TectoMT Tutorial

Jana Kravalová

Welcome at TectoMT Tutorial. This tutorial should take about 3 hours.

What is TectoMT

TectoMT is a highly modular NLP (Natural Language Processing) software system implemented in Perl programming language under Linux. It is primarily aimed at Machine Translation, making use of the ideas and technology created during the Prague Dependency Treebank project. At the same time, it is also hoped to facilitate and significantly accelerate development of software solutions of many other NLP tasks, especially due to re-usability of the numerous integrated processing modules (called blocks), which are equipped with uniform object-oriented interfaces.

Prerequisities

In this tutorial, we assume

- Your system is Linux
- Your shell is bash
- You have basic experience with bash and can read basic Perl

Installation and setup

- Checkout SVN repository. If you are running this installation in computer lab in Prague, you have to checkout the repository into directory /BIG (because bigger disk quota applies here):

```
cd ~/BIG
svn --username mtm co https://svn.ms.mff.cuni.cz/svn/tectomt_devel/trunk tectomt
```

- In `tectomt/install/` run `./install.sh`:

```
cd tectomt/install
./install.sh
```

- In your `.bashrc` file, add line (or source the specified file every time before experimenting with TectoMT):

```
source ~/BIG/tectomt/config/init_devel_environ.sh
```

- In your `.bash_profile` file, add line

```
source .bashrc
```

TectoMT Architecture

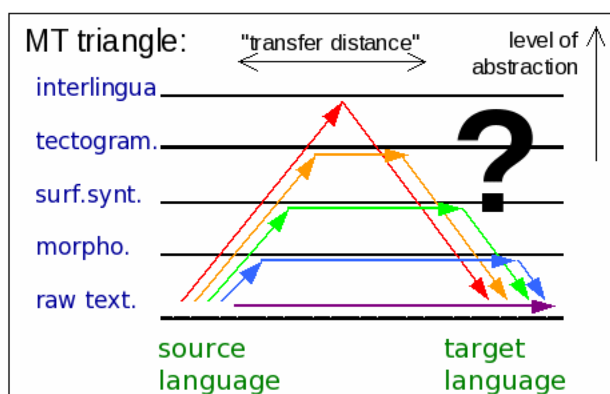
Blocks, scenarios and applications

In TectoMT, there is the following hierarchy of processing units (software components that process data):

- The basic units are blocks. They serve for some very limited, well defined, and often linguistically interpretable tasks (e.g., tokenization, tagging, parsing). Technically, blocks are Perl classes inherited from `TectoMT::Block`, each saved in a separate file. The blocks repository is in `libs/blocks/`.
- To solve a more complex task, selected blocks can be chained into a block sequence, called also a scenario. Technically, scenarios are instances of `TectoMT::Scenario` class, but in some situations (e.g. on the command line) it is sufficient to specify the scenario simply by listing block names separated by spaces.
- The highest unit is called application. Applications correspond to end-to-end tasks, be they real end-user applications (such as machine translation), or 'only' NLP-related experiments. Technically, applications are often implemented as `Makefiles`, which only glue the components existing in TectoMT. Some demo applications can be found in `applications`.

This tutorial itself has its blocks in `libs/blocks/Tutorial` and the application in `applications/tutorial`.

Layers of Linguistic Structures



The notion of 'layer' has a combinatorial nature in TectoMT. It corresponds not only to the layer of language description as used e.g. in the Prague Dependency Treebank, but it is also specific for a given language (e.g., possible values of morphological tags are typically different for different languages) and even for how the data on the given layer were created (whether by analysis from the lower layer or by synthesis/transfer).

Thus, the set of TectoMT layers is a Cartesian product {S,T} x {English,Czech,...} x {W,M,P,A,T}, in which:

- {S,T} distinguishes whether the data was created by analysis or transfer/synthesis (mnemonics: S and T correspond to (S)ource and (T)arget in MT perspective).
- {English,Czech...} represents the language in question
- {W,M,P,A,T...} represents the layer of description in terms of PDT 2.0 (W – word layer, M – morphological layer, A – analytical layer, T – tectogrammatical layer) or extensions (P – phrase-structure layer).

Blocks in block repository `libs/blocks` are located in directories indicating their purpose in machine translation.

Example: A block adding Czech morphological tags (pos, case, gender, etc.) can be found in `libs/blocks/SCzechW_to_SCzechM/Simple_tagger.pm`.

There are also other directories for other purpose blocks, for example blocks which only print out some information go to `libs/Print`. Our tutorial blocks are in `libs/blocks/Tutorial/`.

First application

Once you have TectoMT installed on your machine, you can find this tutorial in `applications/tutorial/`. After you `cd` into this directory, you can see our plain text sample data in `sample.txt`.

Most applications are defined in `Makefiles`, which describe sequence of blocks to be applied on our data. In our particular `Makefile`, four blocks are going to be applied on our sample text: sentence segmentation, tokenization, tagging and lemmatization. Since we have our input text in plain text format, the file is going to be converted into `tmt` format beforehand (the `in` target in the `Makefile`).

We can run the application:

```
make all
```

Our plain text data `sample.txt` have been transformed into `tmt`, an internal TectoMT format, and saved into `sample.tmt`. Then, all four blocks have been loaded and our data has been processed. We can now examine `sample.tmt` with a text editor (vi, emacs, etc).

- One physical `tmt` file corresponds to one document.
- A document consists of a sequence of bundles (`<bundle>`), mirroring a sequence of natural language sentences originating from the text. So, for one sentence we have one `<bundle>`.
- Each bundle contains tree shaped sentence representations on various linguistic layers. In our example `sample.tmt` we have morphological tree (`SEnglishM`) in each bundle. Later on, also an analytical layer (`SEnglishA`) will appear in each bundle as we proceed with our analysis.
- Trees are formed by nodes and edges. Attributes can be attached only to nodes. Edge's attributes must be stored as the lower node's attributes. Tree's attributes must be stored as attributes of the root node.

Changing the scenario

We'll now add a syntax analysis (dependency parsing) to our scenario by adding three more blocks. Instead of

```
analyze:
        brunblocks -S -o \
                SEnglishW_to_SEnglishM::Sentence_segmentation_simple \
                SEnglishW_to_SEnglishM::Penn_style_tokenization \
                SEnglishW_to_SEnglishM::TagMxPost \
                SEnglishW_to_SEnglishM::Lemmatize_mtree \
        -- sample.tmt
```

we'll have:

```
analyze:
        brunblocks -S -o \
                SEnglishW_to_SEnglishM::Sentence_segmentation_simple \
                SEnglishW_to_SEnglishM::Penn_style_tokenization \
                SEnglishW_to_SEnglishM::TagMxPost \
                SEnglishW_to_SEnglishM::Lemmatize_mtree \
                SEnglishM_to_SEnglishA::McD_parser_local \
                SEnglishM_to_SEnglishA::Fix_McD_Tree \
                SEnglishM_to_SEnglishA::Fill_afun_after_McD \
        -- sample.tmt
```

Note: `Makefiles` use tabulators to mark command lines. Make sure your lines start with a tabulator (or two tabulators) and not, for example, with 4 spaces.

After running

```
make all
```

we can examine our `sample.tmt` again. Really, an analytical layer `SEnglishA` describing a dependency tree with analytical functions (`<afun>`) has been added to each bundle.

Blocks can also be parametrized. For syntax parser, we might want to use a smaller but faster model. To achieve this, replace the line

```
SEnglishM_to_SEnglishA::McD_parser_local \
```

with

```
SEnglishM_to_SEnglishA::McD_parser_local TMT_PARAM_MCD_EN_MODEL=conll_mcd_order2_0.1.model \
```

You can view the trees in `sample.tmt` with TrEd by typing

```
tmttred sample.tmt
```

Try to click on some nodes to see their parameters (tag, lemma, form, analytical function etc).

Note: For more information about tree editor TrEd, see TrEd User's Manual.

If you are not familiar with `Makefile` syntax, another way of running a scenario in TectoMT is using `.scen` file (see `applications/tutorial.scen`). This file lists the blocks to be run – one block on a single line.

```
eval \${TMT_ROOT}/tools/format_convertors/plaintext_to_tmt/plaintext_to_tmt.pl English sample.txt
brunblocks -S -o --scen tutorial.scen -- sample.tmt
```

Finally, yet another way is to use a simple `bash` script (see `applications/tutorial/run_all.sh`):

```
./run_all.sh
```

Adding a new block

The linguistic structures in TectoMT are represented using the following object-oriented interface/types:

- document – `TectoMT::Document`
- bundle – `TectoMT::Bundle`
- node – `TectoMT::Node`

You can get TectoMT automatically execute your block code on each document or bundle by defining the main block entry point:

- `sub process_document` – run this procedure on each document
- `sub process_bundle` – run this procedure on each bundle (sentence)

Each block must have exactly one entry point.

We'll now examine an example of a new block in file `libs/blocks/Tutorial/Print_node_info.pm`.

This block illustrates some of the most common methods for accessing objects:

- `my @bundles = $document->get_bundles()` – an array of bundles contained in the document
- `my $root_node = $bundle->get_tree($layer_name)` – the root node of the tree of the given type in the given bundle
- `my @children = $node->get_children()` – array of the node's children
- `my @descendants = $node->get_descendants()` – array of the node's children and their children and children of their children ...
- `my $parent = $node->get_parent()` – parent node of the given node, or undef for root
- `my $root_node = $node->get_root()` – the root node of the tree into which the node belongs

Attributes of documents, bundles or nodes can be accessed by attribute getters and setters, for example:

- `$node->get_attr($attr_name)`
- `$node->set_attr($attr_name, $attr_value)`

Some interesting attributes on morphologic layer are `form`, `lemma` and `tag`. Some interesting attributes on analytical layer are `afun` (analytical function) and `ord` (surface word order). To reach `form`, `lemma` or `tag` from analytical layer, that is, when calling this attribute on an `a-node`, you use `$a_node->get_attr('m/form')` and the same way for `lemma` and `tag`. The easiest way to see the node attributes is to click on the node in TrEd:

```
tmttred sample.tmt
```

Our tutorial block `Print_node_info.pm` is ready to use. You only need to add this block to our scenario, e.g. as a new `Makefile` target:

```
print_info:
        brunblocks -S -o Tutorial::Print_node_info -- sample.tmt
```

We can observe our new block behaviour:

```
make print_info
```

Try to change the block so that it prints out the information only for verbs. (You need to look at an attribute `tag` at the `m` level). The tagset used is Penn Treebank Tagset.

Advanced block: finite clauses

Motivation

It is assumed that finite clauses can be translated independently, which would reduce combinatorial complexity or make parallel translation possible. We could even use hybrid translation – each finite clause could be translated by the most self-confident translation system. In this task, we are going to split the sentence into finite clauses.

Task

A block which, given an analytical tree (`SEnglishA`), fills each `a-node` with boolean attribute `is_clause_head` which is set to `1` if the `a-node` corresponds to a finite verb, and to `0` otherwise.

Instructions

There is a block template with hints in `libs/blocks/Tutorial/Mark_heads.pm`. You should edit the block so that the output of this block is the same a-tree, in addition with attribute `is_clause_head` attached to each `a-node`. There is also a printing block `libs/blocks/Print_finite_clauses.pm` which will print out the `a-nodes` grouped by clauses:

```
finite_clauses:
        brunblocks -S -o \
                Tutorial::Mark_heads \
                Tutorial::Print_finite_clauses \
        -- sample.tmt
```

You are going to need these methods:

- `my $root = $bundle->get_tree('tree_name')`

- `my $attr = $node->get_attr('attr_name')`
- `$node->set_attr('attr_name',$attr_value)`
- `my @eff_children = $node->get_eff_children()`

Note: `get_children()` returns topological node children in a tree, while `get_eff_children()` returns node children in a linguistic sense. Mostly, these do not differ. If interested, see Figure 1 in btred tutorial.

Hint: Finite clauses in English usually require grammatical subject to be present.

Advanced version

The output of our block might still be incorrect in special cases – we don't solve coordination (see the second sentence in sample.txt) and subordinate conjunctions.

Your turn: more tasks

SVO to SOV

Motivation: During translation from an SVO based language (e.g. English) to an SOV based language (e.g. Korean), we might need to change the word order from SVO to SOV.

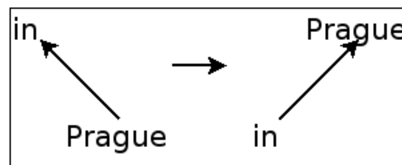
Task: Change the word order from SVO to SOV.

Instructions:

- You can use block template in `libs/blocks/BlockTemplate.pm`.
- To find an object of a verb, look for objects among effective children of a verb (`$child->get_attr('afun') eq 'Obj'`). That implies working on analytical layer.
- For debugging, a method returning surface word order of a node is useful: `$node->get_attr('ord')`. It can be used to print out nodes sorted by attribute `ord`.
- Once you have the node `$object` and the node `$verb`, use the method `$object->shift_before_node($verb)`. This method takes the whole subtree under the node `$object` and recalculates the attributes `ord` (surface word order) so that all the nodes in the subtree under `$object` have a smaller `ord` than `$verb`. That is, the method rearranges the surface word order from VO to OV.

Advanced version: This solution shifts object (or more objects) of a verb just in front of that verb node. So f.e.: *Mr. Brown has urged MPs.* changes to: *Mr. Brown has MPs urged.* You can try to change this solution, so the final sentence would be: *Mr. Brown MPs has urged.* You may need a method `$node->shift_after_subtree($root_of_that_subtree)` Subjects should have attribute `'afun' eq 'Sb'`.

Prepositions



Motivation: In dependency approach the question "where to hang prepositions" arises. In the praguian style (PDT), prepositions are heads of the subtree and the noun/pronoun is dependent on the preposition. However, another ordering might be preferable: The noun/pronoun might be the head of subtree, while the preposition would take the role of a modifier.

Task: The task is to rehang all prepositions as indicated at the picture. You may assume that prepositions have at most 1 child.

Instructions:

You are going to need these new methods:

- `my @children = $node->get_children()`
- `my $parent = $node->get_parent()`
- `$node->set_parent($parent)`

Hint:

- On analytical layer, you can use this test to recognize prepositions: `$node->get_attr('afun') eq 'AuxP'`
- To see the results, you can again use TrEd (`tmttred sample.tmt`)

Advanced version: What happens in case of multiword prepositions? For example, `because of`, `instead of`. Can you handle it?