One format to rule them all – The emtsv pipeline for Hungarian

Balázs Indig[1,2], Bálint Sass[1], Eszter Simon[1], Iván Mittelholcz[1], Noémi Vadász[1], Márton Makrai[1]

[1]Research Institute for Linguistics, Hungarian Academy of Sciences
[2]Centre for Digital Humanities, Eötvös Loránd University
{lastname.firstname}@nytud.mta.hu

1. Motivation

- speed vs. correctness: interaction with the pipeline
- GUI vs. toolbox philosophy
- integratedness vs. locality: easily modifiable data format
- simplicity vs. freedom: think with the head of the users
- sacrifice future extendibility for speed?
- does one size fit all?

2. Related work: other pipelines

In our view, **StanfordNLP**, **UDPipe** and **SpaCy** have many common features:

- fast, multilingual, CoNLL-U pipeline focusing on dependency parsing
- tightly integrated, easy to install, modules cannot be added or substituted
- limited API available for Python with little documentation
- many important features postponed because of the monolithic architecture
- not suitable for manual correction of annotation and experimenting by design

Magyarlanc 3.0, a popular full machine-learning based pipeline for Hungarian

- tightly integrated, easy-to-install pipeline, "batteries included" (one JAR file)
- built-in modules, which cannot be substituted or modified, new modules cannot be added
- UD compatible, CoNLL-U output, suits average need
- no Python interface, little documentation available

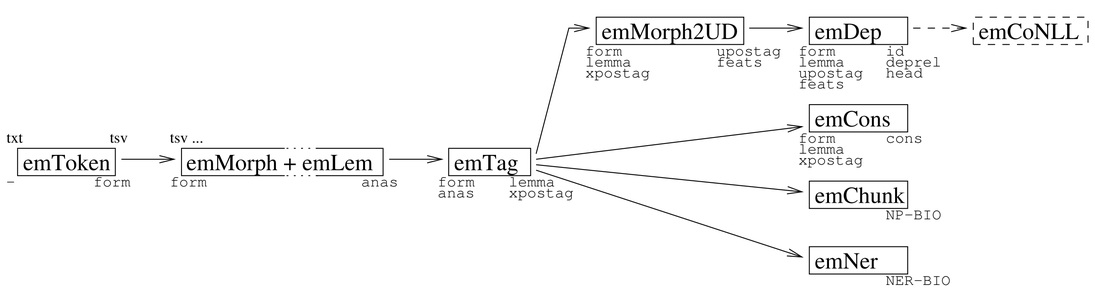
WebSty and **Weblicht**: asynchronous web services of collected tools for UD parsing

- no installation required, source for the pipeline not available
- combinable built-in modules, but new modules cannot be added
- UD compatible, CoNLL-U output, suits average need
- checks if module combination is satisfiable

4. Modules

- emToken (C++, Quex Lexical analyzer): an extendable rule-based tokeniser
- emMorph and emLem (HFST, Python):
  - the HFST-based analyzer yields all possible morpheme-based analyses of each input word (if it is present in the lexicon) in *triplets* containing the deep form, the surface form and the tag
  - the triplets are turned into lemma–simpler tag pairs by emLem lemmatiser (according to the supplied configuration)
  - custom tagset for Hungarian
- emTag (JAVA)
  - an HMM-based POS tagger trained on the Szeged Treebank
  - it can select the output from emMorph+emLem to narrow the search space
  - this allows analyses to be curated manually prior to the tagging
  - the JAVA–Python interfacing is handled by PyJNIus
- emChunk and emNer (Python): a MEMM-based sequential tagger trained on a sub-corpus of the Szeged Treebank supplied with NP-chunk and NE annotation
- emMorph2UD (Python): conversion rules from the emMorph tagset to UDv1
- emDep and emCons (JAVA): the Bohnet parser and the Berkeley parser trained on the Szeged Treebank
- emCoNLL (Python): conversion to CoNLL-U

The architecture of the modules



3. emtsv format and the pipeline

xtsv framework

- tsv-based interchange format with a general-purpose tsv-handling framework
- easy to add new modules to the pipeline, implementing the xtsv API or the format
- checks if a module combination is satisfiable through format (TSV header)
- multiple API usable from beginners to experts: CLI, Python Library, REST API

emtsv = xtsv + modules (see section 4.) + configruation (parameters, models)

- loosely-coupled, modular, open source system
- easy-to-install, compact, for **average need** (Docker image) – even with CoNLL-U output
- built-in REST API (through xtsv), can be run as a **service**
- easy to add/remove modules (even with the same function) anywhere in the pipeline
- modules can be trained, modified, run, and measured separately or in the pipeline
- the open architecture is a great test bed for new experiments and module comparison
- it is possible to correct data between modules manually or automatically

Why not CoNLL-U?

- the CoNLL-U format is not extendable in the middle of the pipeline
- we need intermediate states to enhance (manual) annotation/correction of data
- we prefer formats with standard implementation on many languages (TSV, JSON)
- the format can be easily converted into a standard-compliant form (has built-in converter)

5. Output

Uniform data format: tsv with header and empty lines as separators

```
form          lemma        xpostag                    NP-BIO id deprel head cons
#This is a comment.
A             a            [/Det|Art.Def]             B-NP   1  DET    2    (ROOT(CP(NP(NP*
párzás        párzás       [/N][Nom]                  I-NP   2  SUBJ   7    *)
és            és           [/Cnj]                     I-NP   3  CONJ   2    (CO*)
az            az           [/Det|Art.Def]             I-NP   4  DET    5    (NP*
ellés         ellés        [/N][Nom]                  E-NP   5  COORD  3    *))
egyaránt      egyaránt     [/Adv]                     O      6  MODE   7    (ADVP*)
megtörténhet  megtörténik  [/V][_Mod/V][Prs.NDef.3Sg] O      7  ROOT   0    (V_(VO*))
a             a            [/Det|Art.Def]             B-NP   8  DET    9    (NP(NP*
vízben        víz          [/N][Ine]                  E-NP   9  OBL    7    *)
vagy          vagy         [/Cnj]                     O      10 CONJ   9    (CO*)
a             a            [/Det|Art.Def]             O      11 DET    12   (NP*
szárazon      száraz       [/Adj][_Manner/Adv]        O      12 COORD  10   *))
.             .            [Punct]                    O      13 ATT    7    *))
```

JSON output of the morphological analyzer

```
{
  "bokrot": [
    {
      lemma    "bokor"
      morphana "bokor[/N]=bokr+ot[Acc]=ot"
      readable "bokor[/N]=bokr + ot[Acc]"
      tag      "[/N][Acc]"
      twolevel "b:b o:o k:k :o r:r :[/N] o:o t:t :[Acc]"
    },
    ...
  ]
}
```

6. Usage

- emMorph+emLem has an easy-to-use WEB API to query analyses of words by anyone
  - demo version is running on Heroku: https://emmorph.herokuapp.com/dstem/bokor
- annotator mode: `python emtsv.py tok,morph,pos | editor | python emtsv.py conv-morph,dep,conll`
- CLI: `python emtsv.py tok,morph,pos,conv-morph,dep,conll`
- runnable Docker: `docker run -i emtsv:latest tok,morph,pos,conv-morph,dep,conll`
- REST API (with or without Docker): send a file by HTTP POST to `http://<server>/tok/morph/pos`
- Python library: iterator in, iterator out

7. Future Work

- bootstrap a human-annotated corpus: run, correct and measure distinct modules
- over-tokenizers: stripping suffixes or splitting morphemes
- universal guesser: automatic OOV handler can be inserted before the curation step
- phrases and verb constructions: measure multiple ways to extract such structures
- load-balancing: scale to multiple machines in no time with xtsv and Docker swarm
- a multilingual chain: language identification as 0th step in the pipeline