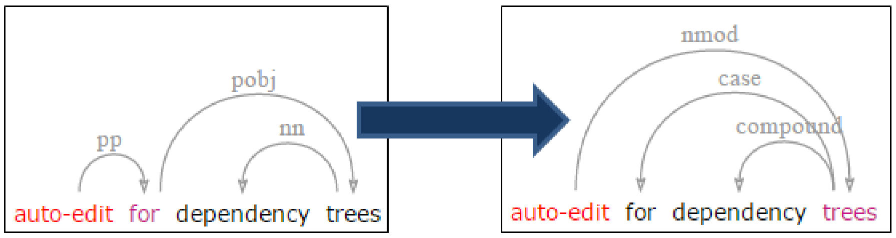
pp
pobj
nn
auto-edit for dependency trees
nmod
case
compound
auto-edit for dependency trees

DepEdit
User Guide – Version 2.3.1

title:	DepEdit User Guide
software version:	2.3.1
guide version:	2.3.1
date:	2021-01-13

author:	Amir Zeldes
e-mail:	amir.zeldes@georgetown.edu
homepage:	http://corpling.uis.georgetown.edu/depedit/

Introduction

DepEdit is a simple configurable tool for manipulating dependency trees, written for Python 2.X and 3.X. To run it you need Python, a configuration file describing the manipulations (see Configuration), and an input file in the CoNLLX or CoNNLU 10 column dependency format (see Format; there is also limited support for other configurations). Basic usage is either file by file, or using a glob pattern (e.g. *.conll10), in which case output files are created with a configurable suffix such as '.depedit' before the extension:

```
> python depedit.py -c config_file.ini INPUT.conllu > OUTPUT.conllu
```

```
> python depedit.py -c config_file.ini *.conllu
```

Additional arguments can be specified:

-d, --docname	Adds a comment # newdoc id =... at the start of each output file
-s, --sent_id	Add running sentence ID comments, e.g. # sent_id = …-1

Batch mode options: (when using glob style, *.conllu input)

-o OUTDIR, --outdir OUTDIR	Output directory in batch mode
-e EXTENSION, --extension EXTENSION	Extension for output files in batch mode
-i INFIX, --infix INFIX	Infix to denote edited files in batch mode

Other optional arguments:

-k {supertoks,comments,both}, --kill {supertoks,comments,both}	Remove supertokens/comments in output
-q, --quiet	Do not output warnings and messages
--stepwise	Output sentence repeatedly after each step (useful for debugging)
--version	show program's version number and exit

DepEdit can also be imported as a module into other projects to preprocess dependency trees (see Importing as a module).

Format

DepEdit uses the 10 column[1] version of the CoNLL dependency format or the more elaborate CoNLLU format from the Universal Dependencies project, with some specific names given to fields in the configuration files. There have been different ways of using the CoNLL format in different projects, e.g. whether or not each second column in a pair is used for gold vs. prediction or some other purpose, fine/coarse grained tags (in MALT), lemmas in the third column, non-projective vs. projective dependencies, etc. DepEdit uses the following column names/mappings, though you may use these column names to run the script even if your own use differs. The following two examples show a minimal and elaborate use of the columns.

Basic input:

The 10 columns are tab-separated, sentences are separated by a blank line.

1. ***num*** - Token ID within sentence
2. ***text*** **-** Token text
3. ***lemma*** (empty denoted by underscore below)
4. ***pos*** - Part of speech
5. ***cpos*** – An alternative, 'coarse' or other language specific POS tag
6. ***morph*** – morphological features
7. ***head*** **–** the head or 'parent' token ID dominating this token
8. ***func*** – dependency function

9.-10. – reserved for alternate trees with multiple parentage or miscellaneous features (see below)

```
1   Wikinews        _   NP    _   _   2   nsubj   _   _
2   interviews      _   VVZ   _   _   0   root    _   _
3   President       _   NN    _   _   2   dobj    _   _
4   of              _   IN    _   _   3   prep    _   _
5   the             _   DT    _   _   7   det     _   _
6   International   _   NP    _   _   7   amod    _   _
7   Brotherhood     _   NP    _   _   4   pobj    _   _
8   of              _   IN    _   _   7   prep    _   _
9   Magicians       _   NPS   _   _   8   pobj    _   _

1   Wednesday       _   NP    _   _   0   root    _   _
2   ,               _   ,     _   _   0   punct   _   _
3   October         _   NP    _   _   4   nn      _   _
4               9   _   CD    _   _   1   appos   _   _
5   ,               _   ,     _   _   0   punct   _   _
6            2013   _   CD    _   _   3   tmod    _   _
```

[1] Starting in version 1.5.1, DepEdit also accepts an 8 column format without the last columns for multiple parentage/function. The last two columns can simply be omitted in such cases.

More elaborate input

This German example also encodes lemmas and morphology in the format, using column 2 (***lemma***) and column 6 (***morph***). A second POS column can also be used in column 5 (***cpos***), which is sometimes used for coarse grained or some alternative POS tag set.

```
1   Die           die          ART     ART     ART.Def.Nom.Pl.*            2   DET    _   _
2   Jugendlichen  Jugendliche  NN      NN      N.Reg.Nom.Pl.*              5   SUBJ   _   _
3   in            in           APPR    APPR    APPR                        2   PP     _   _
4   Zossen        Zossen       NN      NN      N.Name.Dat.Sg.Neut          3   PN     _   _
5   wollen        wollen       VMFIN   VMFIN   VFIN.Mod.3.Pl.Pres.Ind      0   S      _   _
6   ein           eine         ART     ART     ART.Indef.Acc.Sg.Neut       7   DET    _   _
7   Musikcafé     Café         NN      NN      N.Reg.Acc.Sg.Neut           5   OBJA   _   _
```

Super tokens

You can use CoNLLU style 'super tokens' with hyphenated IDs as in the example below. These are not edited by DepEdit, but are simply preserved and printed in the output unchanged:

```
# text = Didn't you?
1-2     Didn't   _      _       _      _               _   _        _   _
1       Did      do     VERB    VBD    SpaceAfter=No   0   root     _   _
2       n't      not    ADV     RB     _               1   advmod   _   _
3       you      you    PRON    PRP    _               1   nsubj    _   _
4       ?        ?      PUNCT   SENT   _               1   punct    _   _
```

Ellipsis tokens

The CoNLLU format allows the insertion of ellipsis tokens that help to create a more standard syntax tree, but which are not actually attested in the original text. These tokens carry decimal IDs, such as 10.1 in the example below. Note that for distance calculation purposes (see Relations below), tokens 10 and 11 are still considered distance=1, and that for ellipsis tokens themselves, position is rounded down (so 10.1 is also 1 token away from token 11).

```
   1  It          it          PRON    PRP    _    4   nsubj      4:nsubj       SpaceAfter=No
   2  's          be          AUX     VBZ    _    4   cop        4:cop         _
   3  more        more        ADV     RBR    _    4   advmod     4:advmod      _
   4  compact     compact     ADJ     JJ     _    0   root       0:root        SpaceAfter=No
   5  ,           ,           PUNCT   ,      _    8   punct      8:punct       _
   6  ISO         iso         NOUN    NN     _    8   compound   8:compound    _
   7  6400        6400        NUM     CD     _    6   nummod     6:nummod      _
   8  capability  capability  NOUN    NN     _    4   list       4:list        _
   9  (           (           PUNCT   -LRB-  _   10   punct      10.1:punct    SpaceAfter=No
  10  SX40        SX40        PROPN   NNP    _    8   parataxis  10.1:nsubj    _
10.1  has         have        VERB    VBZ    _  _    _          8:parataxis   CopyOf=-1
  11  only        only        ADV     RB     _   12   advmod     12:advmod     _
12   3200      3200    NUM     CD      _    10   orphan    10.1:obj      SpaceAfter=No
13   )         )       PUNCT   -RRB-   _    10   punct     10.1:punct    SpaceAfter=No
```

Sentence annotations

Comments and other sentence annotations following the conllu format can appear before the first token of a sentence, as follows:

```
# s_type = decl
1   The      the      DT    DT    _         2   det     _   _
2   people   people   NNS   NNS   _         3   nsubj   _   _
3   go       go       VVP   VBP   _         0   root    _   _
```

This is interpreted as an annotation of the sentence, in this case s_type=decl means sentence type is “declarative”. Sentence annotations can be used in DepEdit conditions and applied to sentences as a result of rules (see below).

Biplanar trees

Some data sources make use of the last two columns to draw secondary edges, such as external subjects for infinitives (xsubj in Stanford Typed Dependencies) or other relations. In such cases, column 9 gives the secondary head (***head2***) and column 10 gives the function for that edge (***func2***). DepEdit exposes these columns partly by using the head2 and func2 fields, but does not directly model the second tree structure. As a result, you may use head2 and func2 in conditions and actions, but not in relations (see below).

Configuration

The manipulations carried out by the script are defined in a configuration file. This is a simple text file with one instruction per line and optional blank lines and comments (beginning with ';' or '#'). Each instruction contains 3 columns, as in the following example:

config_file.ini

```
;Connect nouns to a preceding article or possessive pronoun with the 'det' function
pos=/DT|PRP\$/;pos=/NNS?/      #1.#2           #2>#1;#1:func=det

;Change to-infinitive from aux to mark
text=/^[Tt]o$/&func=/aux/      none            #1:func=mark
```

Column 1: node definitions

The first column describes the tokens to be matched using regular expressions.

- Constraints are given as regular expressions over the fields:
  - ***num*** (column 1 of CoNLL format)

    - ***text*** (column 2)
    - ***lemma*** (column 3)
    - ***pos*** (column 4, alias ***upos, upostag***)
    - ***cpos*** (column 5, alias ***xpos, xpostag***)
    - ***morph*** (column 6, alias ***feats***)
    - ***head*** (column 7) – this is the literal parent token's ID number. Mostly useful when matching roots (head=/0/)
    - ***func*** (dependency function, column 8, alias ***deprel***)
    - ***head2*** (secondary head, for enhanced trees, alias ***deps***)
    - ***func2*** (secondary function, for enhanced trees, alias ***misc***)
    - ***position*** – this is a special constraint which does not correspond to any column, but indicates the token's position in the sentence. Possible values: *first*, *last*, and *mid*, matching the first token, last token, or neither first not last respectively
- Multiple tokens are separated by ';'
- You can specify multiple criteria using '&', as in the second rule
- You may specify **negative criteria** using !=, e.g. lemma!=/able/
- Constraints on sentence annotations are applied like this: #S:s_type=/decl/. Note that the operator to use with such definitions is '>' (see below).
- You can use **capturing groups** in parentheses, which will be referenceable in the actions (third) column as $1, etc.

Column 2: relation definitions

The middle column defines relationships between tokens. It refers to each token in the definition by number (#1, #2...) and specifies:

- **Adjacency** (.): #1.#2 means the first token in column 1 is followed by the second (see note on Ellipsis Tokens above)
- **Distance** (.n or .n,m): #1.4#2 means 4 tokens distance, and #1.1,4#2 means a distance of 1-4. You can also use the shorthand #1.*#2 (indirect precedence, which is the same as #1.1,1000#2.
- **Parentage** (>): #1>#2 means the first token in column 1 is the head of the second token (note: this only applies to the main tree in biplanar input; head2 information is **not** used to establish parentage). This operator is also used for sentence annotations, i.e. the sentence annotation node is the parent of a token in the sentence, #1>#2, where the first definition is of the type #S:x=y.
- **Column identity** (*field*==): in addition to a distance/parentage constraint, two nodes may *also* specify value identity constraints. For example, #1:text==#2 means that #1 and #2 must have exactly the same text (replace 'text' with other fields as needed)

- If the instruction refers to only one token, as in the second example, the middle column says 'none'.

Column 3: action definitions

The third column specifies what to do if a rule matches:

- Change a property of token:
    - ***text***
    - ***lemma***
    - ***pos*** or ***cpos***
    - ***func*** or ***func2*** – dependency functions
    - ***morph*** – the morphological analysis
    - ***head*** or ***head2***[2]
- Add a key-value pair to a complex property and resort key-values:
    - #1:morph+=Gend=Fem (this adds Gend=Fem to the morphological features at the alphabetically appropriate position, preserving existing annotations)
- Make some token in the definitions the head of another: #1>#2
- Add a sentence annotation with the special pointer #S:
    - #S:new_anno_name=somevalue
- You can refer back to values in **capturing groups** from the first column by using the number of that group, e.g. $1:
    - text=/(.*)/&pos=/IN/ … #1:func=prep_$1
- You can also convert the contents of $1, $2 etc. to lower or upper case by using $1L (the contents of $1, in lower case), or $1U (for upper case)
- You can use an equals sign ('=') in the actions column, so the following works as expected (only the first '=' separates the key and value):
    - pos=/NEG/ … #1:morph=Polarity=Negative
- **The special instruction 'last'** makes this rule the last rule to apply to a sentence if it is matched, e.g. the following means 'set the lemma to NONE and stop processing this sentence':
    - #1:lemma=NONE;last

[2] Changing head and head2 to a literal number is supported, and is mainly useful for setting them to 0 (root). If you want to rewire the primary head of some token to be another token, use a dominance directive instead (#1>#2).

Variables

From version 2.3 onwards, it is also possible to define variables for frequently used (parts of) regular expressions. Variables can be declared at the beginning of the configuration file (before rules are listed), and named using the notation:

```
{varname}=/regex/
```

For example, suppose you want to make a rule depend on the animacy of a head noun or pronoun, and you have a long list of nouns known to represent humans (just a few are given in this example), which you can encode using a variable named ‘person’:

```
{person}=/I|you|s?he|people|friend|child/
```

You can then use this variable within subsequent DepEdit rules:

```
pos=/V.*/;lemma=/{person}/&func=/obj/    #1>#2  #2:misc=AnimObj
pos=/V.*/;lemma=/{person}/&func=/nsubj/  #1>#2  #2:misc=AnimSubj
```

You can use multiple variables within the same rule, and inside the same key value, combined with normal text, e.g. lemma=/{var1}abc{var2}/.

Importing DepEdit

Starting in version 1.5.0 you can import depedit as an object into other projects using the DepEdit class, which expects a configuration file handle. You can then use run_depedit() on some input file handles without loading the configuration multiple times. Starting in version 1.6.0, the module is compatible with both Python 2.X and 3.X, and is available via PyPI.

To install the module via pip:

```
> pip install depedit
```

```
from depedit import DepEdit

config_file  = open("path/to/config.ini")
deped = DepEdit(config_file)
docs = ["path/to/infile1.conll10", "path/to/infile2.conll10"]
for doc in docs:
    infile = open(doc)
    result = deped.run_depedit(doc)
```

Alternatively, you can also create a configuration inside your module, without reading it from a text file. There are several ways of doing this, which all achieve the same result:

```
from depedit import DepEdit
d = DepEdit()

##############################
# Ways to add transformations:
##############################
# From a single string per instruction
d.add_transformation("pos=/V/\tnone\t#1:func=x")
# From args
d.add_transformation("pos=/V/\tnone\t#1:func=z","pos=/V/\tnone\t#1:func=y")
# From a list
d.add_transformation(["pos=/V/\tnone\t#1:func=a","pos=/V/\tnone\t#1:func=b"])
# From a dictionary
d.add_transformation({"nodes":"pos=/V/","rels":"none","actions":"#1:pos=a"})


##############################
# Adding variables:
##############################
d.add_variable("person", " I|you|s?he|people|friend|child")
```