Apertium-recursive/Cookbook

< Apertium-recursive

This page is intended to be a collection of solutions to common transfer problems. It is sorted vaguely be complexity, so it could potentially also serve as a tutorial. Functioning rulesets for each of these examples can be found at https://github.com/apertium/apertium-recursive/tree/master/tests/cookbook.

Contents

Noun-Adjective Gender Agreement

```
NP -> adj n { 1[gender=2.gender] _ 2 } ;
```

This will copy the gender tag of the noun to the adjective.

Reordering

```
NP -> adj n { 2 _ 1 } ;
```

This will reverse the order of the noun and adjective.

This will fail to move the titlecasing at the start of a sentence. If you do

```
NP -> adj n { 2[lemcase=1.lemcase] _ 1[lemcase=2.lemcase] } ;
```

the casings will be switched around.

If instead of adj/n you have non-leaf categories like A N in NP -> A N { ... } then you also need to propagate the case manually (% will not do it for you), e.g.

```
N -> %n { 1[lemcase=$lemcase] } ;
A -> %adj { 1[lemcase=$lemcase] } ;
NP -> A %N { 2[lemcase=1.lemcase] _ 1[lemcase=2.lemcase] } ;
```

Agreement across Multiple Levels

```
NP -> n.$gender.$number { 1 } |
      adj NP.$gender.$number { 2 _ 1[gender=2.gender] } ;
DP -> det NP { 1[number=2.number] _ 2 } ;
```

These rules will match a determiner, 0 or more adjectives, and a noun and make all the adjectives agree with the noun in gender and the determiner agree with the noun in number. The adjectives will also be output after the noun in the reverse of their input order.

Selecting Output Patterns

```
v_inf: _.<inf>;
v_fin: _.tense.person.number;
v: (if (1.tense = inf) 1(v_inf)
    else 1(v_fin));
```

When outputting a verb, this will not try to add person and number tags if the tense is `<inf>`.

Case to Preposition

```
case = nom acc dat gen loc abl;
preps = NOpr to of at from;
```

```
case > preps : nom NOpr, acc NOpr, dat to, gen of, loc at, abl from;

NP -> n (if (1.case>preps = NOpr)
            { 1 }
         else
            { *(pr)[lemh=1.case>preps] _ 1 } ) ;
```

Case to Preposition on Another Level

If you want to do something like the previous example, but you want to be able to have determiners and adjectives between the preposition and the noun, you could do something like making `preps` a flag on the `NP` and passing up:

```
NP -> n [$preps=1.case>preps] { 1 } ;
DP -> det NP.$preps { 1 _1 2 } ;
PP -> DP ?(1.preps not = NOpr) { *(pr)[lemh=1.preps] _ 1 } ;
```

Now things can be added to the `NP` rule to deal with adjectives and the `DP` rule could match just an `NP` and insert determiners and so on.
