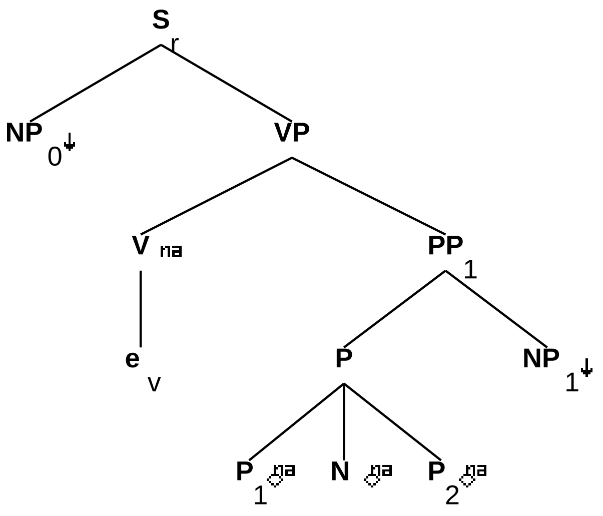
Family "Tnx0PNaPnx1"

March 5, 2008

1 Tree "alphanx0PNaPnx1"

1.1 graphe



1.2 comments

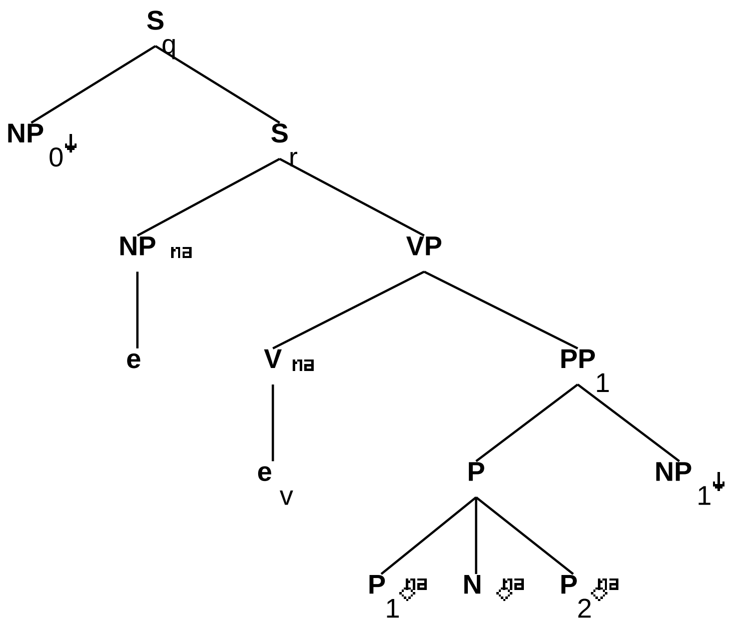
```
Declarative tree for predicative PPs.  This tree family, like other predicative
tree families, is anchored by the predicted object (here, the P), with the
verb, if any, adjoining in.
EX: John is in charge of the case.
```

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
PP_1.b:<assign-case> = P.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
PP_1.b:<wh> = NP_1.t:<wh>
S_r.b:<control> = NP_0:<control>
```

2 Tree "alphaW0nx0PNaPnx1"

2.1 graphe



2.2 comments

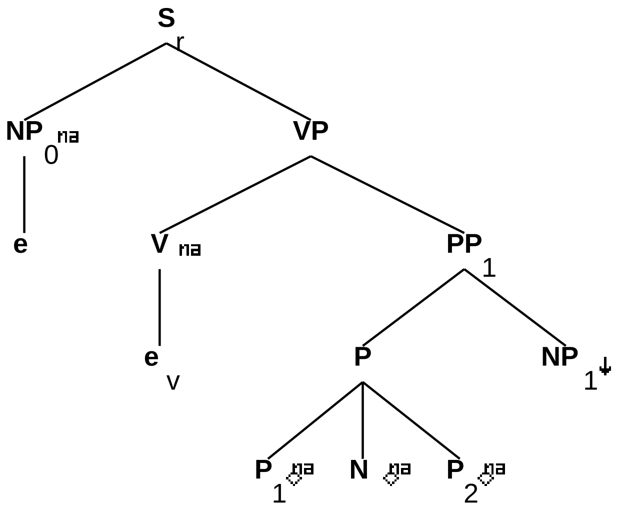
wh subject extraction tree for predicative PPs. This tree does wh+ sentences only, no topicalization, since subject can not topicalize. This tree family, like other predicative tree families, is anchored by the predicted object (here, the P), with the verb, if any, adjoining in.
EX: who is in charge of the park's maintenance?

2.3 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>



VP.b:<compar> = -
VP.t:<passive> = -
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
PP_1.b:<assign-case> = P.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
PP_1.b:<wh> = NP_1.t:<wh>
S_r.b:<assign-comp> = inf_nil/ind_nil/ecm

S_r.t:<conj> = nil
```

3 Tree "alphaInx0PNaPnx1"

3.1 graphe



3.2 comments

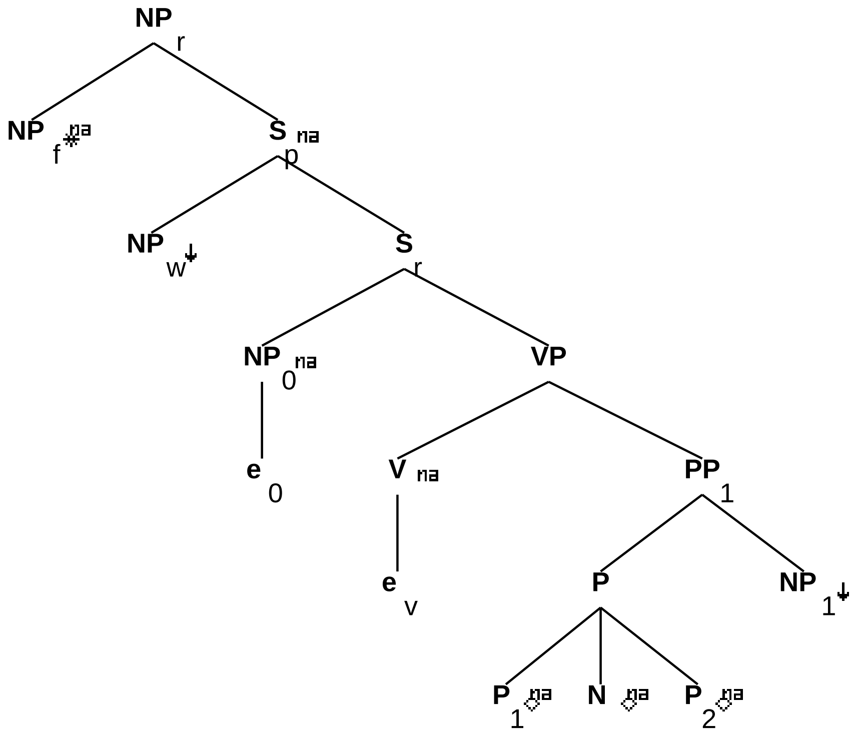
```
Imperative tree for predicative PPs.  It should be noted the the imp form of BE
that adjoins on has its own tree: IVvx.  This tree family, like other
predicative tree families, is anchored by the predicted object (here, the P),
with the verb, if any, adjoining in.
EX: be in back of the brick building by 5 o'clock!
```

3.3 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
PP_1.b:<assign-case> = P.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
PP_1.b:<wh> = NP_1.t:<wh>
```

4 Tree "betaN0nx0PNaPnx1"

4.1 graphe



4.2 comments

```
relative clause subject extraction tree for predicative PPs.
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 in back of the maple tree ...is feeding the pigeons.
```

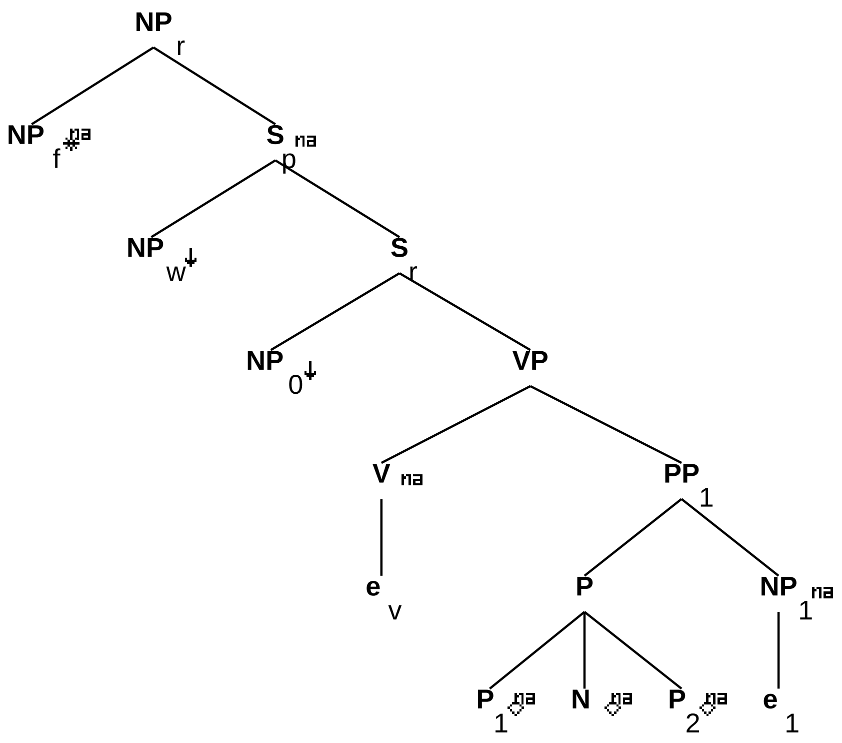
4.3 features

```
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> = -
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.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
PP_1.b:<assign-case> = P.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
P.b:<wh> = -
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
NP_r.b:<rel-clause> = +
NP_f.b:<case> = nom/acc
NP_r.b:<pron> = NP_f.t:<pron>
```

5 Tree "betaN1nx0PNaPnx1"

5.1 graphe



5.2 comments

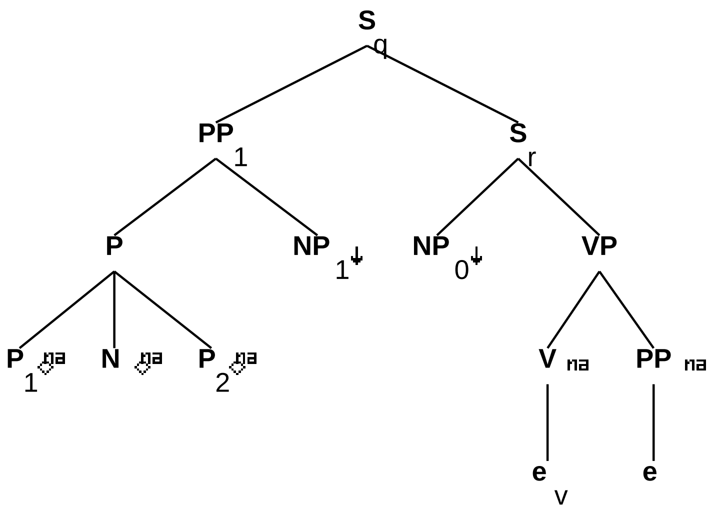
```
relative clause object extraction tree for NP embedded in the predicative PP.
This tree family (Tnx0Pnx1), like other predicative tree families, is anchored
by the predicted object (here, the P), with the verb, if any, adjoining in.
EX: the project that this man is in charge of....is being completed as we
speak.

NOTE: Currently, we are missing the tree that lets us do the following:
the park at which the man is...is being torn up for condominiums.
```

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:<tense> = VP.t:<tense>
S_r.t:<inv> = -
S_r.b:<inv> = -
NP_0:<agr> = S_r.b:<agr>
NP_0:<case> = S_r.b:<assign-case>
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.b:<agr> = VP.t:<agr>
S_r.b:<tense> = VP.t:<tense>
S_r.b:<assign-case> = VP.t:<assign-case>
S_r.b:<mainv> = VP.t:<mainv>
S_r.b:<passive> = VP.t:<passive>
VP.t:<passive> = -
VP.b:<mode> = prep
VP.b:<assign-case> = acc
PP_1.b:<assign-case> = P.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
NP_w.t:<trace> = NP_1.b:<trace>
NP_w.t:<case> = NP_1.b:<case>
NP_w.t:<agr> = NP_1.b:<agr>
NP_w.t:<wh> = +
S_r.t:<comp> = nil
NP_r.b:<rel-clause> = +
NP_f.b:<case> = nom/acc
NP_r.b:<pron> = NP_f.t:<pron>
```

6 Tree "alphapW1nx0PNaPnx1"

6.1 graphe



6.2 comments

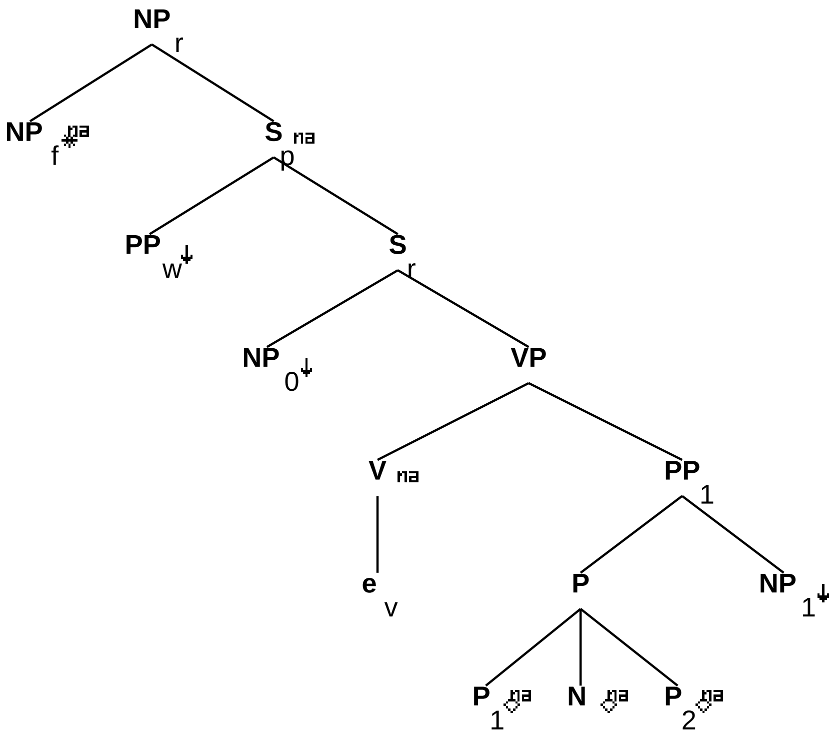
WH object extraction for predicative PPs. This brings the Prep along for the ride with a wh+ NP. The tree in which the entire PP is extracted and made wh+ (i.e. where), is covered under the W1nx0Px1 tree in the Tnx0Px1 family. Here, topicalization is *not* possible. This tree family, like other predicative tree families, is anchored by the predicted object (here, the P), with the verb, if any, adjoining in.
EX: in charge of what is Sally?

6.3 features

```
S_q.b:<extracted> = +

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

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

S_r.b:<passive> = VP.t:<passive>
VP.t:<passive> = -

VP.b:<mode> = prep
VP.b:<assign-case> = acc
VP.b:<compar> = -
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_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_q.b:<wh> = PP_1.t:<wh>
PP_1.t:<trace> = PP.t:<trace>
PP_1.b:<assign-case> = P.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
PP_1.b:<wh> = NP_1.t:<wh>
S_r.b:<control> = NP_0:<control>
S_r.t:<conj> = nil
```

7 Tree "betaNpxnx0PNaPnx1"

7.1 graphe



7.2 comments

```
Declarative tree for predicative PPs.  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 in charge of the case.
```

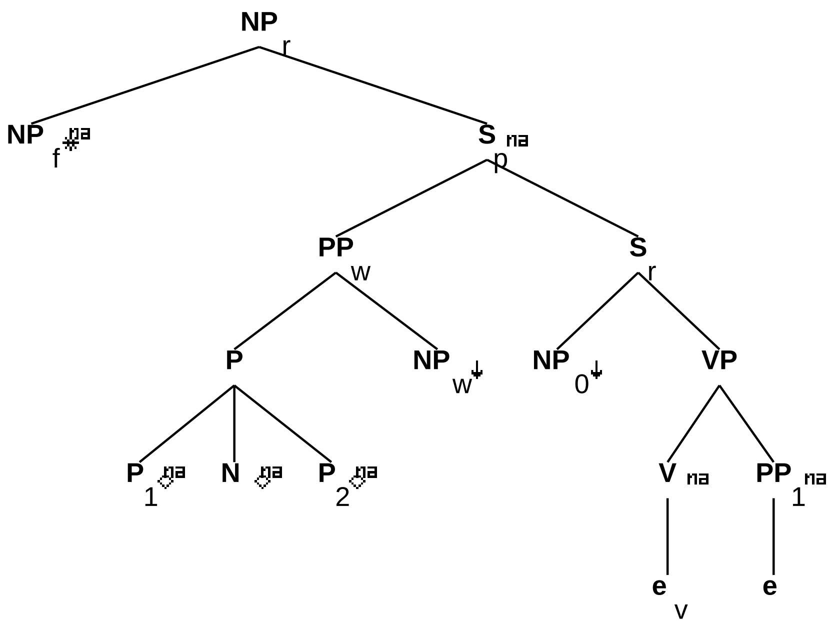
7.3 features

```
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
PP_1.b:<assign-case> = P.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
P.b:<wh> = -
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>
```

8 Tree "betaNPnx1nx0PNaPnx1"

8.1 graphe



8.2 comments

relative clause object extraction tree for NP embedded in the predicative PP. This tree family (Tnx0Pnx1), like other predicative tree families, is anchored

by the predicted object (here, the P), with the verb, if any, adjoining in.
EX: the project that this man is in charge of....is being completed as we speak.

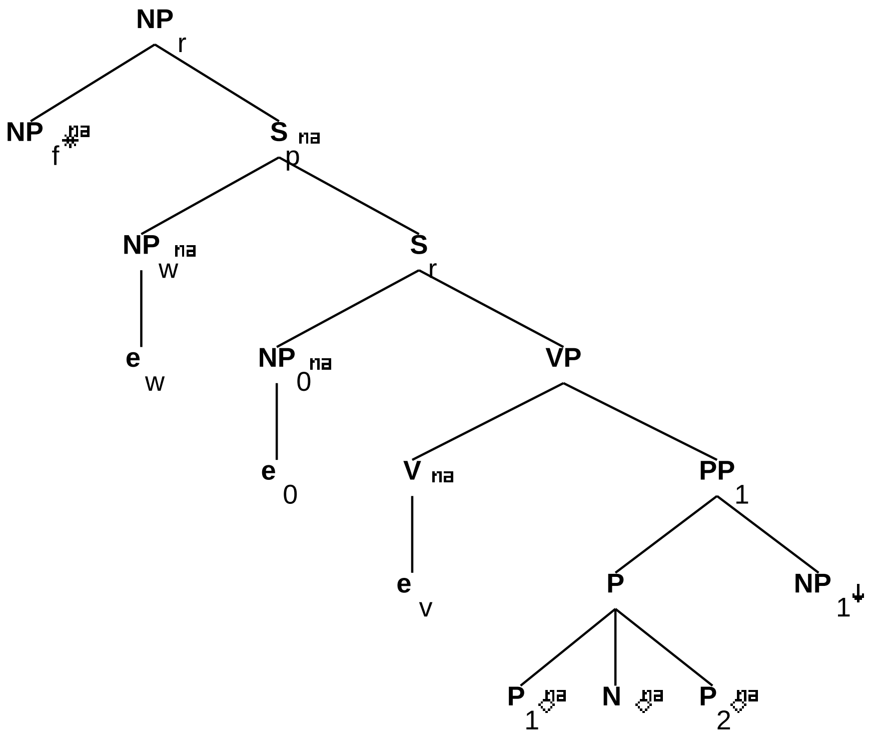
NOTE: Currently, we are missing the tree that lets us do the following:
the park at which the man is...is being torn up for condominiums.

8.3 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:<tense> = VP.t:<tense>
S_r.t:<inv> = -
S_r.b:<inv> = -
NP_0:<agr> = S_r.b:<agr>
NP_0:<case> = S_r.b:<assign-case>
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.b:<agr> = VP.t:<agr>
S_r.b:<tense> = VP.t:<tense>
S_r.b:<assign-case> = VP.t:<assign-case>
S_r.b:<mainv> = VP.t:<mainv>
S_r.b:<passive> = VP.t:<passive>
VP.t:<passive> = -
VP.b:<mode> = prep
VP.b:<assign-case> = acc
NP_w.t:<wh> = +
S_r.t:<comp> = nil
PP_w.t:<trace> = PP_1.b:<trace>
PP_w.t:<case> = PP_1.b:<case>
PP_w.t:<agr> = PP_1.b:<agr>
PP_w.b:<assign-case> = P.t:<assign-case>
PP_w.b:<assign-case> = NP_w.t:<assign-case>
PP_w.b:<wh> = NP_w.t:<wh>
NP_r.b:<rel-clause> = +
NP_f.b:<case> = nom/acc
NP_r.b:<pron> = NP_f.t:<pron>
```

9 Tree "betaNc0nx0PNaPnx1"

9.1 graphe



9.2 comments

```
relative clause subject extraction tree for predicative PPs.
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 in back of the maple tree ...is feeding the pigeons.
```

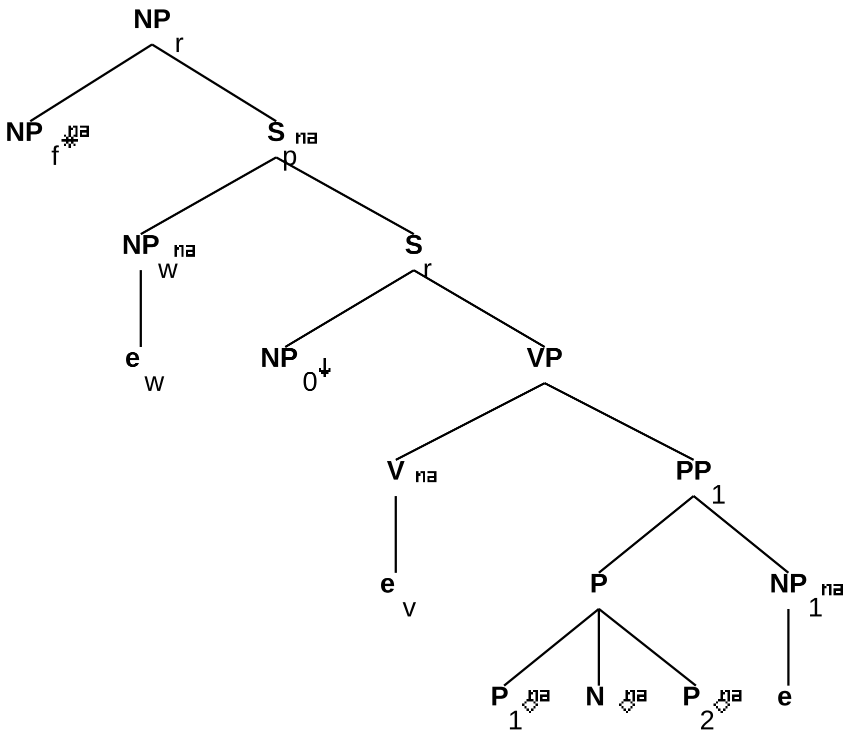
9.3 features

```
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> = -
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.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
PP_1.b:<assign-case> = P.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
P.b:<wh> = -
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>
```

10 Tree "betaNc1nx0PNaPnx1"

10.1 graphe



10.2 comments

```
relative clause object extraction tree for NP embedded in the predicative PP.
This tree family (Tnx0Pnx1), like other predicative tree families, is anchored
by the predicted object (here, the P), with the verb, if any, adjoining in.
EX: the project that this man is in charge of....is being completed as we
speak.

NOTE: Currently, we are missing the tree that lets us do the following:
the park at which the man is...is being torn up for condominiums.
```

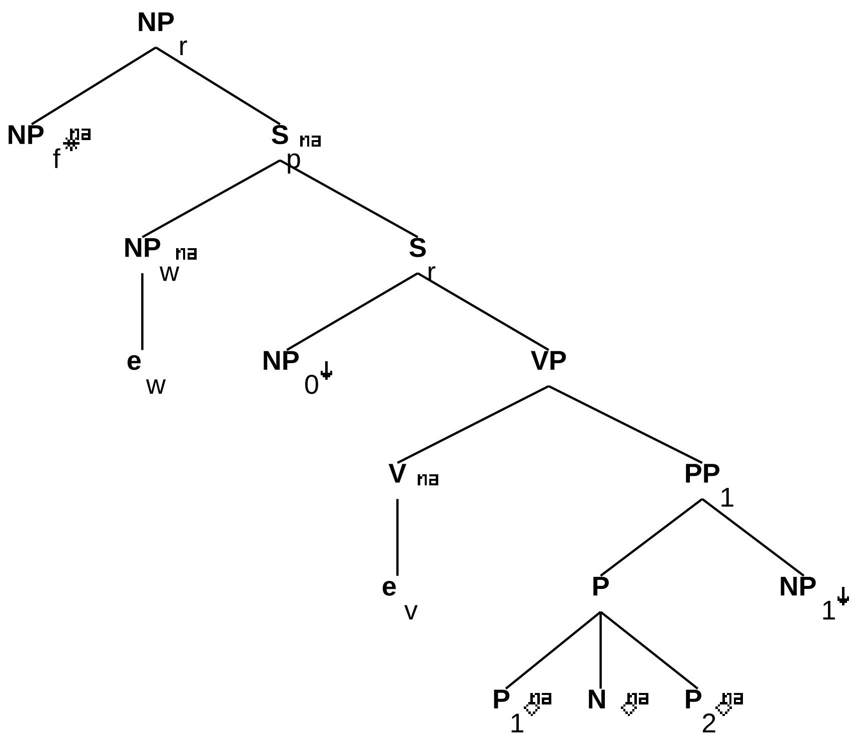
10.3 features

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




S_r.b:<mode> = VP.t:<mode>
S_r.b:<tense> = VP.t:<tense>
S_r.t:<inv> = -
S_r.b:<inv> = -
```

```
NP_0:<agr> = S_r.b:<agr>
NP_0:<case> = S_r.b:<assign-case>
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.b:<agr> = VP.t:<agr>
S_r.b:<tense> = VP.t:<tense>
S_r.b:<assign-case> = VP.t:<assign-case>
S_r.b:<mainv> = VP.t:<mainv>
S_r.b:<passive> = VP.t:<passive>
VP.t:<passive> = -
VP.b:<mode> = prep
VP.b:<assign-case> = acc
VP.b:<compar> = -
PP_1.b:<assign-case> = P.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
NP_w.t:<trace> = NP_1.b:<trace>
NP_w.t:<case> = NP_1.b:<case>
NP_w.t:<agr> = NP_1.b:<agr>
NP_r.b:<rel-clause> = +
S_r.t:<mode> = inf/ind
S_r.t:<nocomp-mode> = ind
VP.t:<assign-comp> = that/for/ind_nil
S_r.b:<nocomp-mode> = S_r.b:<mode>
NP_f.b:<case> = nom/acc
NP_r.b:<pron> = NP_f.t:<pron>
```

11 Tree "betaNcnx0PNaPnx1"

11.1 graphe



11.2 comments

```
Declarative tree for predicative PPs.  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 in charge of the case.
```

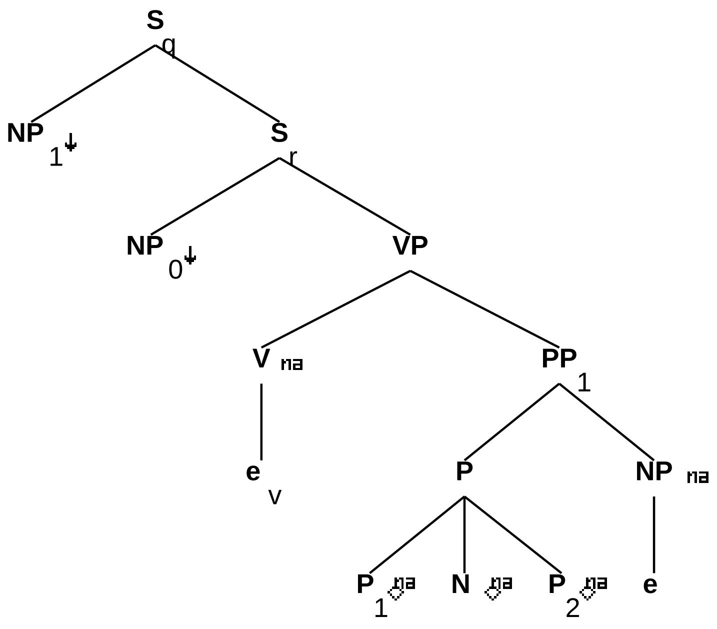
11.3 features

```
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
PP_1.b:<assign-case> = P.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
P.b:<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:<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>
```

12 Tree ”alphaW1nx0PNaPnx1”

12.1 graphe



12.2 comments

wh object extraction tree for predicative PPs. This tree family, like other predicative tree families, is anchored by the predicted object (here, a P, an N and a P), with the verb, if any, adjoining in.

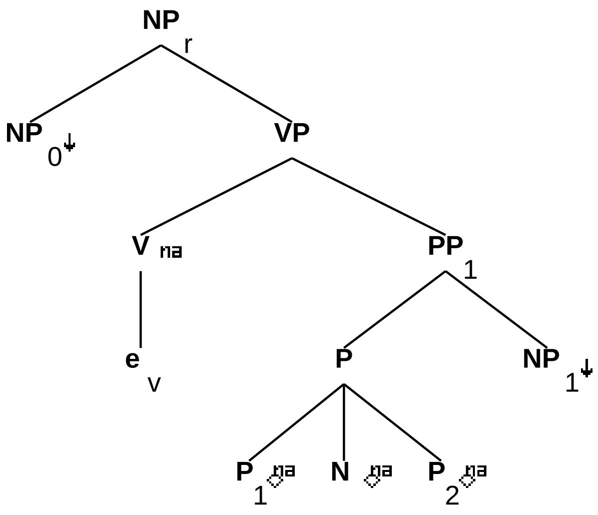
Ex: What is Sally in charge of?

12.3 features

```
S_q.b:<extracted> = +

S_q.b:<inv> = S_r.t:<inv>
S_q.b:<inv> = S_q.b:<invlink>
S_q.b:<wh> = NP_1.t:<wh>
S_r.t:<comp> = nil
S_r.b:<assign-comp> = VP.t:<assign-comp>



VP.b:<compar> = -
S_q.b:<comp> = nil
S_q.b:<mode> = S_r.t:<mode>
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:<inv> = -
NP:<trace> = NP_1:<trace>
NP:<agr> = NP_1:<agr>
NP:<case> = NP_1:<case>
NP:<wh> = NP_1:<wh>
S_r.b:<agr> = VP.t:<agr>
S_r.b:<assign-case> = VP.t:<assign-case>
S_r.b:<agr> = NP_0.t:<agr>
S_r.b:<assign-case> = NP_0.t:<case>
S_r.b:<control> = NP_0.t:<control>
S_r.b:<passive> = VP.t:<passive>
VP.t:<passive> = -
VP.b:<mode> = prep
VP.b:<assign-case> = acc
PP_1.b:<assign-case> = P.t:<assign-case>
PP_1.b:<assign-case> = NP.t:<case>
PP_1.b:<wh> = NP.t:<wh>
S_r.t:<conj> = nil
```

13 Tree "alphaGnx0PNaPnx1"

13.1 graphe



13.2 comments

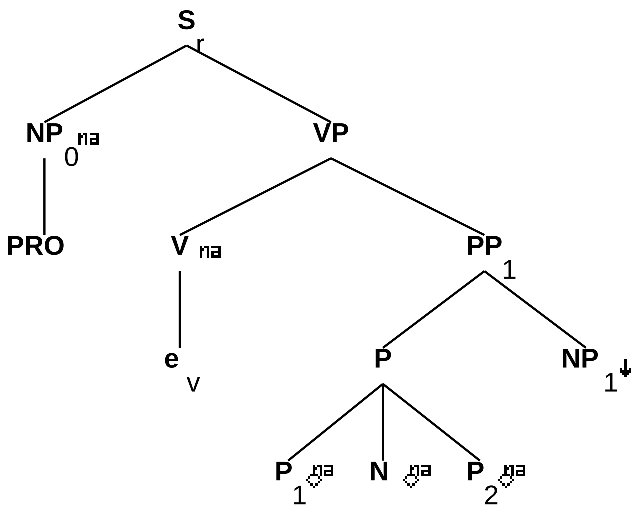
```
Gerund NP tree for predicative PPs.  This tree family, like other predicative
tree families, is anchored by the predicted object (here, the P), with the
verb, if any, adjoining in.  There is no corresponding D tree (*the being of
in the park; *the being in the park).

...John('s) being in charge of the whole army...
```

13.3 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> = -
PP_1.b:<assign-case> = P.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
P.b:<wh> = -



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

14 Tree "alphanx0PNaPnx1-PRO"

14.1 graphe



14.2 comments

```
Predicative PP w/ PRO subject.  This tree family, like other predicative
tree families, is anchored by the predicated object (here, the P), with the
verb, if any, adjoining in.

John wants [PRO to be in charge of the case].
```

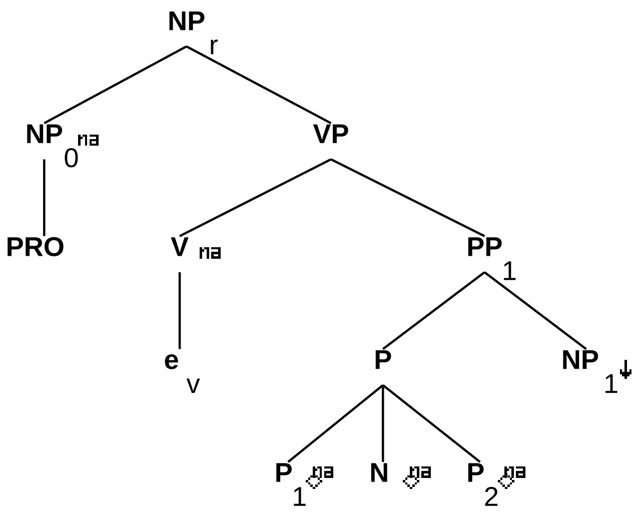
14.3 features

```
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
PP_1.b:<assign-case> = P.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
```

```
PP_1.b:<wh> = NP_1.t:<wh>
S_r.b:<control> = NP_0:<control>
VP.t:<mode> = inf/ger
```

15 Tree "alphaGnx0PNaPnx1-PRO"

15.1 graphe



15.2 comments

```
Gerund NP tree for predicative PPs w/ PRO subject.  This tree family, like other predicati
tree families, is anchored by the predicted object (here, the P), with the
verb, if any, adjoining in.

[PRO being in charge of the whole army] is important to Jim.
```

15.3 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> = -
PP_1.b:<assign-case> = P.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
P.b:<wh> = -
```

```
NP_r.b:<gerund> = +
```