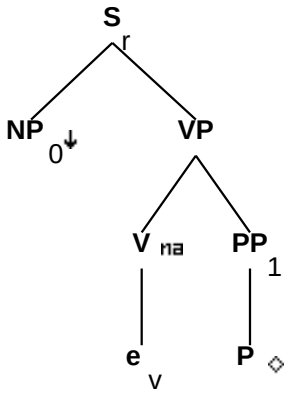
# Family "Tnx0Px1"

March 5, 2008

## 1 Tree "alphanx0Px1"

### 1.1 graphe




### 1.2 comments

Declarative tree for predicative exhaustive PPs. The exhaustive PPs are prepositions such as home, ago, abroad, above, etc.
This tree family, like other predicative tree families, is anchored by the predicted object (here, the P), with the verb, if any, adjoining in.
EX: John is home.
The road is below.

### 1.3 features

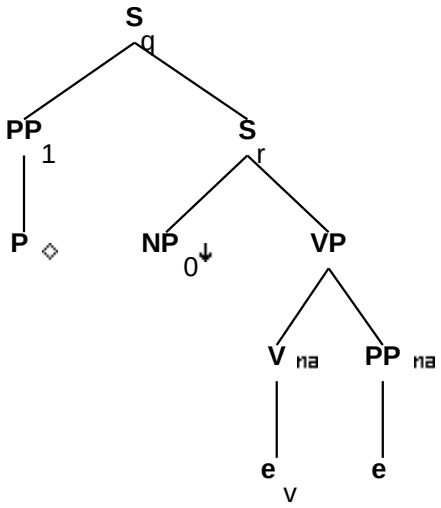
S_r.b:<extracted> = -
S_r.b:<inv> = -
S_r.b:<assign-comp> = VP.t:<assign-comp>

VP.b:<compar> = -
S_r.b:<mode> = VP.t:<mode>
S_r.b:<mainv> = VP.t:<mainv>
S_r.b:<comp> = nil

S_r.b:<tense> = VP.t:<tense>
NP_0:<agr> = S_r.b:<agr>
NP_0:<case> = S_r.b:<assign-case>
NP_0:<wh> = -
S_r.b:<agr> = VP.t:<agr>
S_r.b:<assign-case> = VP.t:<assign-case>
S_r.b:<passive> = VP.t:<passive>
VP.t:<passive> = -
VP.b:<mode> = prep
VP.b:<assign-case> = acc
S_r.b:<control> = NP_0.t:<control>

# 2 Tree "alphaPW1nx0Px1"

## 2.1 graphe




## 2.2 comments

wh object extraction tree for predicative exhaustive PPs. The exhaustive PPs are prepositions such as home, ago, abroad, above, etc. The only wh+ PP is where, so that is what goes in the PP position. This tree does *not* allow topicalization: (*home John is). This tree is *not* duplicated in the Tnx0Pnx1 family. This tree family, like other predicative tree families, is anchored by the predicted object (here, the P), with the verb, if any, adjoining in.
EX: where is John?

## 2.3 features

S_q.b:<extracted> = +

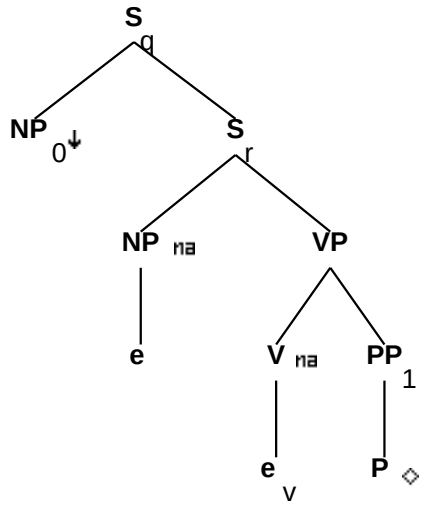
S_q.b:<inv> = S_r.t:<inv>
S_q.b:<inv> = S_q.b:<invlink>

```
PP.t:<trace> = PP_1:<trace>
S_r.t:<comp> = nil
S_r.b:<assign-comp> = VP.t:<assign-comp>
```

```
VP.b:<mode> = prep
VP.b:<compar> = -
VP.b:<assign-case> = acc
S_q.b:<mode> = S_r.t:<mode>
S_q.b:<comp> = nil
S_r.b:<mode> = VP.t:<mode>
S_r.b:<mainv> = VP.t:<mainv>
S_r.b:<comp> = nil
S_q.b:<wh> = PP_1.t:<wh>
S_r.b:<inv> = -
NP_0:<agr> = S_r.b:<agr>
NP_0:<case> = S_r.b:<assign-case>
S_r.b:<agr> = VP.t:<agr>
S_r.b:<assign-case> = VP.t:<assign-case>
S_r.b:<tense> = VP.t:<tense>
S_r.t:<conj> = nil
S_r.b:<control> = NP_0.t:<control>
S_r.b:<passive> = VP.t:<passive>
VP.t:<passive> = -
```

# 3 Tree "alphaW0nx0Px1"

## 3.1 graphe



### 3.2 comments

wh subject extraction tree for predicative exhaustive PPs. The exhaustive PPs are prepositions such as home, ago, abroad, above, etc. This tree does wh+ sentences only, no topicalization, since subject can not topicalize. This tree family, like other predicative tree families, is anchored by the predicted object (here, the P), with the verb, if any, adjoining in.
EX: who is home?

### 3.3 features

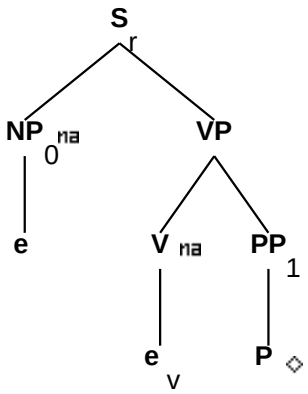
S_q.b:<extracted> = +

S_q.b:<inv> = S_r.t:<inv>
S_q.b:<wh> = NP_0.t:<wh>
S_r.t:<comp> = nil
S_r.b:<assign-comp> = VP.t:<assign-comp>

S_q.b:<comp> = nil
S_q.b:<mode> = S_r.t:<mode>
S_r.b:<mode> = VP.t:<mode>
S_r.b:<comp> = nil
S_r.b:<tense> = VP.t:<tense>
S_r.b:<inv> = -
NP:<trace> = NP_0:<trace>
NP:<agr> = NP_0:<agr>
NP:<case> = NP_0:<case>
NP:<wh> = NP_0:<wh>
NP_0:<wh> = +
S_r.b:<agr> = VP.t:<agr>
S_r.b:<assign-case> = VP.t:<assign-case>
S_r.b:<agr> = NP.t:<agr>
S_r.b:<assign-case> = NP.t:<case>
VP.b:<mode> = prep
VP.b:<assign-case> = acc
VP.b:<compar> = -
VP.t:<passive> = -
S_r.t:<conj> = nil
S_r.b:<assign-comp> = inf_nil/ind_nil/ecm

# 4 Tree "alphaInx0Px1"

## 4.1 graphe




## 4.2 comments

Imperative tree for predicative exhaustive PPs. The
exhaustive PPs are prepositions such as home, ago, abroad, above, etc.
It should be noted the the imp form of BE that adjoins on has its own tree: IVvx.
This tree family, like other predicative tree families, is anchored by the predicted object (here, the P), with the verb, if any, adjoining in.
EX: be home!

## 4.3 features

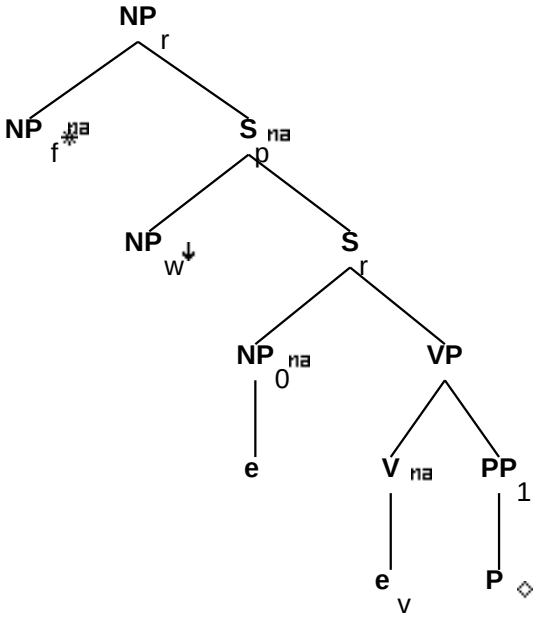
S_r.b:<extracted> = -
S_r.b:<inv> = -
S_r.b:<assign-comp> = VP.t:<assign-comp>

VP.b:<compar> = -
S_r.b:<mode> = imp
S_r.b:<mainv> = VP.t:<mainv>
S_r.b:<comp> = nil
S_r.b:<tense> = VP.t:<tense>
NP_0:<agr> = S_r.b:<agr>
NP_0:<case> = S_r.b:<assign-case>
NP_0:<wh> = -
NP_0:<agr pers> = 2
NP_0:<agr 3rdsing> = -
NP_0:<agr num> = plur/sing
NP_0:<case> = nom
S_r.b:<agr> = VP.t:<agr>
S_r.b:<assign-case> = VP.t:<assign-case>
S_r.b:<passive> = VP.t:<passive>

VP.t:<passive> = -
VP.t:<tense> = pres
VP.t:<mode> = base
VP.t:<neg> = -
VP.b:<mode> = prep
VP.b:<assign-case> = acc

# 5 Tree "betaN0nx0Px1"

## 5.1 graphe




## 5.2 comments

relative clause subject extraction tree for predicative exhaustive PPs. The exhaustive PPs are prepositions such as home, ago, abroad, above, etc.
This tree family, like other predicative tree families, is anchored by the predicted object (here, the P), with the verb, if any, adjoining in.
EX: the man who is home ...forgot to wash the dishes

## 5.3 features

S_r.b:<assign-comp> = VP.t:<assign-comp>

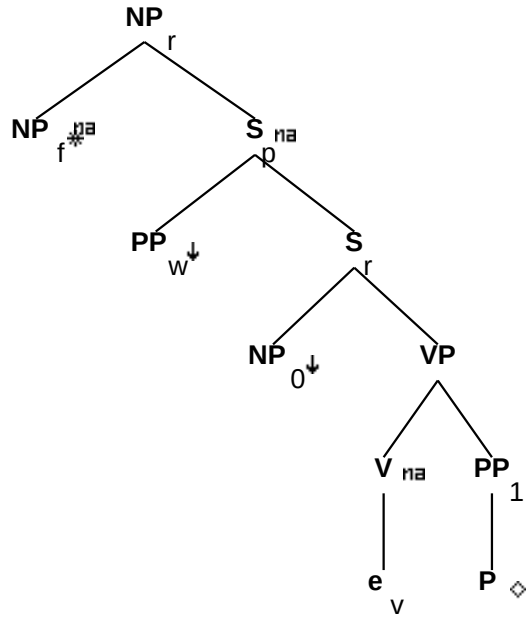
VP.b:<compar> = -
S_r.b:<mode> = VP.t:<mode>
S_r.t:<mode> = ind/inf

```
S_r.b:<comp> = nil
S_r.b:<tense> = VP.t:<tense>
S_r.t:<inv> = -
S_r.b:<agr> = VP.t:<agr>
S_r.b:<assign-case> = VP.t:<assign-case>
S_r.b:<mainv> = VP.t:<mainv>
S_r.b:<agr> = NP_0.t:<agr>
S_r.b:<assign-case> = NP_0.t:<case>
VP.b:<mode> = prep
VP.b:<assign-case> = acc
NP_r.b:<wh> = NP_f.t:<wh>
NP_r.b:<agr> = NP_f.t:<agr>
NP_r.b:<case> = NP_f.t:<case>
S_r.t:<conj> = nil

NP_w.t:<trace> = NP_0.b:<trace>
NP_w.t:<case> = NP_0.b:<case>
NP_w.t:<agr> = NP_0.b:<agr>
NP_w.t:<wh> = +
S_r.t:<comp> = nil
S_r.b:<passive> = VP.t:<passive>
VP.t:<passive> = -
NP_r.b:<rel-clause> = +
NP_f.b:<case> = nom/acc
NP_r.b:<pron> = NP_f.t:<pron>
```

# 6 Tree "betaNpxnx0Px1"

## 6.1 graphe



## 6.2 comments

Declarative tree for predicative exhaustive PPs. The exhaustive PPs are prepositions such as home, ago, abroad, above, etc.
This tree family, like other predicative tree families, is anchored by the predicted object (here, the P), with the verb, if any, adjoining in.
EX: John is home.
The road is below.

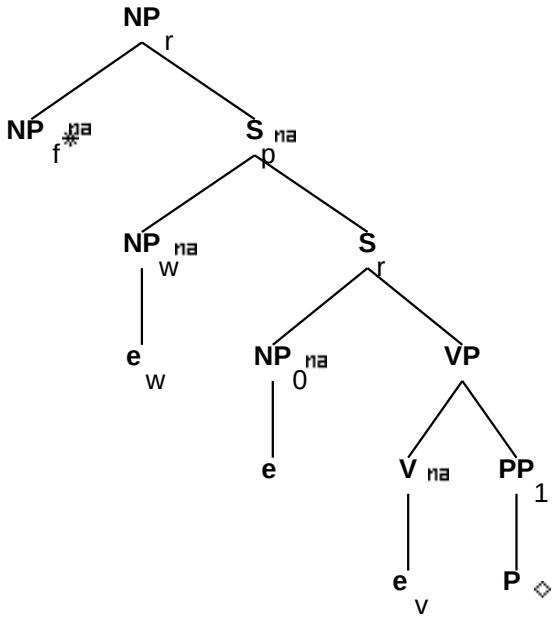
## 6.3 features

S_r.b:<extracted> = -
S_r.b:<inv> = -
S_r.b:<assign-comp> = VP.t:<assign-comp>

S_r.b:<mode> = VP.t:<mode>
S_r.b:<mainv> = VP.t:<mainv>
S_r.b:<comp> = nil
S_r.b:<tense> = VP.t:<tense>
NP_0:<agr> = S_r.b:<agr>
NP_0:<case> = S_r.b:<assign-case>
NP_0:<wh> = -
S_r.b:<agr> = VP.t:<agr>
S_r.b:<assign-case> = VP.t:<assign-case>
S_r.b:<passive> = VP.t:<passive>
VP.t:<passive> = -
VP.b:<mode> = prep
VP.b:<assign-case> = acc
VP.b:<compar> = -
S_r.b:<control> = NP_0.t:<control>
S_r.t:<inv> = -
PP_w.t:<wh> = +
NP_r.b:<wh> = NP_f.t:<wh>
NP_r.b:<agr> = NP_f.t:<agr>
NP_r.b:<case> = NP_f.t:<case>
NP_f.b:<case> = acc/nom
S_r.t:<comp> = nil
NP_r.b:<rel-clause> = +
NP_f.b:<case> = nom/acc
NP_r.b:<pron> = NP_f.t:<pron>

# 7 Tree "betaNc0nx0Px1"

## 7.1 graphe




## 7.2 comments

relative clause subject extraction tree for predicative exhaustive PPs. The exhaustive PPs are prepositions such as home, ago, abroad, above, etc. This tree family, like other predicative tree families, is anchored by the predicted object (here, the P), with the verb, if any, adjoining in. EX: the man who is home ...forgot to wash the dishes

## 7.3 features

S_r.b:<assign-comp> = VP.t:<assign-comp>

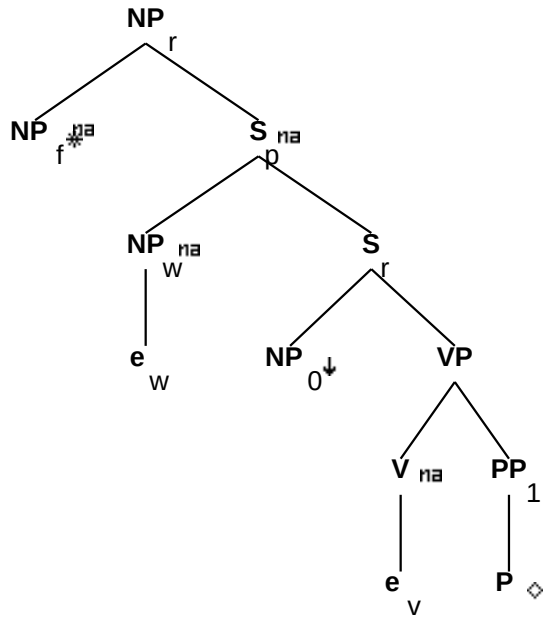
VP.b:<compar> = -
S_r.b:<mode> = VP.t:<mode>
S_r.b:<comp> = nil
S_r.b:<tense> = VP.t:<tense>
S_r.t:<inv> = -
S_r.b:<agr> = VP.t:<agr>
S_r.b:<assign-case> = VP.t:<assign-case>
S_r.b:<mainv> = VP.t:<mainv>
S_r.b:<agr> = NP_0.t:<agr>
S_r.b:<assign-case> = NP_0.t:<case>
S_r.b:<passive> = VP.t:<passive>

VP.t:<passive> = -
VP.b:<mode> = prep
VP.b:<assign-case> = acc
NP_r.b:<wh> = NP_f.t:<wh>
NP_r.b:<agr> = NP_f.t:<agr>
NP_r.b:<case> = NP_f.t:<case>
S_r.t:<conj> = nil

NP_w.t:<trace> = NP_0.b:<trace>
NP_w.t:<case> = NP_0.b:<case>
NP_w.t:<agr> = NP_0.b:<agr>
NP_r.b:<rel-clause> = +
S_r.t:<mode> = inf/ger/ind/prep
S_r.t:<nocomp-mode> = inf/ger/prep
VP.t:<assign-comp> = that/ind_nil/inf_nil/ecm
S_r.b:<nocomp-mode> = S_r.b:<mode>
NP_f.b:<case> = nom/acc
NP_r.b:<pron> = NP_f.t:<pron>

# 8 Tree "betaNcnx0Px1"

## 8.1 graphe




## 8.2 comments

Declarative tree for predicative exhaustive PPs. The exhaustive PPs are prepositions such as home, ago, abroad, above, etc.
This tree family, like other predicative tree families, is anchored by the predicted object (here, the P), with the verb, if any, adjoining in.

```
EX: John is home.
    The road is below.
```

### 8.3 features

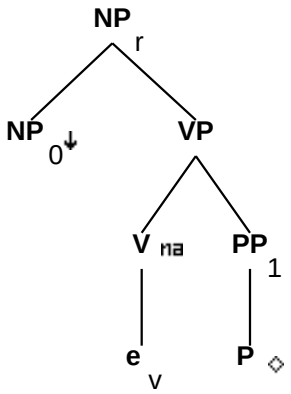
```
S_r.b:<extracted> = -
S_r.b:<inv> = -
S_r.b:<assign-comp> = VP.t:<assign-comp>



S_r.b:<mode> = VP.t:<mode>
S_r.b:<mainv> = VP.t:<mainv>
S_r.b:<comp> = nil
S_r.b:<tense> = VP.t:<tense>
NP_0:<agr> = S_r.b:<agr>
NP_0:<case> = S_r.b:<assign-case>
NP_0:<wh> = -
S_r.b:<agr> = VP.t:<agr>
S_r.b:<assign-case> = VP.t:<assign-case>
S_r.b:<passive> = VP.t:<passive>
VP.t:<passive> = -
VP.b:<mode> = prep
VP.b:<assign-case> = acc
VP.b:<compar> = -
S_r.b:<control> = NP_0.t:<control>
NP_r.b:<wh> = NP_f.t:<wh>
NP_r.b:<agr> = NP_f.t:<agr>
NP_r.b:<case> = NP_f.t:<case>
NP_f.b:<case> = acc/nom
S_r.t:<inv> = -
S_r.t:<mode> = ind/inf
S_r.t:<nocomp-mode> = ind
VP.t:<assign-comp> = that/for/ind_nil
S_r.b:<nocomp-mode> = S_r.b:<mode>
NP_r.b:<rel-clause> = +
NP_f.b:<case> = nom/acc
NP_r.b:<pron> = NP_f.t:<pron>
```

# 9 Tree "alphaGnx0Px1"

## 9.1 graphe




## 9.2 comments

Gerund NP tree for predicative exhaustive PPs. The exhaustive PPs are prepositions such as home, ago, abroad, above, etc. This tree family, like other predicative tree families, is anchored by the predicated object (here, the P), with the verb, if any, adjoining in. There is no corresponding D tree (*the being of home; *the being home).
...John('s) being home...

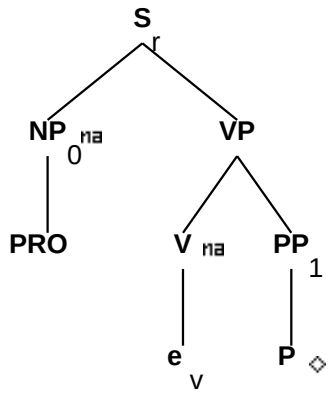
## 9.3 features

NP_0:<wh> = NP_r.b:<wh>
VP.t:<mode> = ger
NP_r.b:<case> = nom/acc
NP_r.b:<agr num> = sing
NP_r.b:<agr pers> = 3
NP_r.b:<agr 3rdsing> = +
VP.b:<mode> = prep
VP.b:<assign-case> = acc
VP.b:<compar> = -

NP_r.b:<gerund> = +
NP_0:<case> = acc/gen

# 10 Tree "alphanx0Px1-PRO"

## 10.1 graphe




## 10.2 comments

Predicative Exhaustive PPs w/ PRO subject
The exhaustive PPs are
prepositions such as home, ago, abroad, above, etc.
This tree family, like other predicative tree families, is anchored by the predicted object (here, the P), with the verb, if any, adjoining in.

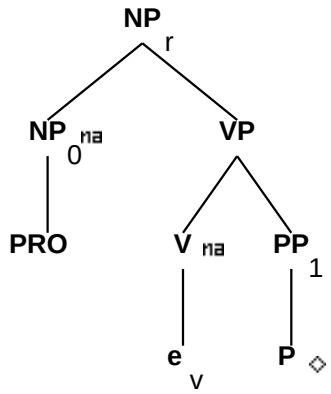
John wants [PRO to be abroad].

## 10.3 features

S_r.b:<extracted> = -
S_r.b:<inv> = -
S_r.b:<assign-comp> = VP.t:<assign-comp>
VP.b:<compar> = -
S_r.b:<mode> = VP.t:<mode>
S_r.b:<mainv> = VP.t:<mainv>
S_r.b:<comp> = nil
S_r.b:<tense> = VP.t:<tense>
S_r.b:<assign-case> = NP_0.t:<case>
NP_0:<agr> = S_r.b:<agr>
NP_0:<wh> = -
NP_0.t:<case> = none
S_r.b:<agr> = VP.t:<agr>
S_r.b:<passive> = VP.t:<passive>
VP.t:<passive> = -
VP.b:<mode> = prep
VP.b:<assign-case> = acc
S_r.b:<control> = NP_0.t:<control>
VP.t:<mode> = inf/ger

# 11 Tree "alphaGnx0Px1-PRO"

## 11.1 graphe




## 11.2 comments

Gerund NP tree w/ PRO subject for predicative exhaustive PPs. The exhaustive PPs are prepositions such as home, ago, abroad, above, etc. This tree family, like other predicative tree families, is anchored by the predicted object (here, the P), with the verb, if any, adjoining in. There is no corresponding D tree (*the being of home; *the being home).
[PRO being home] is fun for John.

## 11.3 features

NP_0:<wh> = NP_r.b:<wh>
NP_0.t:<case> = none
NP_0.t:<wh> = -
VP.t:<mode> = ger
NP_r.b:<case> = nom/acc
NP_r.b:<agr num> = sing
NP_r.b:<agr pers> = 3
NP_r.b:<agr 3rdsing> = +
VP.b:<mode> = prep
VP.b:<assign-case> = acc
VP.b:<compar> = -
NP_r.b:<gerund> = +