PML-TQ and Multiword Expressions

(Jiří Mírovský and Pavel Straňák, PARSEME training school lab session, January 22nd, 2015)

Introduction

Find all Predicates

```
t-node
[ functor = "PRED" ];
```

use button count (or output filter >> count() in the web client)

Predicates with an Actor

```
t-node
[ functor = "PRED",
    t-node $t := [ functor = "ACT" ] ];
```

Distribution of functors below a Predicate

```
t-node
[ functor = "PRED",
    t-node $t := [  ] ];
 >> for $t.functor give $1,count() sort by $2 desc
```

Notice that there are CONJ and DISJ in the result.

Find them:

```
t-node
[ functor = "PRED",
    t-node $t :=
   [ functor ~ "(CONJ|DISJ)" ] ];
```

We need echild – effective parentage

```
t-node
[ functor = "PRED",
   echild t-node $t := [  ] ];
 >> for $t.functor give $1,count() sort by $2 desc
```

Notice the big difference in the distributions.

Predicate without an Actor

```
t-node
[ functor = "PRED",
   0x echild t-node
   [ functor = "ACT" ] ];
```

Lists and counts of inner participants of verbs

```
t-node $p :=
[ gram/sempos = "v",
    echild t-node $c :=
    [ functor in {"ACT", "PAT", "ADDR", "EFF", "ORIG"} ] ];
  >> for $p.id,$c.functor give $1,$2
  >> give distinct $1,concat($2, ' ' over $1 sort by $2)
  >> for $2 give $1,count() sort by $2 desc
```

CPHR, DPHR, is_name_of_person

Find all CPHRs

```
t-node
  [ functor = "CPHR" ]
```

+ button count
+ >> count()

But in how many trees?

More options:

```
t-root
[ 1+x descendant t-node
    [ functor = "CPHR" ] ];
>> count()
```

or

```
t-root $r :=
[ descendant t-node
    [ functor = "CPHR" ] ];
  >> give distinct $r.id
  >> give count()
```

DPHR that is not a leaf

```
t-node
[ functor = "DPHR", sons() != 0 ];
```

DPHR not dependant on a verb

Several options, e.g.:

```
t-node
[ gram/sempos != "v",
    echild t-node
    [ functor = "DPHR" ] ];
```

or

```
t-node
[ functor = "DPHR",
    0x eparent t-node
    [ gram/sempos = "v" ] ];
```

... if you want to list the cases – possible only with the first option:

```
t-node $t :=
[ gram/sempos != "v",
    echild t-node $s :=
    [ functor = "DPHR" ] ];
  >> for $t.t_lemma,$s.t_lemma give $1,$2,count() sort by $3 desc
```

Give a list of a governing word + DPHR, and the sentences

```
t-root
[ descendant t-node $p :=
    [ echild t-node $c :=
        [ functor = "DPHR" ] ],
    atree.rf a-root $r :=
    [ +descendant a-node $a := [  ] ] ];
  >> for $r.id,$p.t_lemma,$c.t_lemma,$a.m/form,$a.ord give $1,$2,$3,$4,$5
  >> give distinct $2,$3,concat($4, ' ' over $1 sort by $5)
```

MWE

Find all t-nodes in all mwes

```
t-root
[ member mwes
    [ tnode.rfs t-node [  ] ] ];
```

+ count their types

```
t-root
[ member mwes $m :=
    [ tnode.rfs t-node [  ] ] ];
>> for $m.type give $1, count()
```

But it counts number of t-nodes in the respective types of mwes.

If we only want counts of mwes, this is enough:

```
t-root
[ member mwes $m :=
    [  ] ];
>> for $m.type give $1, count()
```

Find all t-nodes in mwes of type location

```
t-root
[ member mwes
    [ type = "location",
        tnode.rfs t-node [  ] ] ];
```

Find the first node in the depth-first-order in mwes of type location

```
t-root
[ member mwes
    [ type = "location",
        0x tnode.rfs t-node
        [ depth-first-precedes $n3 ],
        tnode.rfs t-node $n3 := [  ] ] ];
```

Counts of mwes in individual trees

```
t-root $r :=
[ member mwes [  ] ];
  >> for $r.id give count()
```

the same should work for $r in the output filter – but a different order of results.

+ >>max()

+ >>avg() **- but notice that trees without mwes are not counted**

in how many trees are given numbers of mwes:

+ >> for $1 give $1,count() sort by $2 desc

(it is the same for $r.id and $r)

if we do not want to see rare cases (with number of occurences less than 5)

+ >> filter $2 >= 5

Give a list of all mwes (as they appear in the sentence)

```
t-root
[ member mwes $m :=
    [ tnode.rfs t-node
        [ a/lex.rf|a/aux.rf a-node $a := [  ] ] ] ];
  >> give distinct concat($a.m/form, ' ' over $m sort by $a.ord)
```

Find all DPHRs that are not parts of mwe – does not work because of bug in 0x member

```
t-node $n :=
[ functor = "DPHR",
    same-tree-as t-root
    [ 0x member mwes
        [ tnode.rfs $n ] ] ];
```

But works this way: instead of saying that in the given t-root, there is no member mwes from which a link would go to the given t-node, we can say that in the tree is no t-root in which there is a mwe from which a link goes to the given t-node – and this is inrepreted correctly.

```
t-node $n3 :=
[ functor = "DPHR",
    0x same-tree-as t-root
    [ member mwes
        [ tnode.rfs $n3 ] ] ];
```

Find errors in is_name_of_person vs. mwe type person

Find nodes with is_name_of_person that are not a part of mwe of type person:

```
t-node $n3 :=
[ is_name_of_person = "1",
    0x same-tree-as t-root
    [ member mwes
        [ type = "person", tnode.rfs $n3 ] ] ];
```

Finds e.g. companies that have the owner's name in their name.

The other way (t-nodes that are part of mwe of type person but do not have is_name_of_person:

```
t-root
[ member mwes
    [ type = "person", tnode.rfs t-node
        [ !is_name_of_person = "1" ] ] ];
```

Finds e.g. Ing. Vladimír Duda

Distribution of types of mwe along with counts and percentages:

```
t-root
[ member mwes $m :=
    [ tnode.rfs t-node $t := [  ] ] ];
  >> for $m.id,$m.type give $2,count()
  >> for $1 give $1,count(),sum($2),min($2),max($2),round(avg($2),2)
  >> give $1,$2,round(ratio($2 over all) * 100,2),round(ratio($3 over all) * 100,2),$4,$5,$6
  >> give $1 & " … " & $2 & " mwe (" & $3 & "% of all mwes, " & $4 & "% of all mwe t-nodes)
… min. nodes " & $5 & ", max. nodes " & $6 & ", aver. nodes " & $7 sort by $1
```