How to bootstrap a new pair

How to use apertium-init to bootstrap a new language pair (optionally with new monolingual data packages as well).

Contents

Prerequisites

You need to get this installed first:

- apertium/lttoolbox/hfst, see Installation, in particular the *prerequisites* parts. (You most likely don't need to go all the way, since you should get this stuff from Tino's repositories. If you're on Windows, get the Apertium VirtualBox.)
- apertium-init.py – put this script in your working directory where you will be downloading language data. You can get the script from https://apertium.org/apertium-init

Getting the monolingual packages

For each of the two languages of the pair, if it already exists, you can download and compile it by running

```
apertium-get XXX
```

Where XXX is the ISO 639-3 (http://www-01.sil.org/iso639-3/codes.asp) code of the language.

If the module doesn't exist (if it doesn't appear on this list (https://apertium.github.io/apertium-on-github/source-browser.html) or apertium-get can't find it) then you can create it like this:

```
# bootstrap the module
python3 apertium-init.py XXX
# enter the directory
cd apertium-XXX
# compile the module
make -j
```

Bootstrapping the pair

In what follows, replace XXX and YYY for the ISO 639-3 (http://www-01.sil.org/iso639-3/codes.asp) codes of your languages:

Generate the pair:

```
python3 apertium-init.py XXX-YYY
```

Then compile the pair:

```
cd apertium-XXX-YYY
./autogen.sh --with-lang1=../apertium-XXX --with-lang2=../apertium-YYY
make -j
```

And test:

```
echo house | apertium -d . XXX-YYY
echo Haus | apertium -d . YYY-XXX
```

Now you can add words to apertium-XXX-YYY.XXX-YYY.dix, then test again:

```
make -j
echo house | apertium -d . XXX-YYY
echo Haus | apertium -d . YYY-XXX
```

If you had to add words to the monolingual dictionaries, you will have to type "make" in those directories first. Alternatively, there is a shortcut from the pair directory: "make langs" should make the monolingual dictionaries even if you're in the pair directory.

Another useful piece of information about "make langs": it's also possible to speed the process by "make -j8 langs", where 8 is replaced by number of CPU cores your computer has.

Checking the tagger

Unfortunately, `apertium-tagger` will crash if given the wrong arguments. This will generally result in no output being produced when you try to translate something. Currently the best way to deal with this is to open the `modes.xml` files in the two monolingual and checking what options they give in their `XXX-tagger` and `YYY-tagger` modes.

If you find `-x`, `-u 1`, `-u 2`, or `-u 3` after `apertium-tagger`, add it to the corresponding line of the `modes.xml` file in the bilingual directory.

If you just initialized the monolingual directory, add `-u 2` in both places.

If the monolingual `modes.xml` does not mention `apertium-tagger`, bootstrap the pair again but add the argument `--no-prob1` or `--no-prob2`. If you have made changes that you don't want overwritten, add the argument `--rebuild` as well, otherwise it will be simplest to delete the entire directory.

HFST and other alternative setups

If you're making a monolingual module that should use HFST/lexc, pass the option `--analyser=hfst` to apertium-init.py.

If you're making a pair where the "left" side (XXX in the above examples) uses HFST/lexc, pass the option `--analyser1=hfst` to apertium-init.py.

If you're making a pair where the "right" side (YYY in the above examples) uses HFST/lexc, pass the option `--analyser2=hfst` to apertium-init.py.

If you're making a pair where the both sides use HFST/lexc, pass the option `--analysers=hfst` to apertium-init.py.

See https://github.com/apertium/apertium-init for more documentation, or run `./apertium-init.py --help` for all options (you can e.g. also make pairs that don't use a statistical disambiguator, or don't use a Constraint Grammar disambiguator).

Choosing an Analyser

Monolingual dictionaries can be set up to use 1 of 3 available formats: monodix, lexc, and lexd.

Monodix is best for languages that have very little conjugation (such as English) or where the conjugation is entirely with suffixes but the stem doesn't change (such as Spanish). Monodix is the default setting for apertium-init or it can be explicitly specified with `--analyser=lttoolbox`.

Lexc is best for languages where most of the conjugation is done with suffixes, potentially with some stem changes. Apertium-init will generate a Lexc dictionary with the option `--analyser=hfst`.

Lexd should be used for any language with prefixes, infixes, circumfixes, or other morphology that isn't suffixes. Apertium-init will generate a Lexd dictionary with the option `--analyser=lexd`.

If you're unsure which one to pick, Lexd will probably work well and generally involves less typing than either Monodix or Lexc.
