Structural transfer

From LING073

Contents

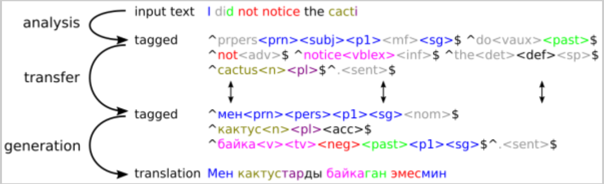
Background

The basic idea of structural transfer in RBMT

The idea of structural transfer in RBMT is to deal with the order and tag differences encountered in translation between two languages.

How structural transfer works in Apertium

Transfer takes the output of the `lex` mode (lexical selection), matches series of words based on patterns you define, and performs operations on and outputs those things. It allows you to change the order of words, change tags, add and remove words, etc.


analysis
transfer
generation
input text I did not notice the cacti
tagged ^prpers<prn><subj><p1><mf><sg>$ ^do<vaux><past>$ ^not<adv>$ ^notice<vblex><inf>$ ^the<det><def><sp>$ ^cactus<n><pl>$^.<sent>$
tagged ^мен<prn><pers><p1><sg><nom>$ ^кактус<n><pl><acc>$ ^байка<v><tv><neg><past><p1><sg>$^.<sent>$
translation Мен кактустарды байкаган эмесмин

The vertical arrows between the two tagged levels represent where structural transfer is needed. Colour coding shows [rough] correspondences.

Syntactic Structures and Parsing

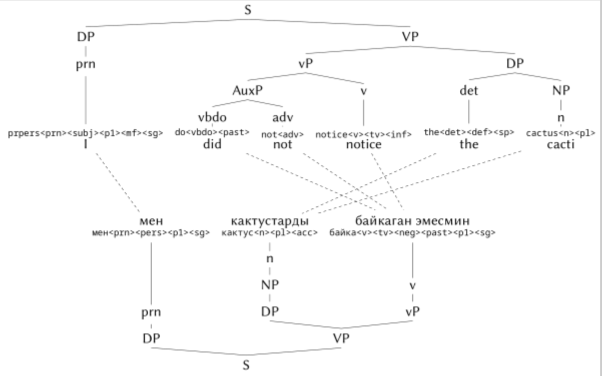
The way Apertium's recursive structural transfer system works is to parse and combine phrases, and then to output each parsed phrase. Rules specify what words or phrases are parsed together into phrases and how they're output.

In other words, at a basic level the recursive structural transfer system first turns analyses from the source languages into a tree according to parse rules, and then outputs that according to both output rules and patterns for each node. You can think of it as building a tree, going up from from the words, and then working its way down from the top to output forms again.

Example: English-to-Kyrgyz

In the English-to-Kyrgyz example for "I did not notice the cacti", in both languages a noun (`<n>`) is parsed into an `NP` (noun phrase), and an `NP` can combine with a determiner (`<det>`) to form a `DP` (determiner phrase), although Kyrgyz doesn't have a definite article. They can also both form a `DP` from just a pronoun. These rules can be written as follows:

```
NP -> %n { %1 } ;
DP -> det %NP { %2 } |
      det %NP { 1 _ %2 } |
      %prn { %1 } ;
```



The mapping between phrase-structure trees of the Kyrgyz and English sentence above ("I did not notice the cacti")

The beginning of each rule describes what kind of phrase to build (`NP` and `DP`, respectively). The part after the arrow (`->`) and the `{}`s is the material to parse, in this case specified as POS tags (`<n>`, `<det>`) or other phrases (`NP`). The part inside the `{}`s shows the order to output the elements in; in the last rule, the first element is not output. The `%`s on the input elements shows what element to copy features from to the phrase level, and the `%`s on the output elements shows which element to copy missing features to from the phrase node on output. Features are specified on phrases as tags (see information on attribute lists and output patterns below), so you can think of it as tag copying; also, different features can come from different elements (see `VP` example below). The _ is simply a blank (roughly a space) in the output.

To decide between the first two DP rules, either specifying lemmas, weighting, or using conditionals would need to be used (specifying lemmas is discussed below, the rest are discussed later). By default, the first one will be applied, which in this case is the one we want.

Similarly to how nominals are dealt with above, verbs (<**v**>, <**vblex**>, etc.) and auxiliaries (<**vaux**>) can combine in various ways to combine into a `vP` (first-level verb phrase) in both languages, although when English uses "do" auxiliaries, Kyrgyz does not use an auxiliary, and Kyrgyz encodes the equivalent of a "not" adverbial as <**neg**> on the main verb.

```
vP -> do@AuxP.$tense.$polarity v.*.inf.$lemh.$transitivity { %2 } ;
```

Here the `.$tense`, `.$polarity`, `.$lemh`, and `.$transitivity` specifications tell which element to get each of those named attributes (features) from, acting together as `%` (which will get anything not specified with a `$` attribute). Specifying `.*.inf` requires the <**v**> to have also an <**inf**> tag after any other tags, and specifying `do@` requires the `AuxP` to have a lemma of `do`, which would have been obtained in the parse using a rule like this:

```
AuxP -> %vbdo.$lemh/sl not@adv [$polarity=neg] { } |
        %vbdo.$lemh/sl { } ;
```

Here the `.$lemh/sl` part ensures the `AuxP` gets its lemma from the source language (SL) lemma. This `AuxP` rule also matches sequences of a do auxiliary and `not`<**adv**> and sets the polarity of the `AuxP` to `neg` upon such a match.

The first difference in ordering is that a `vP` and a `DP`, while both being parsed into a `VP` (a top-level verb phrase), occur in different orders. If translating from English to Kyrgyz, a rule would need to parse a sequence of `vP DP` into `VP`, but output the components in the reverse order, and set the case of the `DP` to <**acc**> (how direct objects are marked in Kyrgyz). This rule would look something like this:

```
VP -> %vP DP { 2[case=acc] _ %1 } ;
```

In both languages, a `DP` and a `VP` combine to form an `S` (sentence) in the same order:

```
S -> DP.$person.$number VP { 1 _ %2 } ;
```

This rule gets the `person` and `number` attributes from the `DP`, and makes sure the verb (`2`) gets those attributes on output (using `%`).

In addition to these rules, each POS and phrase type needs an **output pattern**. These patterns would look something like this for this English-to-Kyrgyz example:

```
n: _.number.possession.case ;
det: _.det_type.number;
prn: % ;
v: _.transitivity.polarity.tense.person.number ;
vaux: _.polarity.tense.person.number ;
vbdo: _.polarity.tense.person.number ;
```

```
S: _.person.number ;
DP: _.person.number.possession.case ;
NP: _.n_number.possession.case ;
n: _.n_number.possession.case ;
VP: _.transitivity.polarity.tense.person.number ;
vP: _.transitivity.polarity.tense.person.number ;
AuxP: _.polarity.tense.person.number ;
```

These output patterns define what order to arrange **attributes** (sets of tags, which also need to be defined) in a particular order. The _ represents a lemma followed by the main POS. Attributes are defined as follows:

```
person = (PD p3) p1 p2 p3 PD ;
number = (ND sg) sg pl sp ND ;
n_number = pl ;
polarity = (PolD "") neg PolD ;
```

For the most part these definitions are simple lists, but the first element in `()`s defines a filler in parsing if no other information is available (e.g., `ND`, for "Number to be Determined") and the default to replace it with in output if no value has been set (`""` for empty).

This example is essentially complete for this sentence, and can be viewed in its entirety or tested in this repository (https://github.swarthmore.edu/Ling073-sp22/ling073-kir-eng/blob/master/apertium-kir-eng.eng-kir.rtx). To deal with other types of things related to this example or otherwise needed between these languages, additional patterns, modifications to the existing patterns, or advanced features would need to be used.

To check what the transfer stage is doing, you can do the following:

```
$ echo "I did not notice the cacti." | apertium -d . eng-kir-transfer
```

The last line of the output is what's being sent to the generator; in this case the following:

```
^Мен<prn><pers><p1><sg><nom>$ ^кактус<n><pl><acc>$ ^байка<v><tv><neg><past><p1><sg>$^.<sent>$^.<sent>$
```

Which is generated correctly as

```
Мен кактустарды байкаган эмесмин.
```

To check the parse tree (before things are adjusted for output), you can get the output of lexical selection and feed it into `rtx-proc -T`:

```
$ echo "I did not notice the cacti" | apertium -d . eng-kir-lex | rtx-proc -T eng-kir.rtx.bin
```

In this case, the output looks like the following:

```
^Default<S><p1><sg>{
        ^Мен<DP><p1><sg><nom>{
                ^Prpers<prn><subj><p1><mf><sg>/Мен<prn><pers><p1><sg><nom>$
        }$
        ^байка<VP><tv><neg><past><PD><ND>{
                ^байка<vP><tv><neg><past><PD><ND>{
                        ^do<AuxP><neg><past><PD><ND>{
                                ^do<vbdo><past>/кыл<v><tv><past>$
                                ^not<adv>/$
                        }$
                        ^notice<v><tv><inf>/байка<v><tv><inf>$
                }$
                ^кактус<DP><PD><pl>{
                        ^the<det><def><sp>/$
                        ^кактус<NP><pl>{
                                ^cactus<n><pl>/кактус<n><pl>$
                        }$
                }$
        }$
}$
^.<sent>/.<sent>$
```

Example: Kyrgyz-to-English

In the reverse of this example, if we wanted to add "did not".

These words need to be added manually based on the polarity (negative) and tense (past) attributes of the verb. They need to be added at the `vP` level, since that's where they adjoin the `v` in the desired English version of the tree.

```
vP -> %v.*.neg.past.* { do@vbdo.past _ not@adv _ %1[tense=inf] } ;
```

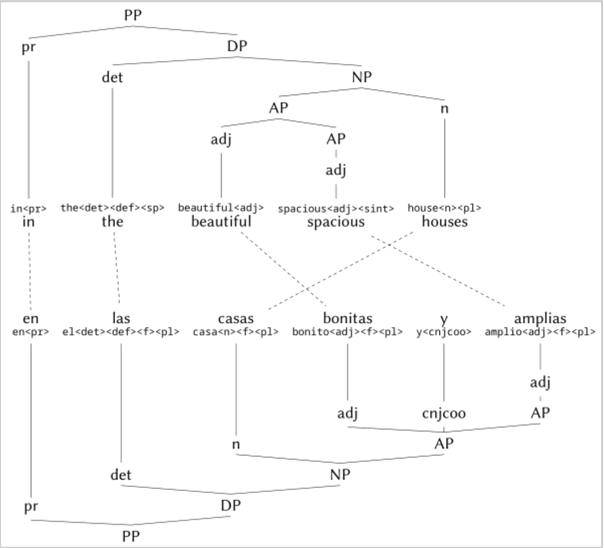
Example: English-to-Spanish

The English-to-Spanish phrase pair "in the beautiful spacious houses" = "en las casas bonitas y amplias" is shown in the image.

This phrase is a good example to walk through together in class or on your own. The following will need to be accounted for:

- The order of `AdjP` and `NP` within `DP`,
- The number and gender agreement on `det` and `adjs`,

- The addition of "y" between two adjectives in Spanish as compared to English.



The mapping between phrase-structure trees of "in the beautiful spacious houses" (English) and "en las casas bonitas y amplias" (Spanish)

Some things to note

Some advanced features include

- weighting
- conditionals
- macros

These let you do some useful things that aren't possible otherwise. See the documentation or ask your prof or TA about it :)

Examples of implemented Apertium transfer systems

Some examples are available:

- eng-spa (https://github.swarthmore.edu/Ling073-sp22/ling073-eng-spa) (in-class): a basic example from class showing how to transfer adjective+noun (etc.) from English to Spanish ("big houses → casas largas": number and gender agreement and reordering).
- eng-spa (https://github.com/apertium/apertium-recursive/blob/master/eng-spa.rtx): a more extensive English-to-Spanish example.
- eng-kir (https://github.com/apertium/apertium-eng-kir/blob/master/apertium-eng-kir.kir-eng.rtx) (apertium): English-to-Kyrgyz, lots of conditionals
- kaz-kir (https://github.com/apertium/apertium-kaz-kir/blob/master/apertium-kaz-kir.kaz-kir.rtx) (apertium): Kazakh-to-Kyrgyz, lots of macros
- uzb-kaa (https://github.com/apertium/apertium-uzb-kaa/blob/master/apertium-uzb-kaa.uzb-kaa.rtx) (apertium): Uzbek-to-Qaraqalpaq, lots of macros
- br-fr (https://github.com/apertium/apertium-br-fr/blob/rtx/apertium-br-fr.br-fr.rtx) (apertium): Breton-to-French, uses lemma lists and weighting

Writing rules

Fairly extensive documentation is available on the Apertium wiki:

- The formalism
- Something of a HOWTO

Evaluating

Scrape a mini test corpus

1. First make sure you have `scrapeTransferTests`. It should already be on the lab computers, but you can test that running `scrapeTransferTests` gives you information on using the tool.
   - If you don't have it, clone the tools repo (or `git pull` to update it, if you already have it cloned from other assignments) and run `make` or `sudo make`. Test again.
2. Scrape the transferTests from your contrastive grammar page into a small parallel corpus. E.g., `scrapeTransferTests -p abc-xyz "Language1_and_Language2/Contrastive_Grammar"` will result in an `abc.tests.txt` and `xyz.tests.txt` file that contain the respective sides of any transferTests on your contrastive grammar page specified as being for abc-to-xyz translation.
3. **Add these two files to your bilingual corpus repository** and add mention of their origin (the wiki page) to the `MANIFEST` file.

WER and PER

WER or **word error rate** is a measure of how different two texts are. You will want to know how different the translation your translation pair performs (the "test translation") is from the known good translation of phrases in your parallel corpus (the "reference translation").

PER (**position-independent error rate**) is the same measurement, just not sensitive to position in a phrase. I.e., a correct translation of every word but in an entirely wrong word order will give you high (bad) WER but low (good) PER.

To test WER and PER:

1. First make sure you have `apertium-eval-translator-line`. It should already be on the lab computers, but you can test that running `apertium-eval-translator-line` gives you information on using the tool.
   - If you don't have it, clone the tools repo (or `git pull` to update it, if you already have it cloned from other assignments) and run `make`.
2. You need two files: one **test translation**, and one **reference translation**. The reference translation is the parallel text in your corpus, e.g. `abc.tests.txt`. To get a test translation, run the source text through apertium and direct the output into a new file, e.g. `cat xyz.tests.txt | apertium -d . xyz-abc > xyz-abc.tests.txt`. You should **add the [final] test translation to your repository**.
3. The following command should then give you WER and PER measures and some other useful numbers:
   - `apertium-eval-translator -r abc.tests.txt -t xyz-abc.tests.txt`

The assignment

This assignment is due early in week 13 (this semester, **noon on Monday, April 25th, 2022**).

Getting set up

Add a page to the wiki called `Language1_and_Language2/Structural_transfer`, linking to it from the main page on the language pair.

- Put the page in the category Category:Sp22_StructuralTransfer and the categories for the two languages.

- **Before you start development** of structural transfer, perform WER, PER, and coverage tests on your short sentences corpus, and add this in to a `pre-evaluation` section.

Adding stems

Add all the words needed for the transfer tests (from the last assignment) to analyse to the bilingual dictionary.

- And make sure both analysers can analyse all sentences correctly, which includes adding the words to the relevant monolingual dictionaries as necessary.

Write structural transfer rules

Implement at least one item from your contrastive grammar.

- (If the group is working on two translation directions, then each person in each group should implement at least one item for the direction that translates into the language that they have been primarily working with. The same item does not need to be used for each direction.)

Wrapping up

Add to your structural transfer wiki page:

- Add at least one example sentence for each item you implement. Show the outputs of the following modes for your translation system: tagger, lex, transfer, and the pair itself (abc-xyz).
- Perform WER, PER, and coverage tests again, and add into a post-evaluation section on the wiki page.

Retrieved from "http://wikis.swarthmore.edu/ling073/index.php?title=Structural_transfer&oldid=17160"

Categories: Assignments | Structural transfer

- This page was last modified on 21 April 2022, at 11:47.