Converting Copenhagen Dependency Treebank into Treex

Zdeněk Žabokrtský
Institute of Formal and Applied Linguistics,
Charles University in Prague

Copenhagen, April 4, 2012

Outline of the talk

- **Part 1 - Introduction**
  - Why do we want to convert CDT into Treex?
- **Part 2 - Conversion procedure**
  - Four phases: (a) collecting data, (b) selecting data, (c) raw conversion of the CDT formats into treex, (d) finilizing treex files
- **Part 3 - Basic howto's**
  - Instructions for installing needed software. Examples of search in the data.
- **Part 4 - Conclusion**
  - What can be learnt from this endeavour? Possible directions of future work.

Part 1 - Introduction

Unusual beginning

- **Disclaimer:** Due to my limited knowledge about the CDT design (especially in technical aspects), **I might have been fundamentally wrong** with some of my expectations, judgements or decisions presented below.

How it started

- September 2011 - Michael Carl contacted me.
- December 2011 - Proof of concept: I implemented a very rough prototype of the converter.
- January 2012 - I spent several days in Copenhagen to gather more information.
- March 2012 - three weeks of quite intensive work.

Motivation for the migration to Treex

- From the technological viewpoint, the CDT project seems to be unmaintained and not far from its clinical death.
- CDT annotations are very hard to exploit, even if the data repository is publicly available.
- It seems to be much easier to completely migrate to the Praguian technology than to try to fix the DTAG technology.
- The cost of the conversion should not be high. We convert other treebanks into Treex routinely (we have “treexed” 30+ treebanks).

Vocabulary

- **PDT** - Prague Dependency Treebank
- **TrEd** - a tree editor, the main tool for PDT annotations, used as visualizer in Treex
- **PML** - Prague Markup Language, XML-based markup language for linguistic data resources
- **treex**
  - full tree (“the whole thing”, the framework)
  - a file format (an application of PML)
  - a command line tool for applying Treex processing blocks on Treex data
- **core** - a collection of modules Treex::Core::*. The main modules in Treex.
- **EasyTreex** - extension for TrEd for displaying Treex files
- **TectoMT** - the original name for Treex (2005-2011), now used rather for MT based on deep-syntactic transfer
- **a-trees** - surface syntactic trees: one tree per sentence, one word per node,

Part 2 - the conversion procedure

Let’s make it modular

- After implementating some prototypes, it became clear that several hundred or perhaps thousand lines of code will be needed - modular solution is obviously needed
- I decomposed the conversion into four phases:
  1. collect the CDT data
  2. select files for conversion
  3. raw conversion to treex
  4. finilizing the conversion, already within treex

Phase 1 - Collecting the CDT data

Subproblem 1:

- **Optimist's expectation:** There's an svn repository for CDT at googlecode.com which was used for the project, so that should be the ultimate source of all releted data.
- **Reality:** Not true. I received some newer updates by an email from Martin Haulrich for en, da, and en-da files. There are probably several other sources of .tag files which I was not aware of and whose status w.r.t CDT is unclear to me.
- **Conclusion:** I used .tag and .atag files from Martin, and files for other languages from the svn repository. I included no other data into the conversion.

Time for an excursion into the data

Let's browse the CDT svn repository a little bit.

First observations

What we can see easily:

- in all data file names, one can easily distinguish at least a 4-digit code, 2-letter ISO language code, extension (.tag,.atag,.conll,.sentences.txt,.info)
- file names sometimes contain also names of annotators
- Example for 0104 and es: 0104-es-auto.tag, 0104-es-henrik.conll, 0104-es-henrik.err, 0104-es-henrik.info, 0104-es-henrik.tag, 0104-es-jonas.conll, 0104-es-jonas.err, 0104-es-jonas.info, 0104-es-jonas.tag, 0104-es-lotte.conll, 0104-es-lotte.err, 0104-es-lotte.info, 0104-es-lotte.tag, 0104-es-sentences.txt, 0104-es-soren.err, 0104-es-soren.info, 0104-es-soren.tag, 0104-es-tagged.tag

Decoding file names

Subproblem 2:

- **Optimist's expectation:** The four-digit number specifies uniquely document alignment.
- **Reality:** True.
- **Conclusion:** I rely on it.

Subproblem 3:

- **Optimist's expectation:** The extension specifies file format and content.
- **Reality:** Almost true. Actually *.txt and *.sentences.txt files play different role.
- **Conclusion:** I rely on the following: .tag files contain syntactic trees, .atag files contain alignment, .sentences.txt files contain line boundaries indicated by inserted line breaks. I use no other data files.

Decoding file names, cont.

Subproblem 4:

- **Optimist's expectation:** *auto* and *tagged* files contain no manual annotation.
- **Reality:** True.
- **Conclusion:** I use only *tagged* files, only if no manually annotated files are available.

Subproblem 5:

- **Optimist's expectation:** A file named after an annotator always contains some manual annotation.
- **Reality:** Not true.
- **Conclusion:** Presence of manual annotation in a file must be checked independently (to prefer files that really contain some annotation).

Decoding file names, cont.

Subproblem 6:

- **Optimist's expectation:** Once a file named after an annotator contains some annotation, the annotation is finished.
- **Reality:** Not true. Files exist with only a partial annotation (e.g. just the first sentence).
- **Conclusion:** Extent of manual annotation in a file must be checked (to prefer files with more annotated units).

Subproblem 7:

- **Optimist's expectation:** There is at most one file per file type, document number, language and annotator.
- **Reality:** Counterexamples such as 1014-es-lotte.tag and 1014-es-disc-lotte.tag exist.
- **Conclusion:** According to Lotte's recomendation, *disc* files are disregarded.

Selecting files

Important design decision: I would like to include **always only one version of annotation** per document number, language and annotation type, into the conversion. I suppose parallel annotations were performed more or less only for evaluation of the annotation scheme.
Subproblem 8:

- **Optimist's expectation:** Once a given language is present for a given document number, then the annotation reaches certain guaranteed level.
- **Reality:** Not true. At least three levels exist: (0) only translated text is available, no annotations, (1) syntactic annotation is available, (2) syntactic and alignment annotation is available
- **Conclusion:** Pity.

Selecting files

Subproblem 9:

- **Optimist's expectation:** Files were annotated in a the same order for all languages, so that the multilingual dimension of CDT is exploited as much as possible.
- **Reality:** Not true. All files are annotated with Danish and English, but annotation of the remaining languages is scattered. Document numbers for which full annotation of all languages are available are extremely rare.
- **Conclusion:** Pity.

Selecting files

Subproblem 10:

- **Optimist's expectation:** If there are more variants of annotation files for the same document number and language (by more annotators), it is clear which one is to be chosen.
- **Reality:** I found no source of such information in the data.
- **Conclusion:** I use a preference rules provided by Lotte, which e.g. says that Lisa's annotations should be always preferred to Lotte's annotations.

Selecting files

Subproblem 11:

- **Optimist's expectation:** *.atag files always refer to two *.tag files. So once I select an .atag file, the selection of .tag files is already determined.
- **Reality:** No. Surprisingly, the referred file names do not refer to the actually aligned files.
- **Conclusion:** I have to choose the aligned files myself.

Selecting files

Subproblem 12:

- **Optimist's expectation:** I can optimize the selection of *.tag independently of *.atag files, just according to the preference rules.
- **Reality:** No! Different *.tag files contain different number of tokens, which makes them incompatible with some *.atag files.
- **Conclusion:** Compatibility of .atag and .tag files is a hard constraint and must have the highest priority. Only then I can optimize the selection w.r.t. the preference rules.

Time for an excursion into the data

Let's look at the list of selected files.

Phase 3 - raw conversion to treex

- After this phase, for each document number (i.e., the 4-digit code), exactly one treex file is created, in which all annotations are merged.
- For each token, a node is created in the treex file.
- Tokens attributes (as well as information on dependency, alignment and coreference links) are stored in the temporary wild attribute (i.e., not properly "treexified").
- The treex file contains all languages.
- For each language, there is one flat wide tree. Sentence boundaries are not represented yet.

Raw treex - reading *.tag and *.atag files

- the CDT files look like XML ...
- ... so we should use standard tools for parsing XML files
- side remark: parsing XML files by regular expression is a BAD practice (it is extremely errorprone, brittle and very hard to maintain).
- my choice: XML::Twig

Reading *.tag and *.atag files

Subproblem 13:

- **Optimist's expectation:** *.tag and *.atag files are well-formed XML files.
- **Reality:** Ups, more than 40,000 violations of XML syntactic rules are reported by a standard XML validation tool! Many different types of errors.
- **Conclusion:** I have to fix it all, otherwise I cannot load the data in a reliable way.

A possible reason (only for some types of errors): the tag file format might have been originally based on SGML, which was the antecedent of XML and which is an outdated technology for about ten years.

Examples of XML errors in *.tag and *.atag files

- incorrect XML entities (e.g. double escaping &amp;quot;, extra spaces & amp ;)
- unquoted attributes
- unescaped special characters
- missing root elements
- missing closing tags
- mismatching opening and closing tags
- locally damaged text encoding (byte sequences that are now allowed in utf-8)

Structures in the input data

Subproblem 14:

- **Optimist's expectation:** If the XML markup tags are used for representing the data structures, one would expect that text content of attributes and elements is not further structured.
- **Reality:** Not true. E.g. the 'in' and 'out' attributes contains sequences of label:offset pairs, while the label value has a further internal structure.
- **Conclusion:** No escape from using regular expression for parsing such values.

A new convention for naming output files

- the content several tag and atag files is now merged into a single file; old naming convention is not applicable
- introduced convention captures document number, cover languages, information on their monolingual annotation (0 - not available, 1 - only translated text, 1 - tree analysis) and alignment to Danish (0 - not available, 1 - available)
- example: `1250-de10-es21-it21.treex.gz`
- advantages: one can search for files with certain annotation directly on the command line (using wildcards),
- e.g. `*de21*es21*` lists all files with syntax and alignment for German and Spanish

Combinations of available annotation

In total, 536 files. Only 7 files with full annotation in all languages, 72 files with at least four languages, 74 files with at least three languages.

```
218 de10-es10-it10
123 de00-es00-it00
 34 de20-es21-it21
 33 de10-es10-it11
 28 de10-es10-it20
 14 de21-es10-it21
 13 de10-es10-it21
 10 de20-es20-it21
 10 de10-es21-it21
  7 de21-es21-it21
  7 de21-es11-it21
  5 de21-es10-it11
  5 de10-es11-it10
  4 de20-es21-it11
  4 de10-es20-it21
  3 de21-es11-it11
```

Storing metadata

- the raw conversion preserves names of the source files
- this need was not anticipated by the schema, so again, it is stored in the wild zone

Time for an excursion into the data

Let’s look at sample data after the raw-treex phase.

Phase 4 - finilizing the converted data inside treex

Once the data are accessible by the treex interface, the standard treex functionality can be employed.
Tasks to be done in this phase:

- fill node's attributes
- create dependency edges
- create non-dependency edges (e.g. for coreference)
- create alignment links
- separate and align sentences (1 treex bundle per 1 sentence tuple)

Creating dependency and non-dependency edges

Subproblem 15:

- **Optimist's expectation:** The dependency tree skeleton is something very central in any dependency treebank, so dependency edges should be easily distinguishable from other types of links.
- **Reality:** Not true. All edges coming into a node are mixed in a single list. The ordering of the list nor labelling of the edges are sufficient for distinguishing the two types.
- **Conclusion:** A heuristic procedure (based on many trial-failure attempts) was developed for distinguishing the two types. Perhaps only an approximative solution.

Creating edges

An edge in CDT is represented as follows: the first node contains an attribute composed of an edge label and file-line offset value (positive or negative integer) with respect the line containing the second node.
Remark: referring to line numbers in the original form of an XML file is a very BAD practice! It goes against the nature of XML!

Creating edges

Subproblem 16:

- **Optimist's expectation:** Every edge connects two nodes.
- **Reality:** There are a few edges that do not point to any other node (contain some string instead of an integer offset).
- **Conclusion:** I don't know how to interpret it. Such edges are disregarded now.

Creating edges

Subproblem 17:

- **Optimist's expectation:** If a node points to some other node by a non-zero integer offset, then the second node should exist
- **Reality:** Some offsets cannot be dereferenced because the referred line does not contain a node's representation.
- **Conclusion:** Such edges are disregarded.

Creating edges

Subproblem 18:

- **Optimist's expectation:** One sentence is expected to be represented by one dependency tree. Dependency tree is a tree, hence all nodes should be connected.
- **Reality:** Not true. Some non-root nodes remain unattached (no edge touches them).
- **Conclusion:** These nodes are attached below the artificial root node.

Creating edges

Subproblem 19:

- **Optimist's expectation:** Dependency tree is a tree, hence it should not contain a cycle.
- **Reality:** Not true. Cycles exist in the data.
- **Conclusion:** Each cycle is interrupted and its root node is attached below the nearest left node outside the cycle group.

Creating edges

Subproblem 20:

- **Optimist's expectation:** The repertory of edge labels should be reasonably small.
- **Reality:** Not true. Some values occur only once or twice and look rather like typos.
- **Conclusion:** Left unchanged.

Sentence segmentation

This is a huge topic. In Treex, sentence segmentation is crucial part of text representation.
Treex document consists of a sequence of bundles, while each bundle corresponds to one sentence and all its representations (in monolingual data), or to a tuple of parallel sentences and all their representations. Each bundle is divided into language zones, which are typically crossed by alignment links.

Sentence segmentation in XML markup

Subproblem 21:

- **Optimist's expectation:** One can see $< s >$ tags in the input data. They perhaps correspond to sentences.
- **Reality:** Not reliable. Sometimes they do, but sometimes they are not present and sometimes they obviously contain more sentences.
- **Conclusion:** These tags cannot be used as the only source of information.

Sentence segmentation by connected trees

Subproblem 22:

- **Optimist's expectation:** One dependency tree should correspond to one sentence.
- **Reality:** Not reliable. As mentioned above, some nodes are unattached. Moreover, syntactic annotation is not available for many files.
- **Conclusion:** This source of information cannot be used alone.

Sentence segmentation by .sentences.txt files

Subproblem 23:

- **Optimist's expectation:** There are .sentences.txt files, in which inserted line breaks seem to represent sentence segmentation.
- **Reality:** Not reliable. Sometimes the .sentences.txt files contains a sligthly different text compared to what was really annotated.
- **Conclusion:** This source of information cannot be used alone.

Sentence segmentation

All three above mentioned sources of information on sentence boundaries are combined by heuristic rules to achieve as reliable sentence segmentation as possible. However, the result is still not perfect.

Sentence alignment

Parallel sentences are supposed to be located in the same bundles in Treex. Thus sentence alignment is needed (ideally 1:1, but not necessarily).

Sentence alignment

Subproblem 24:

- **Optimist's expectation:** The CDT texts were translated exclusively for the needs of CDT. Perhaps the sentence correspondence was preserved somewhere in the data.
- **Reality:** Not true. Sentence alignment can be realiably computed only in files containing word alignment.
- **Conclusion:** I created a simple sentence aligner based on a recursive search for closest relative positions of sentence boundaries within parallel texts. It works surprisingly well for the short CDT texts, but is not perfect.

Time for excursion into the data

Let us have a look at the data after the last conversion phase.

Basic statistics

- Danish - 536 files (100,197 tokens), syntax in all files
- English - 536 files (111,814 tokens), syntax and alignment in all files
- Italian - 413 files (86,230 tokens), 134 with syntax, 106 with alignment
- Spanish - 413 files (84,531 tokens), 72 with syntax, 57 with alignment
- German - 413 files (80,703 tokens), 87 with syntax, 36 with alignment

Part 3 - very basic 'howto' instructions

Where is the new stuff stored?

The new code and data are distributed in two repositories

- the CDT repository: https://copenhagen-dependency-treebank.googlecode.com/svn
  - CDT2012/treex/conversion_from_tag/
  - CDT2012/treex/data/
- the Treex repository, see more on http://ufal.mff.cuni.cz, the repository itself is available at https://svn.ms.mff.cuni.cz/svn/tectomt_devel
  - several blocks in treex/lib/Treex/Block/Misc/CopenhagenDT/
  - two readers (CdtTag and CdtPack) in treex/lib/Treex/Block/Read/

What do you need to access the data?

Make a checkout of the CDT svn repository on your computer.

What do you need to work with the data?

There are two possible situations:

- Either you only want to browse or annotate the treex data in TrEd. Then you need 'EasyTreex'.
- Or you are a programmer and you want to do something more sophisticated, such as writing a search block, or extract some statistic from the data, or convert new .tag files into treex. Then you need full installation of Treex.

In any case, you need to install the tree editor TrEd first. See http://ufal.mff.cuni.cz/tred

How to install and use the EasyTreex Tred extension?

- After installing TrEd, choose the option option 'Manage extensions', or choose 'Setup/Manage extensions' from the menu.
- Click on 'Get new extensions'.
- Select 'EasyTreex' (close to the end of the list)
- Press 'Install selected'
- Press 'Close'.
- Go to menu 'File/Open', change File type to 'All *.*', and open the desired treex file.
- It might be useful to switch on the side panel for displaying node attributes (menu View/Side panel).

How to use EasyTreex for annotating treex files

- you can easily change node's parent by dragging the node to its new parent
- you can easily change node's attributes in the side panel
- more complex annotation actions (such as adding an alignment or coreference link) could be in theory supported too, but it requires some programming work.

How to install the full Treex?

- follow the installation guide at http://ufal.mff.cuni.cz/treex

Very simple search in treex files

- only if you search for a specific value of a specific attribute (impossible to condition e.g. parent's atributes)
- open a treex file (or a set of them) in TrEd with EasyTrees
- press F3 and fill the searched value

(you need EasyTreex installation for this)
WARNING: currently not working, hopefully will be fixed soon.

PML-TQ query engine

- extremely powerful visual query tool
- accessible via web
- needs some programming effort to load the data into the searched database
- CDT no yet available in it

Simple search directly from the bash command line

Example: print all verb forms that appear in the German sections of the data:

```
treex Util::Eval language=de  \
anode=’print \$anode->form."\n" if \$anode->tag =˜/\ˆV/’  \
 -- path\_to\_data/*de*.treex.gz
```

(You need the full treex installation for this.)

More complex search in treex files

If the searching code is not a oneliner, you can create a new block in treex/lib/Treex/Misc/CopenhagenDT/, see SearchDemo.pm. You can run the block as follows:

```
treex -q Misc::CopenhagenDT::SearchDemo -- \
fine_treex/*es21* | sort | uniq -c
   2723 Aligned edge
   8490 Unaligned
```

(You need full treex installation for this.)

XML is nice, but you want a line-oriented format?

You can use a writer block for converting treex trees into the CoNLL format (this format was used for several shared tasks on dependency parsing).

```
treex Write::CoNLLX language=it -- fine\_treex/*it2*
1       Due      due      \_      ADJ     \_      13      subj
2       famosi   famoso   \_      ADJ     \_      3       attr
3       storici  storico  \_      ADJ     \_      1       nobj
4       russi    russo    \_      NOM     \_      3       attr
5       ,        ,        \_      PON     \_      3       pnct
```

How to run the whole CDT-Treex conversion once again

Go into CDT2012/treex/conversion_from_tag and type ‘make all’.
The whole conversion takes around 10 minutes.
(You need full treex installation for this.)

How to convert some other data from tag to treex?

In bash, type

```
treex Read::CdtTag from=test-en.tag --save
```

The resulting file is named test-en.treex.gz.
Note that the language code must be present in the file name.
(You need full treex installation for this.)

Part 4 - Conclusions

What was done

- the whole conversion is fully automatic and can be modified and re-run easily (it takes about 10 minutes)
- most of the information stored in the original tag and atag files was transferred to the Treex representation
- things that did not fit to our schema (such as metadata) are stored in wild attributes
- comfortable browsing as well as basic annotation in TrEd is possible now
- Treex Perl libraries can be used for processing the data

What was not done

- No attempt was made at improving the linguistic content of the annotations.
- No annotation macros were prepared for more complex annotations.
- There is still some bug in TrEd which make it impossible to use F3 search for treex files under EasyTreex.
- Inevitably, I made produced some bugs during the conversion which I am not aware of. Further test should be written. Be critical.

Possible directions of future work

In the nearest future:

- for Prague: fix the search-related bug in TrEd
- for Copenhagen: learn to work with Treex, revise the converted data

In a long term perspective:

- try to exploit the data for research!
- if any manpower for further development of CDT is available, I would recommend to invest the energy rather to cleaning the data, not to new manual annotations

What can we learn from the CDT story?

- Eventually, the format conversion itself was only around 20 percent of my work.
- Almost all the remaining work was needed for struggling with the CDT pecularities, which had nothing to do with Treex and which would have to be solved if the resource is to be used.

Can we generalize the experience?

Undoubtely, creating CDT was a huge endeavor and required large intellectual efforts, and CDT contains interesting annotations. But ...

- Any annotation project runs into deep troubles without continuous development of the underlying technology. Its design quality is more important than it seems!
- My experience is that if you have less than one programmer per one or two annotators, you create a debt for the future. The internal debt was (and still is) quite high in CDT.
- It is a very risky situation if the main developer of the underlying technology leaves the department and his know-how is not conveyed to anyone.
- There are three simple things that could have made it all much easier long time ago: tests, tests, and tests. Without careful testing, any software or data breaks surprisingly quickly.

Last word

In spite of all the tiny little problems along the way, CDT is a unique data resource and Treex will hopefully make it easier to use it. But this is still just the beginning.