* A window will open in which we will remove the first two checkmarks, leaving just the last one (*Apply All default Dictionaries*).

The aim of this section is to automatically transform a lexicon-grammar table into a graph. We will prepare in some spreadsheet program a table that describes how some verbs combine with a set of particles to obtain particle verbs. This table *particleVerbTable.xlsx* is in the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\LexiconGrammar\ParticleVerbs

	A	B	C	D	E	F	G	H	I	J	K
1	Particle verbs										
2			around	away	back	down	in	off	on	out	up
3	break	+	-	+	+	+	+	+	-	+	+
4	carry	+	+	-	+	+	+	+	+	+	-
5	get	+	+	+	+	+	+	+	+	+	+
6	give	+	-	+	+	+	+	+	+	+	+
7	put	+	+	+	+	+	+	+	+	+	+
8	speak	+	-	+	+	-	-	+	+	+	+
9	turn	+	+	+	+	+	+	+	+	+	+
10											

The *A* column in the table contains particle verbs. The plus sign in *B* column indicates that the verb can be used without particles (all verbs in our table can be used without a particle as well). The *C* to *L* columns signal particles with which a verb can be used. The plus sign in a cell indicates that the verb and the particle form a particle verb, while the minus sign means the opposite.

1.2 Transformation of the spreadsheet table

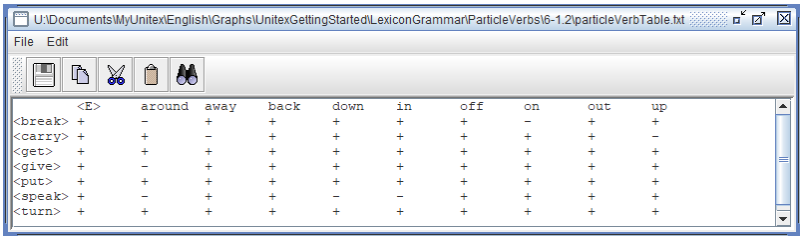
The table that we prepared in a spreadsheet program must be transformed into a Unitex lexicon-grammar table.

First, in the *particleVerbTable.xlsx* spreadsheet, verbs in *A* column must be enclosed in angular brackets so that they should match all forms of the listed verbs. *B2* cell is empty because it is the header of the column about the use of the verbs without a particle; we will write here the symbol *<E>* that corresponds to an empty box in a graph.This file should consist only of table

data with only one header line. We will therefore select *A2:K9* (the *2* to *9* lines and the *A* to *K* columns) and copy its content. We will now open the Unitex editor (the *File Edition/New File* menu) and paste the content from the table.[128]

We will save this file with the name *particleVerbTable.txt* in the same folder:

MyUnitex\English\Graphs\UnitexGettingStarted\LexiconGrammar\ParticleVerbs

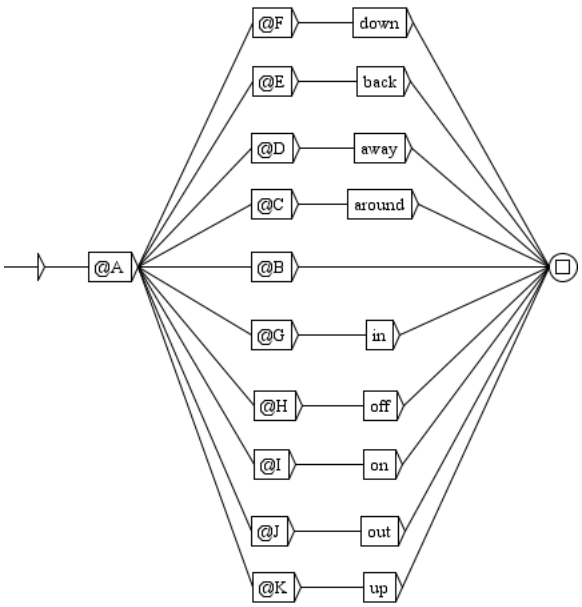


1.3 Parameterized graphs

We will write a graph that retrieves in a text the constructions described in the table and we will save it in the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\LexiconGrammar\ParticleVerbs

using the name *particleVerbParameterized.grf*. This graph uses variables *@A*, *@B*, *@C*… in which the content of cells *A*, *B*, *C*… will be stored. If the variables contain a text it will be copied in the working graph produced from this parameterized graph. If the value of the variable is a plus or a minus sign, the box functions as a test: with *+*, the test is passed and the graph proceeds with what follows, while with - the path is interrupted. There is one more variable, *@%*, that contains the number of the line, which we will use for the name of subgraphs.

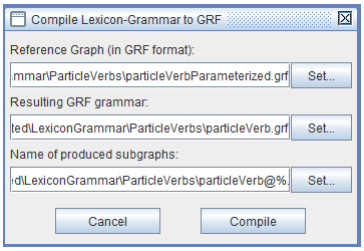


1.4 Generation of graphs

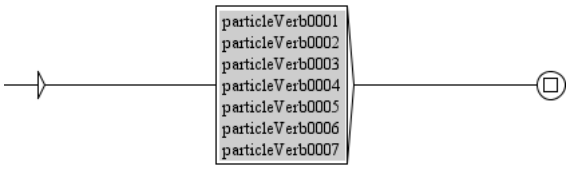
We will make Unitex generate a graph from this parameterized graph and the table. This graph will invoke one subgraph for each line in the table. In order to do this, we must open the

[128] By using the Unitex editor, we are sure that the lines and columns are in the Tabulation/Line break format and that the text is in UTF-8.

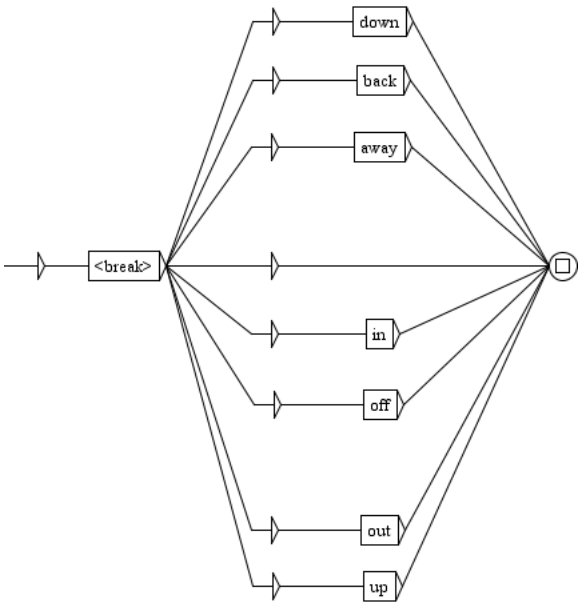
Lexicon-Grammar/Open menu and select the *particleVerbTable.txt* table. Next, we must use the *Lexicon-Grammar/Compile to GRF…* menu to open (by clicking the *SET* button in the first line) the *particleVerbParameterized.grf* parameterized graph. In the second line, we can choose the name of this graph and modify it to *particleVerb.grf*. In the third line, we will choose the previous graph name and change it to *particleVerb@%.grf* (remember that *@%* is the special variable containing the line number). Finally, we will click the *Compile* button.



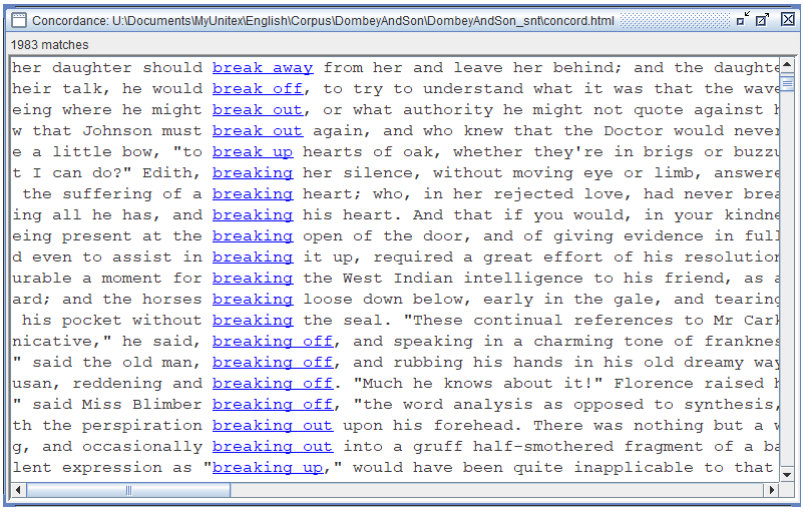
Now we can open the *particleVerb.grf* graph. It invokes seven subgraphs, one for each line in our table (except the header line) which are numbered 0001 to 0007.



We can open the first subgraph by selecting it and clicking the right mouse-button and choosing the *Open subgraph* option. We see that this graph has six paths for six particles used by the verb *break* and one path for the verb without a particle.

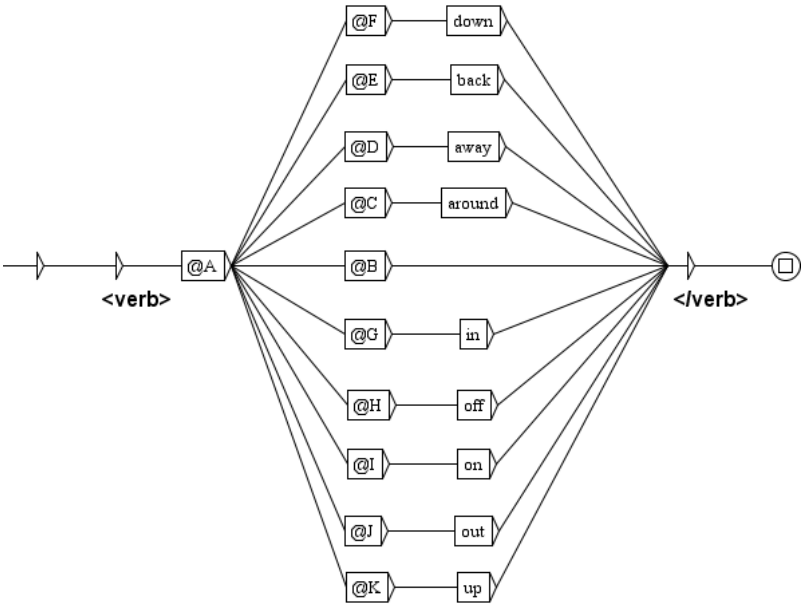


By using the produced *particleVerb.grf* graph in the *Text/Locate Pattern* menu we will get 1 983 concordance lines.

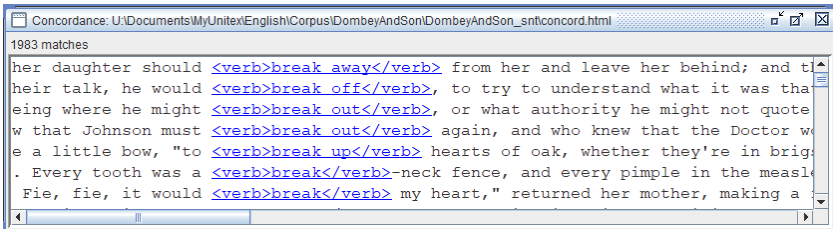


1.5 Annotations

The parameterized graph can be more complex than the previous example. For instance, the *particleVerbAnnotationParameterized.grf* graph annotates the recognized verbs (with particles or not).



After repeating the process of graph generation, and its application to the text in the *Merge* mode, we will obtain a new concordance.

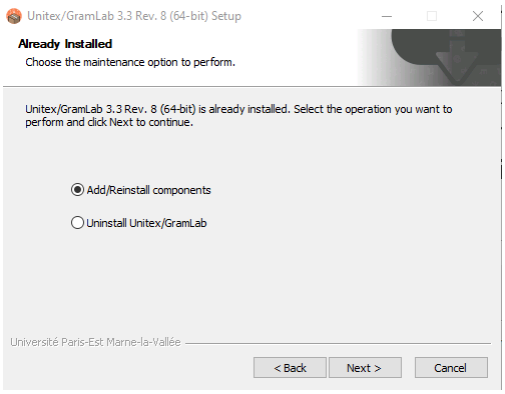


2 XAlign

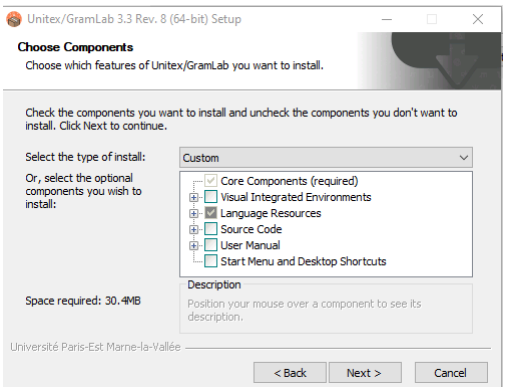
2.1 Installation

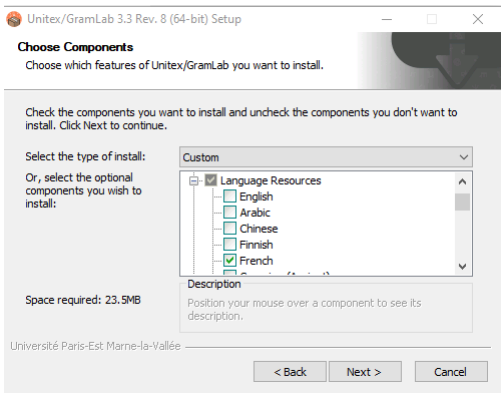
The aim of this software that is integrated into Unitex is to align two similar texts, for instance two drafts of the same document or a text and its translation. We will demonstrate how to use this software with the example of a text and its translation.

It is important to note that in order to reproduce the exercises from this chapter one should have downloaded the French resources for Unitex. If that is not the case, one should repeat the process of the installation of Unitex. However, it is not necessary to repeat the whole installation.



One should only add a language.





2.2 The *French* folder

After automatically creating the *French* folder:

- if Unitex is closed, we should open it by choosing *French* as the working language;
- if it is already open, we will choose *French* as the working language by using the *Text/Change Language…* menu.

It is of course possible to create the French folder yourself, before launching Unitex. But, in this case, Unitex will not copy the three files necessary for its operation (*Norm.txt*,

Alphabet.txt and *Alphabet_sort.txt*).[129] You must do this yourself before running Unitex (or after closing and restarting it).

Before proceeding with this section
1. Go to the folder: *MyUnitex\English\Corpus\UnitexGettingStarted*
Create a new folder named *DombeyAndSonEng* (without space).
Go to the folder:*MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSonEng*
Download the *DombeyAndSonEng.txt* file.

2. Go to the folder: *MyUnitex\French\Corpus\UnitexGettingStarted*
Create a new folder named *DombeyAndSonFra* (without space).
Go to the folder:*MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSonFra*
Download the *DombeyAndSonFra.txt* file.

3. Go to the folder: *MyUnitex*
Create a new folder named *XAlign* (without space).
Go to the folder: *MyUnitex\XAlign*
Create two new folders named *Corpus* and *Sentence*.
Go to the folder: *MyUnitex\XAlign\Corpus\UnitexGettingStarted*
Create a new folder named *DombeyAndSonEngFra*.

2.3 File preparation

We will use files from three folders: *English, French* and *XAlign*. We will choose *English* as the working language by using the *Text/Change Language…* menu.

2.3.1 English *folder*

We will open the file (*File Edition/Open/Text files* menu):[130]

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSonEng\DombeyAndSonEng.txt

We will ignore the message *This is not necessarily the text being processed by Unitex*. We will save this file using the name *DombeyAndSonEng.xml* (we will change from the *.txt* format to *.xml* format).

We will add TEI tags at the beginning and at the end of this file:[131]

<tei>
<teiHeader/>
<body>
<text>
** **
Dombey and Son by Charles Dickens
…
A transient flush of faint surprise overspread the sick lady's face as she raised her eyes towards him.
** **
</text>
</body>
</tei>

[129] About the *Norm.txt* file, see Chapter 2, Section 3.1.2, page 35; about the *Alphabet.txt* and *Alphabet_sort.txt* files, see Chapter 2, Section 3.3.3, page 41.

[130] We produced these samples of text in English and French just in order to illustrate the alignment process.

[131] XAlign does not handle correctly files with a TEI header, since it will delete it and replace it with the empty tag *<teiHeader/>*. If we want that our result files have a header, we can add it after the alignment.

We will save this file and close the editor.

2.3.2 French *folder*

Now we will repeat the process for the *French* version of the text: we will open the file (*File Edition/Open/Text files* menu):

MyUnitex\French\Corpus\UnitexGettingStarted\DombeyAndSonFra\DombeyAndSonFra.txt

We will save this file using the name *DombeyAndSonFra.xml*. Finally, we will add the same tags at the beginning and the end of this file.

```
<tei>
<teiHeader/>
<body>
<text>
<div>
Dombey et fils traduit de Charles Dickens
…
Une rougeur passagère, causée par la surprise, colora légèrement les joues de la malade qui leva les yeux vers son mari.
</div>
</text>
</body>
</tei>
```

We will save this file and close the editor.

2.3.3 XAlign *folder*

We will now open a new file using the editor *File Edition/New File*. In this file, we will copy the following lines:

```
<tei>
<teiHeader/>
</tei>
```

We will save this file (*Save text* button), using the name *DombeyAndSonEngFra.xml*, in the folder:

MyUnitex\XAlign\Corpus\UnitexGettingStarted\DombeyAndSonEngFra

During the alignment XAlign uses a file for sentence splitting for the source language (which is, in our case, *English*); we will use the graph that comes with the Unitex distribution. We will copy the *Sentence.fst2* file from the folder

MyUnitex\English\Graphs\Preprocessing\Sentence

And paste it in the folder:

MyUnitex\XAlign\Sentence

Once this is done, we will rename this file into *SentenceXAlign.fst2*.

2.4 Automatic alignment

We will click the *Text/Open…* menu to open the file:

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSonEng\DombeyAndSonEng.txt

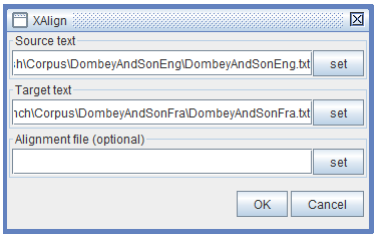
and answer *Yes* to the question *Do you want to preprocess the text*. A window will open in which we will remove the first two checkmarks, leaving just the last one *Apply All default Dictionaries*.

We will open the *Xalign/Open Files* menu and fill the first line (by selecting the *Set* button) with the file:

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSonEng\DombeyAndSonEng.txt

And we fill the second line with the file:

MyUnitex\French\Corpus\UnitexGettingStarted\DombeyAndSonFra\DombeyAndSonFra.txt



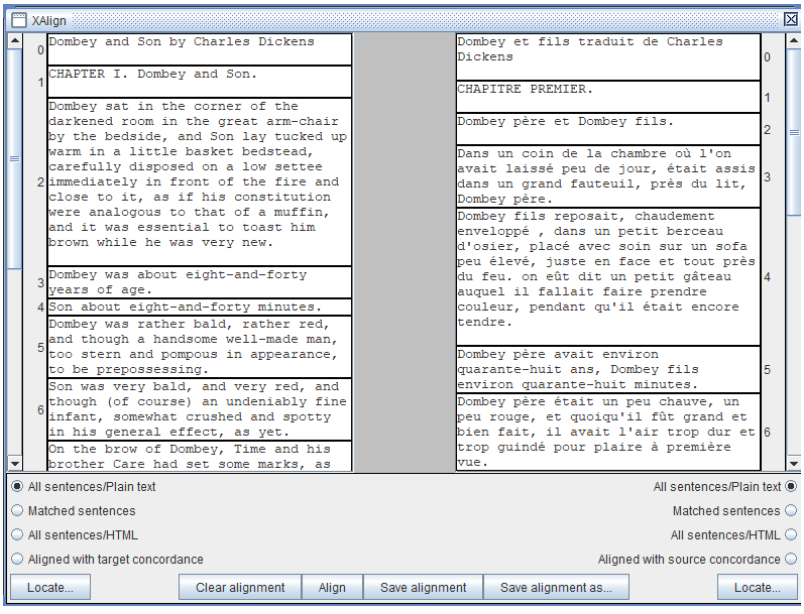
We will click the *OK* button. A window will ask us to choose the XML file that corresponds to the text file in the first line (*source file*). After clicking *OK*, we will choose the first of the previously developed files:

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSonEng\DombeyAndSonEng.xml

Again, another window will open asking us to select an XML file corresponding to the text file given in the second line (*target file*). We will choose the second previously developed file:

MyUnitex\French\Corpus\UnitexGettingStarted\DombeyAndSonFra\DombeyAndSonFra.xml

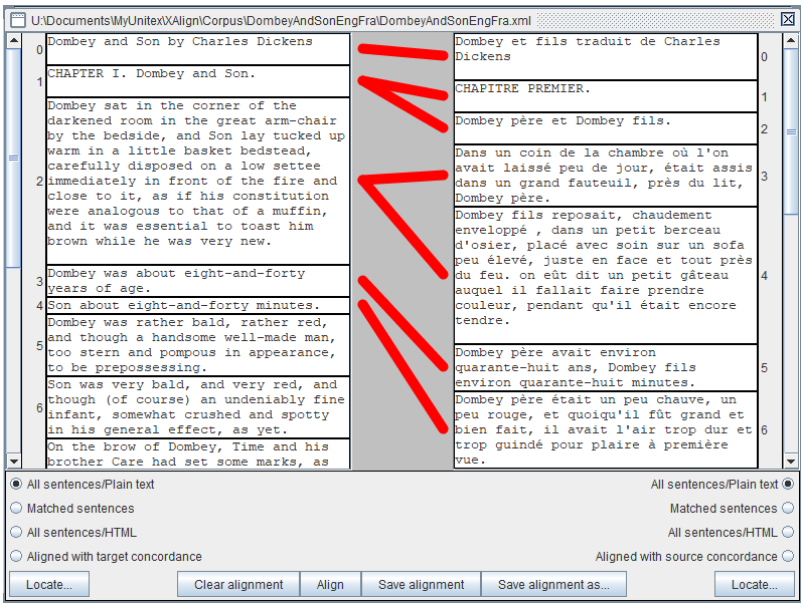
The new window will open:



We will click the *Align* button that will initiate the alignment process. A new window will ask us to select the XML file for the result. We will choose the third and the last file that we created before, namely the file:

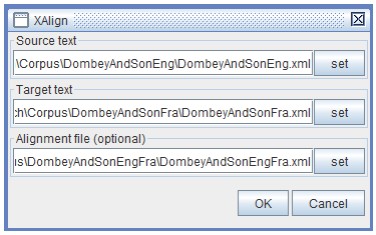
MyUnitex\XAlign\Corpus\UnitexGettingStarted
DombeyAndSonEngFra\DombeyAndSonEngFra.xml

After the alignment process is finished, some error messages will be displayed, which we will ignore, and click the *OK* button



Click the *Save alignment* button.

You should note that if you close Unitex after saving the alignment, you can open them again by filling all three lines with the appropriate XML files.



2.5 Reading and manual correction

We have done the first alignment. It is necessary now to read and correct it. It is enough to click the *English* sentence (or phrase) and then the *French* sentence (or phrase) to establish the link, if it does not exist, or to remove it if it exists:[132]

[132] In order to remove the link it is enough to simply click on it.

```
remove 4-6;
add 4-5;
remove 5-7;
add 5-6;
remove 6-8;
remove 6-9;
add 6-7;
add 7-8;
add 8-14;
remove 9-14;
add 9-15;
add 10-16;
remove 11-16;
remove 11-17;
add 12-15.
```

Click the *Save alignment* button.

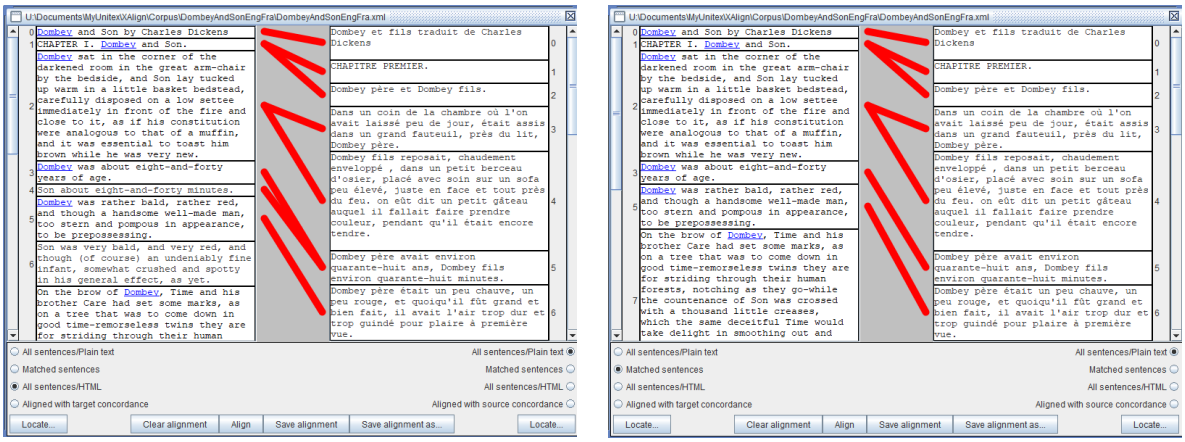
Generally, a sentence is translated into another sentence: (0-0), (5-6), (6-7) (9-15) and (13-18). However, there are exceptions:

- three sentences in the source file, 1, 2 and 10, are translated into two sentences: (1-1, 1-2), (2-3, 2-4) and (10-15, 10-16);
- three sentences in the source file, 7, 8 and 12, are translated into three sentences: (7-8, 7-10, 7-11), (8-12, 8-13, 8-14) and (12-15, 12-16, 12-17);
- a sentence in the target file, 9, is added;
- a sentence in the source file, 11, is not translated;
- two sentences in the source file, 3 and 4, are translated together into a single sentence: (3-5, 4-5);
- two sentences in the source file, 10 and 12, are reversed: (10-16, 12-15).

2.6 Search performed on one of two texts

In XAlign we can use Unitex search functions via *Text/Locate Pattern* and get as a result the visualized alignment. For instance, we can be interested in whether the name of Mr Dombey was systematically represented in the French translation. We will click the *Locate…* button, that can be found on the left side of the *XAlign* window, and then answer *Yes* to the question that appears. Unitex proposes to open a text deduced from the XML file, text which it names with the suffix *_xalign* (*DombeyAndSonEng_xalign.txt*).

We will type *Dombey* in the *Regular expression* line and click the *SEARCH* button. The *_xalign* file will be processed by Unitex and the result will appear in the *XAlign* window, after checking the *All sentences/HTML* option (at the left-hand side). If the *Matched sentences* option is chosen only the sentences that correspond to our search query will be displayed.



In the obtained results we can see that, in the translation (12-16), the proper name was translated by *son épouse* (*his wife*).

We will choose the *All sentences/Plain text* option at the left side of the *XAlign* window in order to start a new search. This time we will use a graph, for instance the *willV.grf* graph that we developed in Chapter 3, Section 1.8, page 56.[133] We will search for this graph (*Set* button) in the *Graph* line and click the *SEARCH* button. We will select the *Matched sentences* option and we will get only one match in the source file sentence 7.

The same options exist on the right of the *XAlign* window. It is therefore possible to search in the target language.

However, it is not possible to launch a cascade on an aligned text, except if we proceed as explained in the next section (only for confident users).

2.7 Cascade search and XAlign (only for confident users)

The main idea is to launch cascades on the *DombeyAndSonEng.xml* file without touching the <p> and <s> tags placed by XAlign.

2.7.1 Launching cascades

We will go to the folder:

MyUnitex\English\Corpus\UnitexGettingStarted\DombeyAndSonEng

and rename the *DombeyAndSonEng.xml* file as *DombeyAndSonEng.xml.txt* (see Chapter 4, Section 2, page 84). We will open this text (*Text/Open* menu) and answer *Yes* to the question *Do you want to preprocess the text*. A window will open in which we will remove the first two checkmarks, leaving just the last one *Apply All default Dictionaries*. We will find the *analysis* cascade (*Apply CasSys Cascade…* menu) in the folder:

MyUnitex\English\CasSys\UnitexGettingStarted\Measures

And we will launch this cascade. The result file is the *DombeyAndSonEng.xml_csc.txt* file. We will open this text with Unitex, without processing it, and launch the *synthesis* cascade. This second result file is the *DombeyAndSonEng.xml_csc_csc.txt* file with two recognized measures (sentences 4 and 5).

[133] Or Chapter 3, Section 1.9.3, page 58, if you have done this section.

```
<s id="n4" xml:id="d1p1s4">Dombey was about <Measure type="Duration"><Number>eight-and-forty</Number> years</Measure> of age. </s><s id="n5" xml:id="d1p1s5">Son about <Measure type="Duration"><Number>eight-and-forty</Number> minutes</Measure>. </s>
```

2.7.2 File transformation

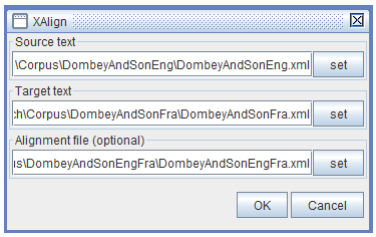
We will rename the *DombeyAndSonEng.xml_csc_csc.txt* file as *DombeyAndSonEng.xml*. If we open this file with XAlign, the tags disappear. To keep them, we will use the HTML codes *<* for < and *>* for >. We will pay attention to modify only our tags (*Measure* and *Number*) and not the <p> and <s> tags produced by XAlign.

```
<s id="n4" xml:id="d1p1s4">Dombey was about &lt;Measure type="Duration"&gt;&lt;Number&gt;eight-and-forty&lt;/Number&gt; years&lt;/Measure&gt; of age. </s><s id="n5" xml:id="d1p1s5">Son about &lt;Measure type="Duration"&gt;&lt;Number&gt;eight-and-forty&lt;/Number&gt; minutes&lt;/Measure&gt;. </s>
```

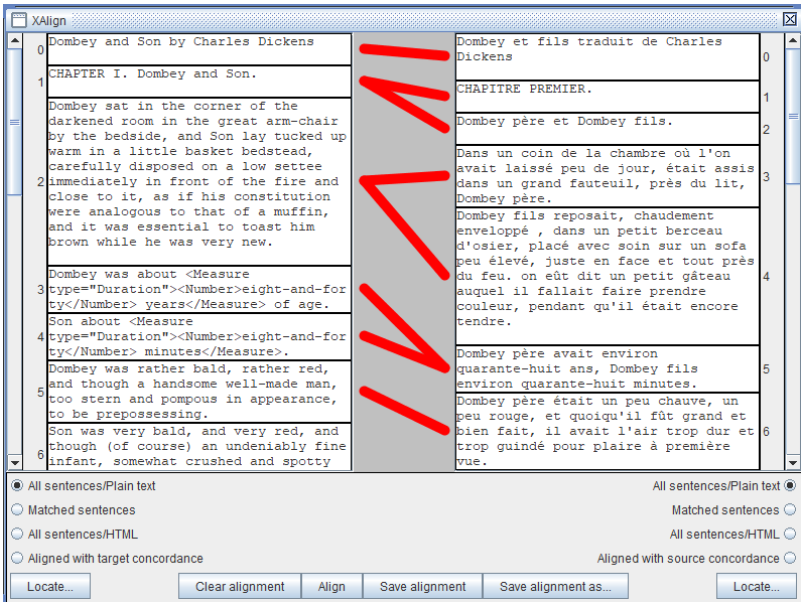
We will save this file.

2.7.3 Using XAlign

If we open XAlign with this new file:



We obtain the desired result.



Chapter 7: scripts

Before reading this chapter, please, read this caution:

This chapter requires some computer knowledge.
You must at least be comfortable with the notion of scripts.

Before starting this chapter:[134]

1. Go to the folder: *MyUnitex*
Create a new folder named *Scripts*.
Go to the folder: *MyUnitex\Scripts*
Create a new folder named *ScriptEntities* (without space).
Go to the folder: *MyUnitex\Scripts\ScriptEntities*
Create three new folders named *Input*, *Output* and *ScriptEntitiesLingpkg*.
Go to the folder: *MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg*
Create two new folders named *Resources* and *Scripts*.

2. Go to the folder:*MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText*
Copy the *jubileePlainText.txt* file.
Go to the folder: *MyUnitex\Scripts\ScriptEntities\Input*
Paste this file.

3. Go to the folder:*MyUnitex\English\Corpus\UnitexGettingStarted\JubileeXmlText*
Copy the *jubileeXmlText.xml* file.
Go to the folder: *MyUnitex\Scripts\ScriptEntities\Input*
Paste this file.

1 Introduction

The Unitex interface is a tool for developing graphs and graph cascades. Once these resources have been developed and tested, it is possible to create Unitex scripts which apply the developed tools on a corpus of several documents stored in several files. These Unitex scripts are launched from a command line and they access a compressed file that can contain all types of resources: Unitex scripts, graphs, cascades and dictionaries. We will take as an example the cascades developed in Chapter 4, Section 1, page 71 and Section 2, page 84. If you have allowed a dictionary for use in the morphological mode, please, remove them now from the list (see Chapter 5, Section 4.5, page 127).

In general, users will copy in the *Input* folder all texts they want to process, as we have done.

2 Creation of the *linguistic package*

The *linguistic package* (*scriptEntitiesLingpkg* folder) will contain everything needed to launch the analysis and synthesis cascades.

[134] Remember that file names should not contain spaces and are case sensitive. The same goes for folder names. See Chapter 2, Section 1.1.1, page 7.

2.1 The *Resources* folder

We will replicate our personal working folder in the *Resources* folder.

1. Go to the folder:*MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Resources*
Create a new folder named *English*.
Go to the folder:*MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Resources\English*
Create three new folders named *CasSys, Dela and Graphs*.

2. Go to the folder: *MyUnitex\English*
Copy the three *Alphabet.txt*, *Alphabet_sort.txt* and *Norm.txt* files.
Go to the folder:*MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Resources\English*
Paste these files.

In general, one will copy in the CasSys folder the cascades that one wants to use for the processing.

Go to the folder: *MyUnitex\English\CasSys\UnitexGettingStarted\XmlTextEntities*
Copy the two *analysis.csc* and *synthesis.csc* files.
Go to the folder:*MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Resources\English\CasSys*
Paste these files.

In general, users copy the graphs used by cascades in the Graphs folder. In fact, it is enough to copy the compiled files (*.fst2*). However, the presence of source graph files (*.grf*) makes it possible to use the language package as a backup of one's work.

1. Go to the folder:
MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Resources\English\Graphs
Create a new folder named *UnitexGettingStarted*.

2. Go to the folder: *MyUnitex\English\Graphs\UnitexGettingStarted*
Copy the folder *Entities*.
Go to the folder: *MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Resources\English\Graphs\UnitexGettingStarted*
Paste this folder.

Neither of the cascades use dictionaries, so we do not need to copy anything to the folder:

MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Resource\English\Dela

Otherwise, if the cascades used dictionaries, see Section 2.4.1, page 155.

2.2 The *scripts* folder

2.2.1 Using the console

The *scripts* folder will contain the Unitex scripts. The scripts can be developed manually, but the Unitex console will help us.

If Unitex is already open, we need to close and reopen it in order to clean the console. After that, everything we do with the menu will be recorded by the console. We will launch our two cascades.

1. We will use the *Text/Open* menu (answering *No* to the question *Do you want to preprocess the text*) to open the file:

MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\jubileePlainText.txt

2. We will use the *Text/ApplyCasSysCascade…* menu, then subsequently double-click the*UnitexGettingStarted* and *XmlTextEntities* folders to open them; we will select the *analysis.csc* cascade and click the *Launch* button (do not build the concordance).[135]
3. We will use the *Text/Open* menu (answering *No* to the question *Do you want to preprocess the text*) to open the file:

MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\JubileePlainText_csc.txt

4. We will use the *Text/ApplyCasSysCascade…* menu, then select the *synthesis.csc* cascade and click the *Launch* button (do not build the concordance).
5. We will use the *Info/Console* menu and copy its content (*CTRL-C*).
6. We will use the *File Edition/New File* menu, then paste the commands recorded in the console.
7. We will use the *Save text* button and save this file with the name s*criptEntities.uniscript*, placed in the folder:

MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Scripts

[135] We will use the *XmlTextEntities* cascade, as we will later parse also the *XmlTextEntities.xml* file.

This file contains eight commands, which look like this:[136]

```
mkdir "C:\Documents\MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\
JubileePlainText_snt"

"C:\Program Files (x86)\Unitex-GramLab\App\UnitexToolLogger.exe" Normalize
"C:\Documents\MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\
jubileePlainText.txt"
"-rC:\Documents\MyUnitex\English\Norm.txt" "--output_offsets=C:\Documents\MyUnitex\
English\Corpus\UnitexGettingStarted\JubileePlainText\JubileePlainText_snt\
normalize.out.offsets" -qutf8-no-bom

"C:\Program Files (x86)\Unitex-GramLab\App\UnitexToolLogger.exe" Tokenize
"C:\Documents\MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\
JubileePlainText.snt"
"-aC:\Documents\MyUnitex\English\Alphabet.txt" -qutf8-no-bom

"C:\Program Files (x86)\Unitex-GramLab\App\UnitexToolLogger.exe" CasSys
"-aC:\Documents\MyUnitex\English\Alphabet.txt"
"-tC:\Documents\MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\
JubileePlainText.snt"
"-lC:\Documents\MyUnitex\English\CasSys\UnitexGettingStarted\XmlTextEntities\analysis.csc" -v
"-rC:\Documents\MyUnitex\English\Graphs\"
"--
input_offsets=C:\Documents\MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\
JubileePlainText_snt\normalize.out.offsets" -qutf8-no-bom

mkdir "C:\Documents\MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\
JubileePlainText_csc_snt"

"C:\Program Files (x86)\Unitex-GramLab\App\UnitexToolLogger.exe" Normalize
"C:\Documents\MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\
JubileePlainText_csc.txt"
"-rC:\Documents\MyUnitex\English\Norm.txt"
"--
output_offsets=C:\Documents\MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\
JubileePlainText_csc_snt\normalize.out.offsets" -qutf8-no-bom

"C:\Program Files (x86)\Unitex-GramLab\App\UnitexToolLogger.exe" Tokenize
"C:\Documents\MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\
JubileePlainText_csc.snt"
"-aC:\Documents\MyUnitex\English\Alphabet.txt" -qutf8-no-bom

"C:\Program Files (x86)\Unitex-GramLab\App\UnitexToolLogger.exe" CasSys
"-aC:\Documents\MyUnitex\English\Alphabet.txt"
"-tC:\Documents\MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\
JubileePlainText_csc.snt"
"-lC:\Documents\MyUnitex\English\CasSys\UnitexGettingStarted\XmlTextEntities\synthesis.csc"
-v "-rC:\Documents\MyUnitex\English\Graphs\"
"--
input_offsets=C:\Documents\MyUnitex\English\Corpus\UnitexGettingStarted\JubileePlainText\
JubileePlainText_csc_snt\normalize.out.offsets" -qutf8-no-bom
```

[136] Users can obtain a different content if the location and names of the folders in question differ. Do not forget this when preparing a script.

2.2.2 File adjustment

We must adjust this file because of differences between the operating systems Windows, MacOs and Unix:

1. We will remove the 56 quotation marks;
2. We will replace the 148 backslashes (\) with slashes (/).[137]
3. We will add two lines at the beginning of the script to create a virtual working folder where the files will be processed one after the other (*CURRENT_WORK_DIR*). The virtual folder and its virtual files will not be stored on the disk, but in random-access memory, which will speed up the processing.

```
CURRENT_WORK_DIR = {CORPUS_WORK_DIR}/{UNIQUE_VALUE}
DuplicateFile -p {CURRENT_WORK_DIR}
```

4. We will delete the two lines starting with *mkdir* command and replace them by these two lines at the beginning of the script, after the previous two inserted lines.

```
DuplicateFile --make-dir {CURRENT_WORK_DIR}/corpus_snt
DuplicateFile --make-dir {CURRENT_WORK_DIR}/corpus_csc_snt
```

2.2.3 Script generalization

In order to make this script more general, we will transform this sequence of commands in the following way:

1. We will add a line (it will become the fifth line) to process a file from our folder.

```
DuplicateFile -i {INPUT_FILE_1} {CURRENT_WORK_DIR}/corpus.txt
```

2. We will replace the ten occurrences of the beginning of the text file name:

   C:/Documents/MyUnitex/English/Corpus/JubileePlainText/jubileePlainText

   with:

```
{CURRENT_WORK_DIR}/corpus
```

3. We will add a line (the penultimate of the script) to save the result file.

```
DuplicateFile -i {CURRENT_WORK_DIR}/corpus_csc_csc.txt {OUTPUT_FILE_1}
```

4. We will remove the six calls to the *UnitexToolLogger* program and the path that leads to it:

   C:/Program Files (x86)/Unitex-GramLab/App/UnitexToolLogger.exe

5. We will replace the ten paths that lead to our personal Unitex working folder:

   C:/Documents/MyUnitex

   with the folder of resources of the package

[137] This is due to the inversion of these symbols between Windows, on the one hand, and MacOs or Unix, on the other. If you are not using Windows, you can therefore ignore this step. You may have a different number of backslashes if the hierarchy of folders on your computer is different.

```
{PACKAGE_DIR}/Resources
```

6. We will modify the two paths leading to our cascades (the two *Cassys* lines) by removing (before *analysis.csc* and *synthesis.csc*):

UnitexGettingStarted/XmlTextEntities/

7. We will add a line at the end of the script, to free up some space before processing the next file in the input folder.

```
DuplicateFile --recursive-delete {CURRENT_WORK_DIR}
```

Our script file (thirteen lines) will now look like this:

```
CURRENT_WORK_DIR = {CORPUS_WORK_DIR}/{UNIQUE_VALUE}

DuplicateFile -p {CURRENT_WORK_DIR}

DuplicateFile --make-dir {CURRENT_WORK_DIR}/corpus_snt

DuplicateFile --make-dir {CURRENT_WORK_DIR}/corpus_csc_snt

DuplicateFile -i {INPUT_FILE_1} {CURRENT_WORK_DIR}/corpus.txt

Normalize {CURRENT_WORK_DIR}/corpus.txt -r{PACKAGE_DIR}/Resources/English/Norm.txt --
output_offsets={CURRENT_WORK_DIR}/corpus_snt/normalize.out.offsets -qutf8-no-bom

Tokenize {CURRENT_WORK_DIR}/corpus.snt -a{PACKAGE_DIR}/Resources/English/Alphabet.txt -
qutf8-no-bom

Cassys -a{PACKAGE_DIR}/Resources/English/Alphabet.txt -t{CURRENT_WORK_DIR}/corpus.snt -
l{PACKAGE_DIR}/Resources/English/CasSys/analysis.csc -v -
r{PACKAGE_DIR}/Resources/English/Graphs/ --
input_offsets={CURRENT_WORK_DIR}/corpus_snt/normalize.out.offsets -qutf8-no-bom

Normalize {CURRENT_WORK_DIR}/corpus_csc.txt -r{PACKAGE_DIR}/Resources/English/Norm.txt
--output_offsets={CURRENT_WORK_DIR}/corpus_csc_snt/normalize.out.offsets -qutf8-no-bom

Tokenize {CURRENT_WORK_DIR}/corpus_csc.snt -
a{PACKAGE_DIR}/Resources/English/Alphabet.txt -qutf8-no-bom

Cassys -a{PACKAGE_DIR}/Resources/English/Alphabet.txt -
t{CURRENT_WORK_DIR}/corpus_csc.snt -
l{PACKAGE_DIR}/Resources/English/CasSys/synthesis.csc -v -
r{PACKAGE_DIR}/Resources/English/Graphs/ --
input_offsets={CURRENT_WORK_DIR}/corpus_csc_snt/normalize.out.offsets -qutf8-no-bom

DuplicateFile -i {CURRENT_WORK_DIR}/corpus_csc_csc.txt {OUTPUT_FILE_1}

DuplicateFile --recursive-delete {CURRENT_WORK_DIR}
```

This script successively launches two cascades on a file and will produce the final result without writing on the disk numerous intermediate files, which are processed in RAM instead. It is the *DuplicateFile* command which allows access to a virtual folder and its virtual files.

2.2.4 Final touches

Since the *corpus.txt* file has already been normalized, the file resulting from the first cascade, *corpus_csc.txt*, is also normalized. Therefore, it is not necessary to do it again, although it would not be wrong. Since the second cascade uses the *corpus_csc.snt* file, we must rename the *corpus_csc.txt* file produced by the first cascade.

We are therefore going to replace the second *Normalize* command which follows the first *CasSys* command, on line 9, by:

```
DuplicateFile -i {CURRENT_WORK_DIR}/corpus_csc.txt {CURRENT_WORK_DIR}/corpus_csc.snt
```

In this case, the offset file is that of the first, and now only, *Normalize* command.

```
Normalize {PACKAGE_DIR}/Resources/corpus.txt
-r{PACKAGE_DIR}/Resources/English/Norm.txt
--output_offsets={CURRENT_WORK_DIR}/corpus_snt/normalize.out.offsets
-qutf8-no-bom
```

We need to replace the *input_offsets* option of the second *CasSys* command with the *output_offsets* option of the *Normalize* command.

```
Cassys -a{PACKAGE_DIR}/Resources/English/Alphabet.txt
-t{PACKAGE_DIR}/Resources/corpus_csc.snt
-l{PACKAGE_DIR}/Resources/English/CasSys/synthesis.csc -v
-r{PACKAGE_DIR}/Resources/English/Graphs/
--input_offsets={CURRENT_WORK_DIR}/corpus_snt/normalize.out.offsets
-qutf8-no-bom
```

With these modifications we obtain the last version of our script.

2.3 Compression

The *scriptEntitiesLingpkg* folder is now complete. The only thing that remains to be done is to create the compressed file. In order to do that we have to open this folder, select the two subfolders, named *Resources* and *Scripts*, compress them together and name the compressed *scriptEntitiesLingpkg.zip* file.[138] We will cut this compressed file and paste it in the *MyUnitex**Scripts**ScriptEntities* folder.

2.4 Another possibility (only for confident users)

2.4.1 The use of dictionaries

2.4.1.1 The default English dictionary

If the cascades use the default English dictionary, *dela-en-public*:

Go to the folder where Unitex is installed:[139]
C:\Users\userName\AppData\Local\Unitex-GramLab
or *C:\Program Files (x86)\Unitex-GramLab\App*
Copy the three files named *dela-en-public.bin*, *dela-en-public.inf* and *dela-en-public.txt*.
Go to the folder:*MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Resources\English\Dela*
Paste these files.

We must add to our script five new lines between the first *Tokenize* command and the first *Cassys* command. The first one for the *Dico* command and the other four for the *SortTxt* commands that create four files *dlf.n*, *dlc.n*, *err.n* and *tags_err.n*.

[138] It must be a *.zip* file, *.rar* files are not accepted.

[139] See Chapter 2, Section 1.1.2, page 7.

Dico -t{CURRENT_WORK_DIR}/corpus.snt -a{PACKAGE_DIR}/Resources/English/Alphabet.txt {PACKAGE_DIR}/Resources/English/Dela/**dela-en-public.bin**
SortTxt {CURRENT_WORK_DIR}/corpus_snt/dlf -l{CURRENT_WORK_DIR}/corpus_snt/dlf.n -o{PACKAGE_DIR}/Resources/English/Alphabet_sort.txt
SortTxt {CURRENT_WORK_DIR}/corpus_snt/dlc -l{CURRENT_WORK_DIR}/corpus_snt/dlc.n -o{PACKAGE_DIR}/Resources/English/Alphabet_sort.txt
SortTxt {CURRENT_WORK_DIR}/corpus_snt/err -l{CURRENT_WORK_DIR}/corpus_snt/err.n -o{PACKAGE_DIR}/Resources/English/Alphabet_sort.txt
SortTxt {CURRENT_WORK_DIR}/corpus_snt/tags_err -l{CURRENT_WORK_DIR}/corpus_snt/tags_err.n -o{PACKAGE_DIR}/Resources/English/Alphabet_sort.txt

2.4.1.2 A private dictionary

If the cascades use a private dictionary of forms, for example the *myProperNameDictionary.dic* dictionary, created in Chapter 5, Section 4.2, page 123.

Go to the folder *MyUnitex\English\Dela*
Copy the two files named *myProperNameDictionary.bin* and *myProperNameDictionary. inf*.
Go to the folder:*MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Resources\English\Dela*
Paste these files.[140]

Just the *Dico* command changes.

Dico -t{CURRENT_WORK_DIR}/corpus.snt -a{PACKAGE_DIR}/Resources/English/Alphabet.txt {PACKAGE_DIR}/Resources/English/Dela/**myProperNameDictionary.bin**

2.4.1.3 Several dictionaries

If the cascades use both the default English dictionary and the private dictionary, the both dictionaries are placed in the same line.

Dico -t{CURRENT_WORK_DIR}/corpus.snt -a{PACKAGE_DIR}/Resources/English/Alphabet.txt {PACKAGE_DIR}/Resources/English/Dela/**dela-en-public.bin** {PACKAGE_DIR}/Resources/English/Dela/**myProperNameDictionary.bin**

2.4.1.4 A morphological dictionary

It is also possible to allow the use of dictionaries in the morphological mode (see Chapter 5, Section 4.5, page 127). For example, if the private dictionary used in the previous section, *myProperNameDictionary.dic*, is to be declared also as a morphological dictionary, its name must be written a second time, preceded by the *-m* prefix.

Dico -t{CURRENT_WORK_DIR}/corpus.snt -a{PACKAGE_DIR}/Resources/English/Alphabet.txt {PACKAGE_DIR}/Resources/English/Dela/dela-en-public.bin {PACKAGE_DIR}/Resources/English/Dela/myProperNameDictionary.bin **-m{PACKAGE_DIR}/Resources/English/Dela/myProperNameDictionary.bin**

In any case, if the user's cascades use dictionaries, the user can always repeat the procedure described in Section 2.2.1, page 150, which consists of applying the cascades through the Unitex interface, copying the console to the editor and adjusting the resulting commands.

[140] Optionaly, you can also copy and paste the *myProperNameDictionary.dic* file and, if it exists, *myProperNameDictionary.txt*, if you want the language package to be also a backup of your work.

2.4.2 The last file from the _csc folder

It is possible to retrieve the last file from the *_csc* folder. This is useful because a bug has crept into the stable version 3.3 of Unitex: the output files are not correct if the cascade annotates categories with graphs, then deletes them with other graphs. This rarely happens, but in order to be sure that we have the correct output, the right file to recover is the last file from the *_csc* folder.

For that:

1. Add, at the beginning of the script, a new *DuplicateFile* line.

```
DuplicateFile --make-dir {CURRENT_WORK_DIR}/corpus_csc_csc
```

2. Modify the output file.

```
DuplicateFile -i {CURRENT_WORK_DIR}/corpus_csc_csc/corpus_0_0.snt {OUTPUT_FILE_1}
```

Since intermediate files are not kept, the number of the last file processed is always 0_0.

3 The command line

3.1 The batch launching file

We will use the *File Edition/New File* menu to create a file that will contain the following command line (it has to be adapted for each computer, depending on where the Unitex software is installed):[141]

```
"C:\Program Files (x86)\Unitex-GramLab\App\UnitexToolLogger.exe" { BatchRunScript -i .\Input
-e -o .\Output -t1 .\scriptEntitiesLingpkg.zip -f -s Scripts\scriptEntities.uniscript }
```

We will save this file, naming it *multiLaunchEntities.bat*, in the folder:

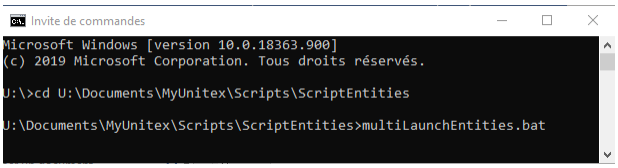
MyUnitex\Scripts\ScriptEntities

3.2 Running the script

We will now open the command prompt window and choose as the working folder:

MyUnitex\Scripts\ScriptEntities

And then launch the *multiLaunchEntities.bat* batch file.



As a result, we will obtain two files (*jubileePlainText.result.txt* and *JubileePlainText.xml.result.txt*) in the folder:

MyUnitex\Scripts\ScriptEntities\Output

[141] A space is obligatory after the opening brace and before the closing brace. The t1 option means that you are working with one core. If your computer has several and your corpus is very large, you can work by parsing the texts of the corpus in parallel, by modifying this option: t2, t3…

3.3 A command line with comments and several possible outputs

The command line above has two drawbacks: errors are not reported and only one file is possible as output. To overcome this, it is possible to launch the files one by one with the command line below which can be saved under the name *launchEntities.bat* in the folder:[142]

MyUnitex\Scripts\ScriptEntities

```
"C:\Program Files (x86)\Unitex-GramLab\App\UnitexToolLogger.exe"
{ SelectOutput --output=on }[143] { InstallLingResourcePackage
-p C:\Documents\MyUnitex\Scripts\ScriptEntities\scriptEntitiesLingpkg.zip
-x $:UnitexPkgResource -v } { RunScript -v
-a INPUT_FILE_1=C:\Documents\MyUnitex\Scripts\ScriptEntities\Input\jubileePlainText.txt
-a CORPUS_WORK_DIR=$:UnitexPkgWork -a PACKAGE_DIR=$:UnitexPkgResource
-a OUTPUT_FILE_1=
C:\Documents\MyUnitex\Scripts\ScriptEntities\Output\jubileePlainText.result1.txt
-a OUTPUT_FILE_2=
C:\Documents\MyUnitex\Scripts\ScriptEntities\Output\jubileePlainText.result2.txt
$:UnitexPkgResource\Scripts\scriptEntities.uniscript } { InstallLingResourcePackage
-p C:\Documents\MyUnitex\Scripts\ScriptEntities\scriptEntitiesLingpkg.zip
-x $:UnitexPkgResource -u -v }
```

If you want to get a single output file, but without comments, you need to remove the *-v* option. It can also be used to have comments with a single output file by removing the *-a OUTPUT_FILE_2* command. Finally, we can also add the commands *-a OUTPUT_FILE_3*, and so on, to have more than two output files.

But the use of this script is not mandatory.

4 An inventory of tag occurrences

In a cascade that processes XML-like input text, the standoff option launches the creation of an inventory of tag occurrences. For example, a standoff cascade can be used after the annotation with two cascades (analysis and synthesis). These standoff cascades are not integrated in the interface version of Unitex and can therefore be launched only by a script. To illustrate their use, we will create a new script by modifying the previously developed script.

The data used for this step is a cascade of graphs (which may possibly consist of only one graph) and a template file for formatting the results.

In order not to lose the results of the previous section (the annotated files), we will create a new folder for the results of the standoff cascade: *StandoffOutput* in the *Scripts* folder:

MyUnitex\Scripts\ScriptEntities

4.1 The standoff cascade

The *standoff* cascade graphs indicate which tags we would like to count. For these graphs we will create a *Standoff* folder in the folder*:*

MyUnitex\English\Graphs\UnitexGettingStarted\Entities

---

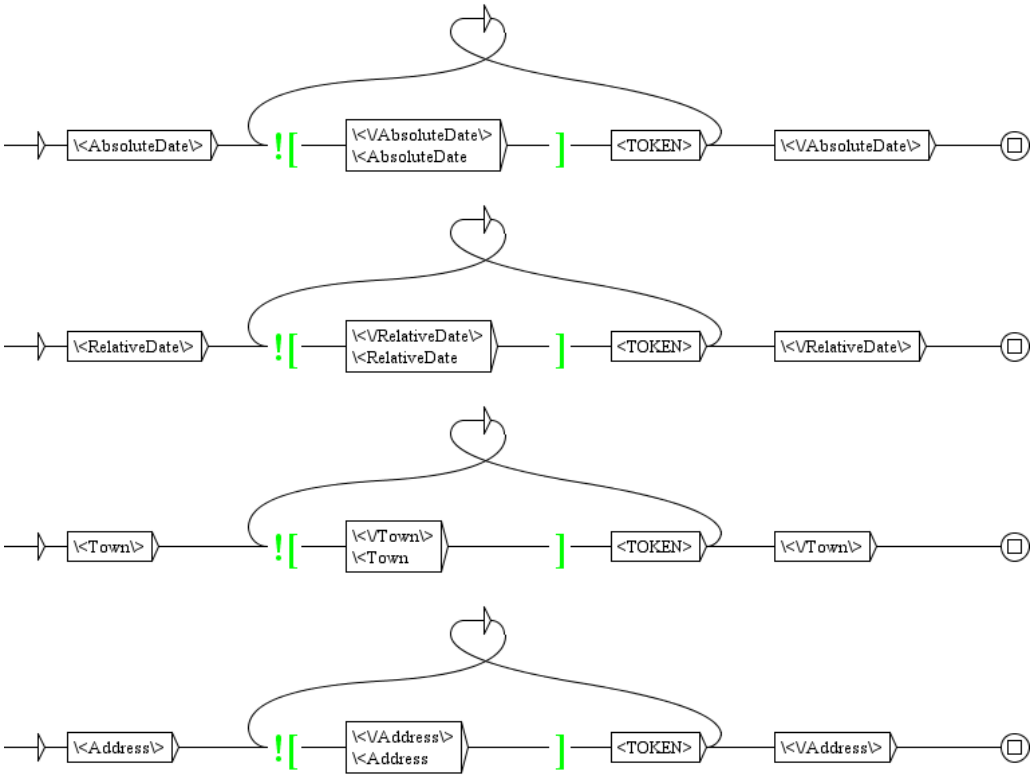
[142] Do not forget to adapt it for your computer, e.g. the paths to Unitex and to your personal Unitex folder.

[143] This script can also be used without comment with the *{ SelectOutput --output=off }* option.

If we want to process the same examples where we have inserted the tags *<AbsoluteDate></AbsoluteDate>*, *<RelativeDate></RelativeDate>*, *<Town></Town>* and *<Address></Address>*,[144] we need four graphs that we can combine into one, with four subgraphs, which we will name *grfStandoff.grf*.[145] The graph and its subgraphs will be recorded in the folder:

MyUnitex\English\Graphs\UnitexGettingStarted\Entities\Standoff

Each subgraph recognizes the start tag, the end tag and the content between them, and at the same time prevents nesting of tags of the same type.



We will now create a cascade that consists of only one graph (*grfStandoff.fst2* in *Merge* mode). We will record it using the name *standoff.csc* in the folder:

MyUnitex\English\Cassys\UnitexGettingStarted\XmlTextEntities

4.2 The template file

The template file sets the design of the report that inventories the tag occurrences. It is divided into three parts, the second part being itself divided into two.

---

[144] Chapter 4, Section 1, page 71 and Section 2, page 84.

[145] This grouping makes it possible to process the text only once, which saves time. There may be a bias in case of nesting.

```
Introduction
Inventory (#LINE… #REST): to be repeated for each type and type+subtype combination
Inventoried item: {TYPE} or {TYPE} <<and {SUBTYPE}>>[146]
Inventoried term: {TERM} and its number of occurrences in the document: {COUNT}
Conclusion
```

We will use the *File Edition* menu to create a new file with the name *standoffPattern.txt* in the folder:[147]

MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Resources\English

```
<xml>
#LINE
      <listAnnotation type="{TYPE}" <<subtype="{SUBTYPE}">>>
#BLOCK
            <term frequency="{COUNT}">{TERM}</term>
#END
      </listAnnotation>
#REST
</xml>
```

4.3 The script

We will use the final version of the script we developed in Section 2.2.4, page 154. We will rename this file under the name s*tandoffScriptEntities.uniscript* in the folder:

MyUnitex\Scripts\ScriptEntities\scriptEntitiesLingpkg\Scripts

We will add in it a line before *Normalize* to indicate the location of the template file:

```
RunScript {PACKAGE_DIR}/Resources/English/standoffPattern.txt -R -o
{CURRENT_WORK_DIR}/patternResult.txt --no_translate_separator
```

This line uses a specific Unitex function *RunScript*, which loads the *standoffPattern.txt* file under the name *patternResult.txt*. The *no_translate_separator* option prohibits replacement of slashes by backslashes.

We will copy the last three lines *DuplicateFile*, *Tokenize* and *Cassys*; we will paste them right after. We will modify them: the working file is now *corpus_csc_csc.snt*; the *standoff* option replaces the *input_offsets* option; and the name of cascade to be used is *standoff.csc*. The result of the standoff cascade, the *corpus_csc_csc_standoff.txt* file, replace the previous output in the ultimate line of the new script.

```
DuplicateFile -i {CURRENT_WORK_DIR}/corpus_csc_csc.txt
{CURRENT_WORK_DIR}/corpus_csc_csc.snt
Tokenize {CURRENT_WORK_DIR}/corpus_csc_csc.snt
-a{PACKAGE_DIR}/Resources/English/Alphabet.txt -qutf8-no-bom
Cassys -a{PACKAGE_DIR}/Resources/English/Alphabet.txt
-t{CURRENT_WORK_DIR}/corpus_csc_csc.snt
-l{PACKAGE_DIR}/Resources/English/CasSys/standoff.csc -v
-r{PACKAGE_DIR}/Resources/English/Graphs/
--standoff={CURRENT_WORK_DIR}/patternResult.txt -qutf8-no-bom
DuplicateFile -i {CURRENT_WORK_DIR}/corpus_csc_csc_standoff.txt {OUTPUT_FILE_1}
```

[146] The text between double angle brackets will only be written in the presence of a subtype.

[147] This example of template file is in XML format, which is not the obligatory format.

4.4 The new linguistic package

We will now copy graphs in the linguistic package.

Go to the folder: *MyUnitex\English\Graphs\UnitexGettingStarted\Entities*
Copy the *Standoff* folder.
Go to the folder: *MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Resources\English\Graphs\UnitexGettingStarted\Entities*
Paste this folder.

As well as the new cascade.

Go to the folder: *MyUnitex\English\Cassys\UnitexGettingStarted\XmlTextEntities*
Copy the *standoff.csc* file.
Go to the folder:*MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Resources\English\CasSys*
Paste this file.

We will create the *scriptEntitiesLingpkg.zip* file in the same way as in Section 2.3, page 155.

4.5 The new batch file

We will save the *multiLaunchEntities.bat* file under the name *multiLaunchStandoffEntities.bat*. We need to change in this file the name of the script and name of the folder for results (bold and underlined):

```
"C:\Program Files (x86)\Unitex-GramLab\App\UnitexToolLogger.exe" { BatchRunScript -i .\Input -e -o .\StandoffOutput -t1
.\scriptEntitiesLingpkg.zip  -f -s Scripts\standoffScriptEntities.uniscript  }
```

As a result of running this script, we will obtain two identical files (*jubileePlainText.result.txt and JubileePlainText.xml.result.txt*) in the folder:[148]

MyUnitex\Scripts\ScriptEntities\StandoffOutput

Note that with the command line with several possible outputs in Section 3.3, page 158, you could merge both the script of Section 2.2.4, page 154, and the script of Section 4.3, page 160, to output the tagged file and the inventory of tag occurrences file.

[148] These two files are identical since the entities are the same in both.

```
<xml>
	<listAnnotation type="AbsoluteDate">
		<term frequency="1">June 2, 2022</term>
		<term frequency="1">February 6, 1952</term>
	</listAnnotation>
	<listAnnotation type="Address">
		<term frequency="1">Queen Elizabeth Street</term>
		<term frequency="1">London Street</term>
	</listAnnotation>
	<listAnnotation type="RelativeDate">
		<term frequency="1">Thursday June 2</term>
		<term frequency="1">June 5</term>
	</listAnnotation>
	<listAnnotation type="Town">
		<term frequency="2">Nearlondon</term>
		<term frequency="1">London</term>
	</listAnnotation>
</xml>
```

4.6 Some additional information

It is possible to add information to the *standoff* file by using specific variables. We can retrieve the name of the file and, if it is in XML format, also information from its header.[149]

4.6.1 The name of the input file

The file name can be included in the template file. To do this, we must add the *{FILENAME}* variable to the *standoffPattern.txt* file, which is in the folder:

MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Resources\English

```
<xml>
<sourceFile>
	<fileName>{FILENAME}</fileName>
</sourceFile>
#LINE
	<listAnnotation type="{TYPE}" <<subtype="{SUBTYPE}">>>
#BLOCK
		<term frequency="{COUNT}">{TERM}</term>
#END
	</listAnnotation>
#REST
</xml>
```

Besides this, we must add the option *-a FILENAME={INPUT_FILE_1}* to the *RunScript* line of the file:

MyUnitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Scripts\standoffScriptEntities.uniscript

[149] It is also possible to retrieve the Unitex version and its data by using a specific Unitex function *VersionInfo*, but we will not go into details here.

```
RunScript {PACKAGE_DIR}/Resources/English/standoffPattern.txt -R -o
{CURRENT_WORK_DIR}/patternResult.txt -a FILENAME={INPUT_FILE_1}
--no_translate_separator
```

After repeating the compression explained in Section 2.3, page 155, and running the script, we again get two identical files. These files contain at the beginning the name of the processed file.

4.6.2 Extracting information from the XML header

When the source file is in XML format, some information located before the body of the text can be interesting to include in the standoff file.

For instance, if we want to retrieve the content inside tags *<title></title>* of the *jubileeXmlText.xml.txt* file, we must:

1. Create a file containing the path of the XML element we want to retrieve.[150] This file, which we will name, for instance *pathTitle.txt*, must be recorded in the folder:

   Unitex\Scripts\ScriptEntities\ScriptEntitiesLingpkg\Resources\English

   ```
   /xml/title
   ```

2. Add to the script just before the *RunScript* line a new line that invokes the specific Unitex function *Unxmlize* that copies the content of a tagged element:

   ```
   Unxmlize -x --selxpath_file={PACKAGE_DIR}/Resources/English/pathTitle.txt -o
   {CURRENT_WORK_DIR}/title.txt -n IGNORE {CURRENT_WORK_DIR}/corpus.txt
   ```

3. Also add the option *-c TITLE={CURRENT_WORK_DIR}/title.txt* to the *RunScript* line:

   ```
   RunScript {PACKAGE_DIR}/Resources/English/standoffPattern.txt -R -o
   {CURRENT_WORK_DIR}/patternResult.txt -a FILENAME={INPUT_FILE_1} -c
   TITLE={CURRENT_WORK_DIR}/title.txt --no_translate_separator
   ```

4. Add the variable *{TITLE}* in the *standoffPattern.txt* file:

   ```
   <xml>
   <sourceFile>
         <fileName>{FILENAME}</fileName>
         <title>{TITLE}</title>
   </sourceFile>
   #LINE
         <listAnnotation type="{TYPE}" <<subtype="{SUBTYPE}">>>
   #BLOCK
               <term frequency="{COUNT}">{TERM}</term>
   #END
         </listAnnotation>
   #REST
   </xml>
   ```

[150] The path structure follows the XPath syntax: this path indicates how XML tags are nested in the document. Since the *<date>* tag is at the same level as the *<title>* tag, the path to this tag will be /xml/date and not /xml/title/date. If the information to be retrieved is in a tag with a specific value of an attribute, for instance *<title type="newspaper">*, the path to it becomes: */xml/title[@type="newspaper"]*.

After repeating the compression process and launching the script, we get two different files, one with empty *<title>{TITLE}</title>* tags and the other with the extracted text, enclosed in *<title></title>* tags. For the *jubileeXmlText.xml* file:

```
<xml>
	<sourceFile>
		<fileName>.\Input\jubileeXmlText.xml</fileName>
		<title>The municipal newspaper of the city of Nearlondon</title>
	</sourceFile>
	<listAnnotation type="AbsoluteDate">
		<term frequency="1">June 2, 2022</term>
		<term frequency="1">February 6, 1952</term>
	</listAnnotation>
	<listAnnotation type="Address">
		<term frequency="1">Queen Elizabeth Street</term>
		<term frequency="1">London Street</term>
	</listAnnotation>
	<listAnnotation type="RelativeDate">
		<term frequency="1">Thursday June 2</term>
		<term frequency="1">June 5</term>
	</listAnnotation>
	<listAnnotation type="Town">
		<term frequency="2">Nearlondon</term>
		<term frequency="1">London</term>
	</listAnnotation>
</xml>
```

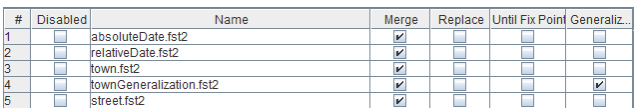
5 Two small remarks on the optimization of a cascade

Let us say we want to launch our cascade over a very large number of texts, all placed in the same directory. Cascade time, even with multiple cores, can become an important parameter.

5.1 Number of graphs

Each time a graph of a cascade is applied, the whole text is reread. If two graphs are not ambiguous or if the first does not prepare data to be processed by the second, it is therefore better to merge them into a single graph which calls two subgraphs.

For instance, the five graphs of the analysis cascade of Chapter 4, Section 1.1, page 71, cannot be merged.



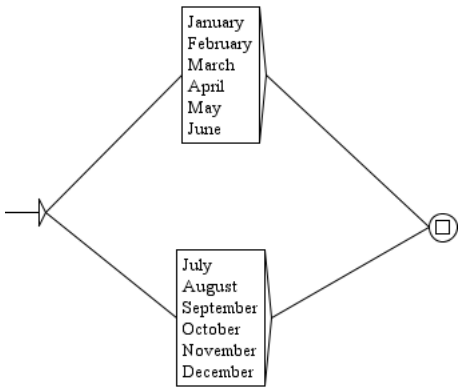
#	Disabled	Name	Merge	Replace	Until Fix Poin	Generaliz...
1	☐	absoluteDate.fst2	☑	☐	☐	☐
2	☐	relativeDate.fst2	☑	☐	☐	☐
3	☐	town.fst2	☑	☐	☐	☐
4	☐	townGeneralization.fst2	☑	☐	☐	☑
5	☐	street.fst2	☑	☐	☐	☐

The two *absoluteDate.grf* and *relativeDate.grf* graphs are ambiguous and they must be applied in this order; the *town.grf* graph uses the previous two; the *townGeneralization.grf* graph uses the previous one; and the *street.grf* graph uses all previous graphs.

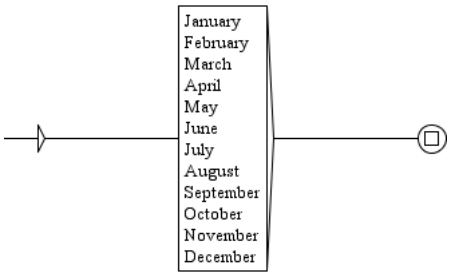
5.2 Number of boxes

It takes a little longer to read two boxes of a graph than one, It is therefore better to merge them, if it is possible.

For instance, the *month.grf* graph below:



could be replaced by the *optimizedMonth.grf* graph:



Note that such adjustments reducing the number of graphs or boxes for optimization may hinder the development of the entire work. It is therefore recommended to do it near the end of the elaboration, only if the number of files to process is large.

Contents

Index

Denis Maurel

is emeritus professor at the University of Tours, member of Lifat (Laboratory of Fundamental and Applied Computer Science of Tours) and associate member of the LLL (Laboratory of Ligerian Linguistics).

He defended his thesis in 1989 under the supervision of Professor Maurice Gross.

His research focuses on NLP (Natural Language Processing), mainly using Unitex, as well as the creation of linguistic resources: Proper Names Processing (*Prolex* project), Named Entity Recognition (*CasEN* project), scientific documents mining (*Istex* and *Abliss* projects).

He taught Unitex in linguistics and digital humanity masters courses, as well as in computer science engineering schools. He gave several tutorials to doctoral students and researchers.

Cvetana Krstev

served as a professor at the University of Belgrade, Faculty of Philology from 1998 to 2020. She is a co-founder and current president of the Language Resources and Technologies Society *JeRTeh*.

Her research focuses on the development of NLP resources and tools. She developed e-dictionaries of Serbian as well as numerous tools for Unitex. These tools, which target Serbian texts, include the named entity recognition system, correction of OCR errors, restoration of diacritics, extraction of definitions, and many more.

She gave tutorials on using Unitex to researchers and PhD students studying Linguistics, Philology and Information Sciences.