PhraseTable2Glossary: Software description

Gr. Thurmair, Linguatec,
V1: 2012-05-15

1. Introduction

This document describes a tool which extracts term and lexicon entries from SMT phrase tables, without further reference to monolingual data. It applies filters to such tables, and builds lexicon entries from the 'good' candidates. Error rates of the tool can be as low as 7.3%, accumulated from source, target, and transfer errors

1.1 Main processing flow

The approach is to apply filters on input records of aligned phrases, whereby formats of different alignment tools (Moses, AnymAlign etc.) are supported as input.

Three filters are applied, as shown in Fig. 1.

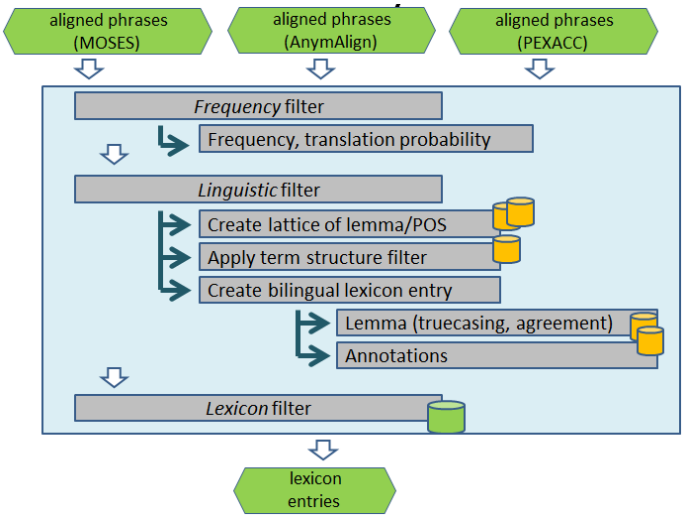

Fig. 1: Operation flow of the P2G system

The filters are:

- A **Frequency filter**: Only phrases with a given frequency and / or translation probability are accepted as term candidates
- A **Linguistic filter**: Only phrases which have certain linguistic properties are acceptable.

  If a candidate passes the linguistic filter, it is brought into the right lexicon form, in terms of lemma creation, assignment of annotations, etc.
- The **Lexicon filter** compares the lexicon entries just produced with a filter resource. This way, candidate entries can be removed which are already known, or are not wanted, or should not be part of the output for some other reason.

1.2 Language coverage

The tool is language-independent in its kernel. But of course it is language-sensitive. This refers to two aspects:

- Language specific data are kept in special files (language resources).
- There are also classes which contain language-specific code.

Adding a new language means to consider both aspects of this.

Language resource files

The tool needs the following information items for each language; note that the tool is bilingual, i.e. at least two languages must be considered at the same time:

System resources:

- normalisation information: Mapping of illegal strings to legal ones (like spelling variants)
- lexicon lookup / lemmatisation information: Assigning lemma and part of speech to a text form
- multiword information: Defining legal part-of-speech patterns which a term must follow
- gender information: needed in multiword terms for adjective agreement
- adjective inflection: building properly inflected adjectives, depending on the head gender

These files are described in detail in Chapter 3 below.

User resources: In addition, there can be another file:

- stoplist information: This file is used to stop term candidates from being displayed. This file is optional, and will be provided by the tool users.

Language-specific code

There is language-specific code, mainly in the CreateLexEntry class, dealing with truecasing, gender issues, and inflection of Adjectives and participles.

This code would have to be adapted if a new language is added to the system.

Tagsets

A special case is the tagset to be used. While the tagset in the resource files can be selected freely, the tags in the language specific codes may have to be adapted. The tags used in the code are explained in the annex.

2. Software structure

2.1 Classes

The software consists of the following classes:

- `PhrT2GloMain`. The main class and a method `doExtraction` which does the overall processing (input and output file)
- `FilterInput`. Interprets the different input formats and applies the frequency filter. Returns term candidates passing this filter. Main method is `createBitext`. Main supported formats are *phrasetable* and *anymalign*.
- `TermExtract`: Does term filtering and term creation for the ones which pass the tests. Main method is `extractTerm`, which controls the whole flow; it calls `createWordlattice` to

create the word lattice, then calls the MWFilter, and finally the CreateLexEntry for source and target term creation. `entryToString` creates the pretty-print output.

- `MWFilter`: Applies the linguistic (multiword) filter to the term candidates. Main method is `runMWFilter` which compares the expanded wordlattice nodes with the MW filter file.
- `CreateLexEntry`: Creates proper lemmata and lexical feature structures for the term candidates. Main method is `createLexEntry`, which creates a proper lemma (`getLemma`), and all the annotations. For multiword terms, this includes the identification of the gender of the head noun, and proper inflection of adjectives and participles in agreement to the head noun. It also does truecasing of the lemmata (`Casing.truecaseLemma`).
- `SimpleTokeniser`: Helper class to do tokenisation and normalisation of the input strings
- `LexInfo`: Helper class to look up lexicons, for lemmatisation and gender assignment
- `Casing`: Helper class to define case information, and do truecasing of the terms
- `FilterExceptions`: Helper class to reduce ambiguities for words which are homographs to function words or irregular verbs. Called to remove disturbing lemmatiser results in `createWordlattice` and proper inflection for participles.
- `Stoplist`: Applies the lexicon filter to the term output. Reads the stoplist and applies it to the terms of the term list.

All classes are in a single package, called p2g.

2.2 Class Hierarchy

The following figure gives an overview of the class hierarchy of the tool. Methods in orange have some language-specific code, the others are language-independent. The hierarchy is shown in Fig. 2.

The `main.loadResources` method initiates all classes and triggers the loading of all resources needed by them.

2.3 Main data formats

Term candidates come in as strings, for source and target side.

Tokeniser creates an ArrayList of <String>, one string per token/textform.

TermExtract creates **Nodes** for each token; nodes consisting of textform, casing, lemma, and tag. While lemma and tag are filled by the lemmatiser, and can have several values, the ArrayList <Node> is expanded by the lexicon lookup into a **word lattice**, which is an ArrayList <ArrayList <Node>>, such that for each [lemma-POS] pair there is a special Node.

From the lattice, the multiword filter takes the best node sequence into again a **termlist** (which is an ArrayList <Node>), with the ‚best' [lemma-POS] pair surviving.

This list is given to the CreateLexEntry, which creates a **LexEntry** object, consisting of a canonical form, lemma, POS, gender, entrytype, headposition, lemmasequence and POS sequence.

This object is passed to the `TermExtract.entrytoString` for a string output.

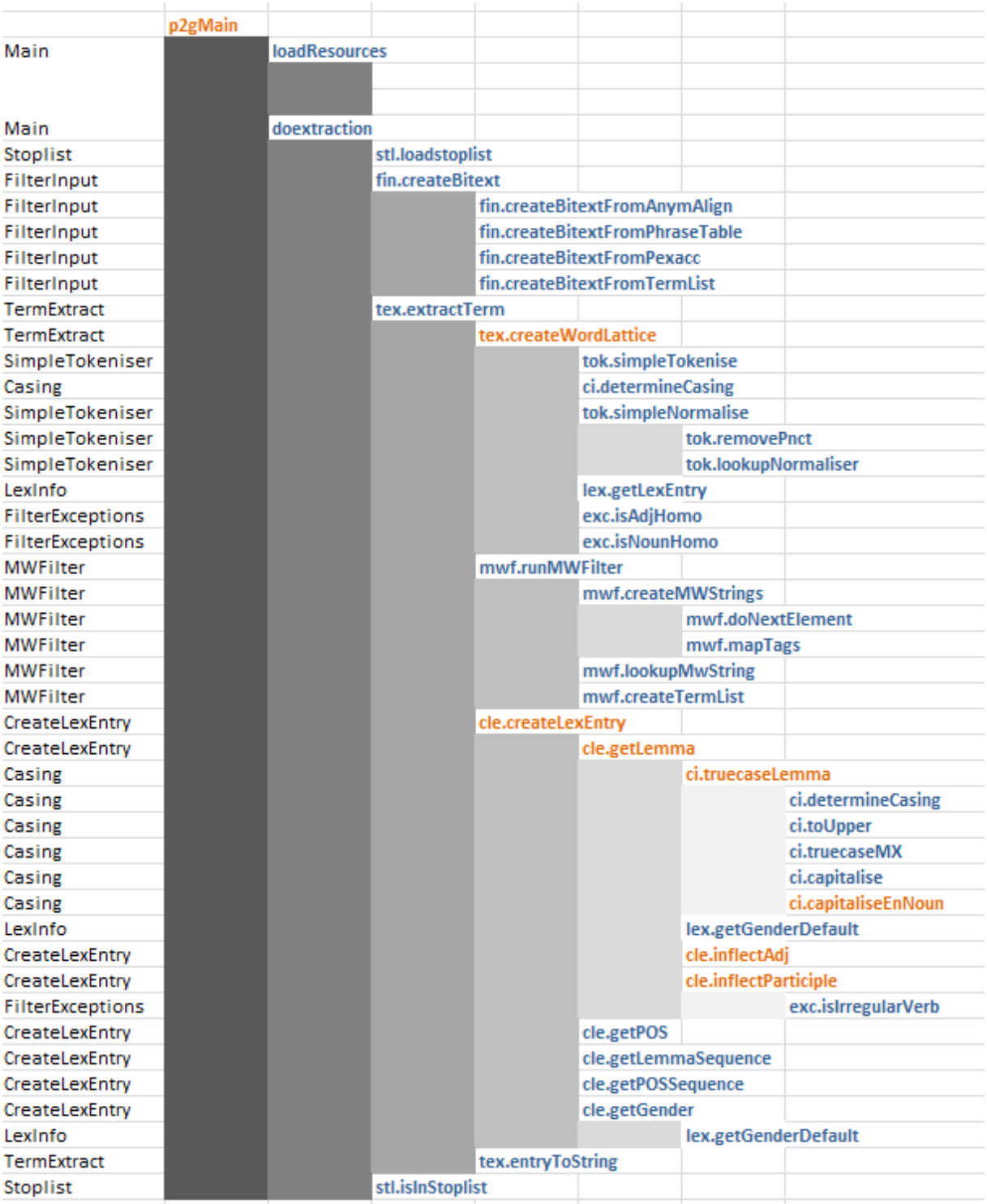

Fig. 2: Call Hierarchy

.

3. Resources and file formats

The P2G tools uses a series of resources. They come in special resource files, and are described in this section. the files are language-specific (cf. the <lg> tag below); supported <lg> tags are: en fr es de it pt.

Note that the software comes *without* these resources! Only example entries in the different files are included.

3.1 LemmaLexicon

This lexicon is used to lemmatise the input words (assign pairs of <lemma, part-of-speech>). It contains possibly all full forms of the given language.

LemmaLexicon	::=	lemmaentry +	
lemmaentry	::=	textform SEP lemma SEP posinfo	
textform	::=	<utf8>	// string, lowercased and normalised
lemma	::=	<utf8>	// string with lemma, lowercased
posinfo	::=	a legal part-of-speech	
SEP	::=	"\t"	

The name of the file is `pg2-<lg>-lemma.txt`. One file per language is required.

The lexicon is read by the **`LexInfo.loadLemmaLexicon`** method.

3.2 Normaliser Lexicon

This lexicon is used to map ‚illegal' strings onto legal ones. It contains spelling variants, locale differences etc.

NormaliserLexicon	::=	normaliseentry +	
normaliseentry	::=	badentry SEP goodentry	
badentry	::=	<utf8>	// string, lowercased
goodentry	::=	<utf8>	// string, lowercased
SEP	::=	"\t"	

The name of the file is `p2g-<lg>-norm.txt`. One file per language is required.

The lexicon is read by the **`SimpleTokeniser.loadNormlex`** method.

3.3 Multiword patterns list

This list is used to filter term candidates with a ‚legal' multiword structure from other input strings. It also defines the head of the multiword.

MultiwordPatterns	::=	mwpattern +	
mwpattern	::=	headposition SEP posinfo (SEP posinfo) *	
headposition	::=	digit	// index of the mw pattern's head, start from 1
posinfo	::=	a legal part of speech	
SEP	::=	"\t"	

The name of the file is `p2g-<lg>-multiword.txt`. One file per language is required.

The lexicon is read by the `MWFilter.loadMWPatterns` method.

3.4 GenderDefault

This list is used to determine the gender of the head of the term; gender must be known in case of agreement construction of adjectival modifiers.

GenderDefault	::=	genderentry +	
genderentry	::=	wordending SEP gender	
wordending	::=	<utf8>	// string of word endings
gender	::=	„mask“ | „fem“ | „neutr“	
SEP	::=	“\t“	

The name of the file is `p2g-<lg>-gender.txt`. One file per language is required (except for EN).

The lexicon is read by the `LexInfo.loadGDDefault` method.

3.5 AdjFlexFile

This is a list which is used to inflect adjectives according to the gender of the head noun. Depending on the gender, the different inflections are looked up in the different columns of the file.

AdjFlexFile	::=	adjflexentry +	
adjflexentry	::=	adjlemma SEP adjflex (SEP adjflex) +	
adjlemma	::=	<utf8>	// the lemma
adjflex	::=	<utf8>	// an inflected form
SEP	::=	“\t“	

The number of inflected forms depends on the language: FR ES IT PT have two (mask / fem), DE has three (mask / fem / neutr), EN does not have this file at all.

The name of the file is `p2g-<lg>-adjflex.txt`. One file per language is required.

The lexicon is read by the `CreatelexEntry.loadAdjFlexFile` method.

3.6 Stoplist

This list is used as a lexical filter. Term candidates are removed if they can be found in the stoplist.

Stoplist	::=	stopentry +	
stopentry	::=	sllemma SEP slpos SEP tllemma SEP tlpos	
sllemma	::=	<utf8>	// lemma of source language
tllemma	::=	<utf8>	// lemma of target language
slpos	::=	<a legal pos>	// pos of source language
tlpos	::=	<a legal pos>	// pos of the target language
SEP	::=	“\t“	

The stop list is optional; if it is not there then simply all term candidates pass the lexicon filter. It is also not a system resource like the other files, but user-dependent.

The name of the file is user-defined. The lexicon is read by the `Stoplist.loadStoplist` method.

4. Differences between LT-P2G and Open-P2G

4.1 Lemmatisation and Lexicon Lookup

a. Classes `P2GBLFDictionary` and `P2Glemmatiser` have been replaced by class `LexInfo`: reads a BLF file; data structure is Array<String> (lemma \t pos) instead of LemmaLinfo.

b. `lemmatiseString` replaced by getLexEntry. Simplified lookup, no partial lemmatisation, no decomposer!, no use of LemmaLinfo. Also no call to trueCasing. Both methods take a Node, return a list of Node.

Replace objects like P2GBLFDictionary, P2Glemmatiser, DefaultGD for sl and tl by LexInfo. Remove Decomposer and LemmaLinfo references.

4.2 GenderDefaulter

a. Class DefaultGD (package Defaulter) replaced by LexLinfo: reads a 2-column file

b. getGenderDefault implemented in LexInfo; same interface. Returns only one value, no Probabilities (class GDProb) needed.

4.3 CreateLexEntry

Remove reference to de-adj-er file and loading. LT-P2G did not use it!! <change!!>

4.4 Filenames

resourcefiles are named differently

Annex: Some tags explained

The code uses the following tags; they occur in `TermExtract.createWordLattice` and in `CreateLexEntry` and `Casing.truecaseLemma`:

Ad	adjective; comprises all subtags of adjectives
AdAt	attributive (inflected) adjective
AdNa	nationality Adjective (must be capitalised in English: „*French*“) (with a fallback cpEnglish for the most common nationality adjectives)
No	noun; can be common (NoCo) or proper (NoPr) noun
NoCo	common noun
NoP	proper noun
VbFQ	inflected past participle of finite verbs (need to be inflected for agreement)
VbFH	inflected present participles of finite verbs (need to be inflected for agreement)
Unk	unknown