DepPattern User Manual

beta version

December 2008

Contents

Chapter 1

DepPattern: A Grammar Based Generator of Multilingual Parsers

DepPattern toolkit is a linguistic package providing a grammar compiler, PoS taggers, and dependency-based parsers for several languages.

The compiler, named *compi-beta*, was implemented in Ruby. It generates parsers in PERL from DepPattern grammars. To write formal grammars using the formalism ”DepPattern”, please, look up the tutorial.

In addition, DepPattern is provided with parsers for 5 languages: English, Spanish, Galician, French, and Portuguese. The parsers were implemented in PERL and are stored in the directory “parsers”. Their ancestor is Multilingua, a more basic parser without morphological features. The parsers take as input PoS tagged text. To tag texts, DepPattern uses either Tree-Tagger or Freeling. Treetagger is provided by the distributed package. Freeling must be previously installed. In the directory “parsers”, there is also a test parser, which was generated by compi-beta from a test grammar, stored in the “grammars” directory.

1.1 Contributions

Pablo Gamallo Otero and Isaac González
Grupo Gramática do Espanhol
University of Santiago de Compostela
Galiza, Spain
`pablo.gamallo@usc.es`

1.2 Supported Languages

`The distributed version of DepPattern includes software to analyze 5 languages: English, Spanish, Galician, French, and Portuguese.`

- `English texts are analysed using either tree-tagger-english or Freeling (en.cfg).`
- `Spanish texts are analysed using either tree-tagger-spanish or Freeling (es.cfg)`
- `Galician texts are analysed using either tree-tagger-galician or Freeling (gl.cfg)`
- `Portuguese texts are analysed using tree-tagger-portuguese or Freeling (pt.cfg)`
- `French texts are analysed using only tree-tagger-french`

`Given that both Treetagger and Freeling support other languages than those listed above, e.g., Catalan, Bulgarian, German, Italian, ...DepPattern can be easily adapted to them.`

1.3 Requirements

To install DepPattern you'll need:

- A typical Linux box with usual development tools:
  - bash
  - perl
- In addition, you must install ruby. To do it, you can try as root:
  `apt-get install ruby`
- Optionally, you may install Freeling:
  http://garraf.epsevg.upc.es/freeling/

1.4 Installation

As the distributed package only contains both intepretable code (perl and ruby) and binary files (treetagger), the installation procedure is very simple.

- download of DepPattern-beta.tgz package in LINK.
- Decompress the file in any directory:
  `tar xzvf DepPattern.tgz`
- move to DepPattern directory:
  `cd DepPattern`
- run the following comand (it is not necessary to be the root):
  `sh install-DepPattern.sh`

DepPattern-beta.tgz package is also provided with the parameter files required by tree-tagger.

1.5 Executing

A simple main program, dp.sh, is included in the package to execute either an existing parser or a generated parser from a DepPattern grammar. The syntax of dp.sh is the following:

```
dp.sh  <type_of_output> <tagger> <lang> <file> [parser] [grammar]

    type_of_output= -a (dependency analysis), -c (correct tagged text)
    tagger=freeling, treetagger
    language=gl, es, en, pt, fr
    file=path of the file input
             parser=path of the parser, or name of the parser generated from grammar
             grammar=path of the file grammar
```

If you haven't installed Freeling, don't choose 'freeling'.

Next chapter describes in more detail the usage of dp.sh.

1.6 Input File

The input file must be in plain text format. File codification must be ISO-8859-1. In the next version, it will be possible to use files codified in Unicode.

1.7 Grammar File

The file containing the grammar must be in plain text format. Below, you'll find a toy example of a grammar with 4 dependency-based rules:

```
AdjnR: NOUN ADJ
Agr:  number, genre
SpecL: DT NOUN
Agr:  number, genre
SubjL: NOUN [ADV]* VERB
Agr:  number
DobjR: VERB [ADV]* NOUN
```

To learn more about DepPattern formalism, look up the tutorial in the doc directory.

1.8 Options for Different Output Formats

The output is in raw plain text. In further versions, we'll provide more elaborate output extensions (xml, html, ...). By now, we provide 3 different ways of visualizing the resulting data: basic representation of dependency analysis (-a), full representation of dependency analysis (-fa), and PoS tag representation (-c).

1.8.1 Analyser (-a)

Option -a means that the dp.sh generates a file with a dependency-based analysis. Each analysed sentence consists of two elements:

1. a line containing the POS tagged lemmas of the sentence. This line begins with the tag SENT. The set of tags used here are listed in file TagSet.txt. All lemmas are identified by means of a position number from 1 to N, where N is the size of the sentence.

2. All dependency triplets identified by the grammar. A triplet consists of: (relation;head_lemma;dependent_lemma)

For instance, the sentence "I am a man." generates the following output:

```
SENT::<I_PRO_0_<number:0|lemma:I|possessor:0|case:0|genre:0|person:0|politeness:0|type:P|token:I|>
am_VERB_1_<number:0|mode:0|lemma:be|genre:0|tense:0|person:0|type:S|token:am|>
a_DT_2_<number:0|lemma:a|possessor:0|genre:0|person:0|type:0|token:a|>
man_NOUN_3_<number:S|lemma:man|genre:0|person:3|type:C|token:man|> ._SENT>
(SubjL;be_VERBF_1;I_PN_0)
(SpecL;man_NOM_3;a_DT_2)
(DobjR;be_VERBF_1;man_NOM_3)
```

The set of dependency relationships used by the 5 grammars can be consulted and modified in the corresponding configuration file: src/dependencies.conf. Morpho-syntactic information is provided by a POS tagger, either tree-tagger or freeling.

1.8.2 Analyser with full representation (-fa)

Option -fa yields and analiser-based format enriched with full morpho-synactic information about each head and dependent word.

1.8.3 Corrector (-c)

Option -c allows dp.sh to generate a file with the same format as the input (i.e., a tagged text). The aim is to use specific syntactic rules to make morpho-syntactic corrections. This option is useful to identify and correct systematic errors of PoS taggers using grammatical rules. More information can be found in Chapter 2.2 of the Tutorial.

1.9 Extensions

1.9.1 CoNLL Output File Format

It is also possible to get an output file with the format defined by CoNLL-X, inspired by Lin (1998):

Lin, D., 1998. Dependency-based Evaluation of MINIPAR. In Proceedings of the Workshop on the Evaluation of Parsing Systems, First International Conference on Language Resources and Evaluation. Granada, Spa in. 12

This format was adopted by the evaluation tasks defined in CoNLL.

To get this output format file, you have to run ./scripts/saidaCoNLL.perl taking as input the output of dp.sh with -a. This is also possible to generate a CoNLL ouput format with a full representation. To do it, you have to run ./scripts/saidaCoNLL-fa.perl taking as input the output of dp.sh with -fa.

1.9.2 Cooccurrences file

The output of the parser can be used to build a cooccurrences file. This file contains all coocurrences between lemmas and dependency contexts. It consists of 3 columns:

context lemm frequency

To get this cooccurrence information, run ./scripts/contextsDep.perl taking as input the output file with the default parsers and flag -a.

1.9.3 Precluding Iteration

The by-default parsing algorithm consists of the following iterative process: rules are applied sequencially in an iterative process. The process stops when no rule can be applied. However, the linguist can choose an algorithm where iteration is precluded. The parsing algorithm without iteration consists in applying rules sequencially; the process stops when the parser achieves the last rule to be applied. In order to set up the restrictive algorithm without iteration, copy the files within 'src/iterationOFF' into the 'src' folder:

```
 cp  src/iterationOFF/*  src/.
```

1.10 Porting to other platforms

Chapter 2

Test samples

DepPattern package is provided with a main program which allows the user to process an input text to obtain either a dependency analysis or a PoS tagged text.

The main program is called with the command:

```
dp.sh <flag> <tagger> <lang> <input_file> [parser_name] [grammar]
```

If the optional arguments (parser_name and grammar) are not specified, the default parser is searched in the 'parsers' directory.

The dp.sh program reads from standard input and prints results to standard output, with plain format.

2.1 Usage example

Assuming we have the folowing input file mytext.txt:

```
I have a dream.
```

the command 'dp.sh' provides us with several options to generate a dependency-based analysis.

2.2 Usage of default parsers

If we want to use the default parsers included in the package, the command to be use is the following:

```
dp.sh  -a treetagger en mytext.txt > mytext.dep
```

This command generates a simple dependency-based representation (option -a), making use of the english (en) treetagger. The expected result is:

```
SENT::<I_PRO_0_<number:0|lemma:I|possessor:0|case:0|genre:0|person:0|politeness:0|type:P|token:I|>
have_VERB_1_<number:0|mode:0|lemma:have|genre:0|tense:0|person:0|type:A|token:have|>
a_DT_2_<number:0|lemma:a|possessor:0|genre:0|person:0|type:0|token:a|>
dream_NOUN_3_<number:S|lemma:dream|genre:0|person:3|type:C|token:dream|>._SENT>
(SubjL;have_VERB_1;I_PRO_0)
(SpecL;dream_NOUN_3;a_DT_2)
(DobjR;have_VERB_1;dream_NOUN_3)
```

The output consists of 4 lines. The first one is the input of the parsing strategy. It starts by 'SENT::' and contains 5 columns separated by a space. Each column is provided with the morpho-syntactic information assigned to each token of the input text. The remaining 3 lines show the dependency-based analysis. For instance, the second line represents the Subject dependency between pronoun ‘‘I’’ and verb ‘‘have’’.

If we would like to parse a spanish text, we should use the following command:

```
dp.sh  -a treetagger es mytext-es.txt > mytext-es.dep
```

where ’mytext-es’ stands for the name of a file containing a text in Spanish. The remaining languages are called with ‘‘fr’’ (french), ‘‘pt’’ (portuguese), and ‘‘gl’’ (galician).

Instead of treetagger, we can use freeling:

```
dp.sh  -a freeling en mytext.txt > mytext.dep
```

Freeling is not provided with the DepPattern package. It must be previously installed. The configuration files should be in ’/usr/local/share/FreeLing/config/’, which is the by default directory in the standard installation.

2.3 Using a specific parser

If we are provided with a specific DepPattern parser, the command ’dp.sh’ can be called using the path to this parser:

```
dp.sh  -a treetagger en mytext.txt user_parser  > mytext.dep
```

where ’user_parser’ is the path to retrieve an available parser. The text in mytext.txt will be analysed with such a parser.

2.4 Using a parser just compiled from an user grammar

If we have defined an user grammar following the DepPattern requirements, the command ’dp.sh’ could be the following:

```
dp.sh  -a treetagger en mytext.txt new_parser user_grammar.txt > mytext.dep
```

Here ’new_parser’ is the name of the parser just generated using both a DepPattern grammar (’user_grammar’) and the DepPattern compiler (Compi-beta). If you are not provided with a DepPattern grammar, you can find a testing one in the ’grammars’ directory. This grammar can be compiled to generate a new parser, ’parser_test’, which can be used to analyse the input file, ’mytext.txt’ as follows:

```
dp.sh  -a treetagger en mytext.txt parser_test grammars/grammar_test.txt > mytext.dep
```

Chapter 3

System Modules

3.1 Pipeline architecture

A DepPattern parser file is a Perl script taking as input the result of translating the output of either Treetagger or Freeling into a new file with a shared layout.

In order to analyse an English text stored in the input file 'mytext.txt', we need the following scripts:

- a Perl script containing the DepPattern parser (for instance, 'parser-en'.
- the command required to run a PoS tagger, for instance 'tree-tagger-english', which use the English parameters trained with Treetagger.
- the script 'ChangeTreetagger-en.perl', which is used to change the output of 'tree-tagger-en into a new file likely to be read by 'parser-en'.

In fact, the following command:

```
dp.sh  -a treetagger en mytext.txt parser-en > mytext.dep
```

generates the following pipeline:

```
cat  mytext.txt | tree-tagger-english | scripts/AdapterTreetagger-en.perl | parser-en.perl -a > m
```

So, to analyse a plain text, we'll need to organise 3 processes in a pipeline, i.e., a chain of processing elements, arranged so that the output of each element is the input of the next.

When no parser is available, we can generate it from a DepPattern grammar (e.g., 'user_grammar.txt'). So the following command:

```
dp.sh  -a treetagger en mytext.txt parser-en user_grammar.txt > mytext.dep
```

generates the following pipeline:

```
ruby compi-beta.rb user_grammar.txt parser-en
cat  mytext.txt | tree-tagger-english | scripts/AdapterTreetagger-en.perl | parser-en.perl -a > m
```

The grammar compiler 'compi-beta.rb' was developped, in Ruby, by Isaac González. To build well-formed DepPattern grammars, look up the corresponding tutorial in 'doc'.

3.2 PoS taggers

The first process of our pipeline architecture is PoS tagging. Up to now, a DepPattern parser is able to process any text tagged with the following 9 PoS taggers:

- tree-tagger-english (= 'treetagger en')
- tree-tagger-spanish (= 'treetagger es')
- tree-tagger-french (= 'treetagger fr')
- tree-tagger-portuguese (= 'treetagger pt')
- tree-tagger-galicien (= 'treetagger gl')
- analyzer -f /usr/local/share/FreeLing/config/en.cfg (= 'freeling en')
- analyzer -f /usr/local/share/FreeLing/config/es.cfg (= 'freeling es')
- analyzer -f /usr/local/share/FreeLing/config/gl.cfg (= 'freeling gl')
- analyzer -f /usr/local/share/FreeLing/config/pt.cfg (= 'freeling pt')

3.3 Changing Treetagger and Freeling PoS tags into a common tagset

The second process of the pipeline is to translate the PoS tags of Treetagger and Freeling into a new tagset interpretable by DepPattern parsers. As we used 8 PoS taggers, we need 8 'adapters':

- AdapterTreetagger-en.perl (= 'treetagger en')
- AdapterTreetagger-es.perl(= 'treetagger es')
- AdapterTreetagger-fr.perl (= 'treetagger fr')
- AdapterTreetagger-pt.perl (= 'treetagger pt')
- AdapterTreetagger-gl.perl (= 'treetagger gl')
- AdapterFreeling-en.perl (= 'freeling en')
- AdapterFreeling-es.perl (= 'freeling es')
- AdapterFreeling-gl.perl (= 'freeling gl')

To process a new language supported by either Treetagger or Freeling, we only need to create a new 'Adapter'. This is a very easy task provided that the tagset of the input PoS tagger is available. In addition, we also need the tagset required by DepPattern, which is available at 'docs/tutorialDepPattern.pdf'.

Le't see an example. The sentence 'I have a dream' is PoS tagged by 'tree-tagger-english' as follows:

```
I       PP      I
have    VBP     have
a       DT      a
dream   NN      dream
.       SENT    .
```

This tagged text is translated by AdapterTreetagger-en.perl into:

```
I       genre:0|lemma:I|number:0|person:0|politeness:0|possessor:0|tag:PRO|token:I|type:P|
have    genre:0|lemma:have|mode:0|number:0|person:0|tag:VERB|tense:0|token:have|type:A|
a       genre:0|lemma:a|number:0|person:0|possessor:0|tag:DT|token:a|type:0|
dream   genre:0|lemma:dream|number:S|person:3|tag:NOUN|token:dream|type:C|
```

This is the input format expected by any DepPattern parser.
On the other hand, if the sentence is tagged with freeling-en ('analyzer -f en.cfg'), then we obtain:

```
I i NN
have have VBP
a a DT
dream dream NN
. . Fp
```

This tagged text is translated by AdapterFreeling-en.perl into:

```
I       genre:0|lemma:i|number:S|person:3|tag:NOUN|token:I|type:C|
have    genre:0|lemma:have|mode:0|number:0|person:0|tag:VERB|tense:0|token:have|type:A|
a       genre:0|lemma:a|number:0|person:0|possessor:0|tag:DT|token:a|type:0|
dream   genre:0|lemma:dream|number:S|person:3|tag:NOUN|token:dream|type:C|
```