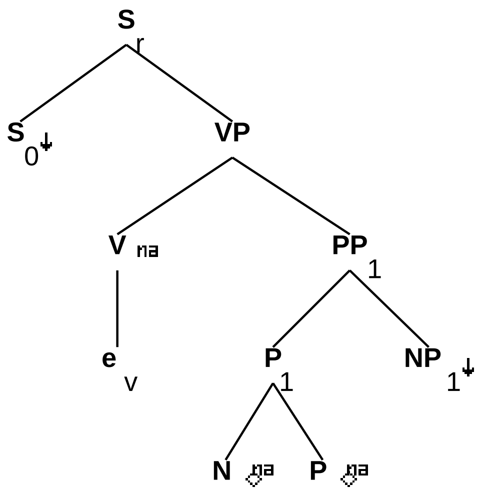
Family "Ts0NPnx1"

March 5, 2008

1 Tree "alphas0NPnx1"

1.1 graphe



1.2 comments

```
Declarative tree for predicative PPs that take sentential subjects.
The sentential subjects can be indicative or infinitive with comps
of that/whether/for/nil, although nil can only co-occur with the infinitive.
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: That John got in to medical school was thanks to his aunt's connections.
```

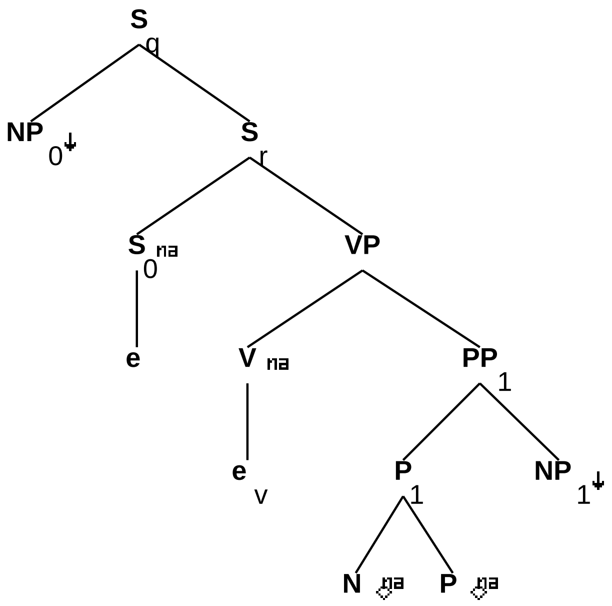
1.3 features

```
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>
S_0.t:<extracted> = -
S_0.t:<mode> = ind/inf
S_0.t:<comp> = that/whether/for/nil
S_0.t:<assign-comp> = inf_nil
S_0.t:<inv> = -
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:<agr pers> = 3
VP.b:<mode> = prep
VP.b:<assign-case> = acc
PP_1.b:<assign-case> = P_1.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
PP_1.b:<wh> = NP_1.t:<wh>
```

2 Tree "alphaW0s0NPnx1"

2.1 graphe



2.2 comments

Subject extraction tree for predicative PPs that take sentential subjects. The tree does only wh extraction, not topicalization, since subjects do not topicalize. The extracted S becomes an NP in its wh+ form, so this

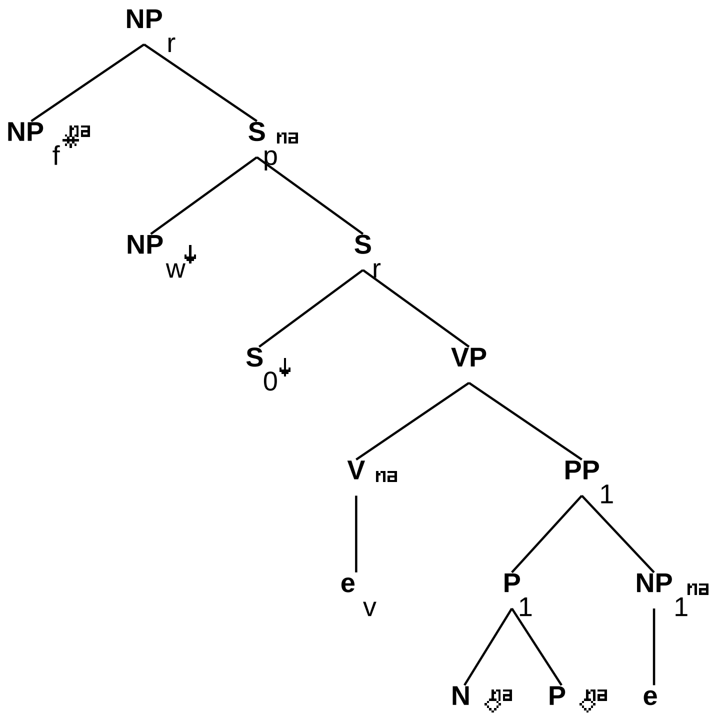
tree will parse the same sentence as W0nx0Pnx1, but we keep it here in spite of its redundancy because the underlying structure is different.
This tree family, like other predicative tree families, is anchored by the predicted object (here, the P), with the verb, if any, adjoining in.
EX: What was thanks to John's aunt's connections?

2.3 features

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

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



VP.t:<passive> = -
VP.t:<agr pers> = 3

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:<comp> = nil
S_r.b:<tense> = VP.t:<tense>
S_r.b:<inv> = -
NP_0:<trace> = S_0.t:<trace>
NP_0:<wh> = +
S_r.b:<agr> = VP.t:<agr>
S_r.b:<assign-case> = VP.t:<assign-case>
VP.b:<mode> = prep
VP.b:<assign-case> = acc
PP_1.b:<assign-case> = P_1.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
PP_1.b:<wh> = NP_1.t:<wh>
S_r.t:<conj> = nil
```

3 Tree "betaN1s0NPnx1"

3.1 graphe



3.2 comments

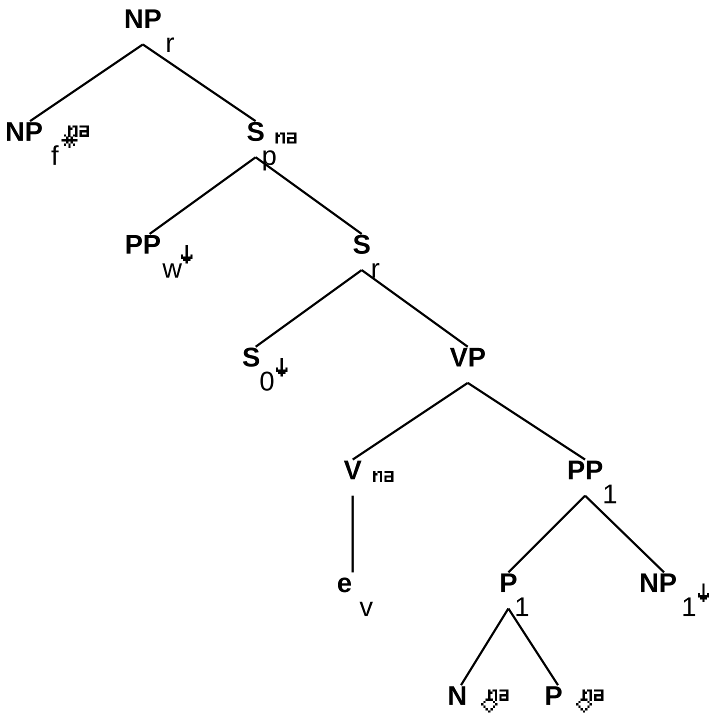
```
Relative clause tree for predicative PPs that take sentential subjects.
The NP inside the PP is what is extracted.
The sentential subjects can be indicative or infinitive with comps
of that/whether/for/nil, although nil can only co-occur with the infinitive.
This tree family, like other predicative tree families, is anchored by the
predicted object (here, the P), with the verb, if any, adjoining in.
EX: that she can walk is thanks to her crutches => I found her crutches that
that she can walk is thanks to
(these examples are stilted, but not so bad that we wanted to
exclude them)
```

3.3 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> = -
S_0.t:<extracted> = -
S_0.t:<mode> = ind/inf
S_0.t:<comp> = that/whether/for/nil
S_0.t:<assign-comp> = inf_nil
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
PP_1.b:<assign-case> = P_1.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
S_r.t:<conj> = nil

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>
```

4 Tree ”betaNpxs0NPnx1”

4.1 graphe



4.2 comments

Declarative tree for predicative PPs that take sentential subjects.
The sentential subjects can be indicative or infinitive with comps
of that/whether/for/nil, although nil can only co-occur with the infinitive.
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: That John got in to medical school was thanks to his aunt’s connections.

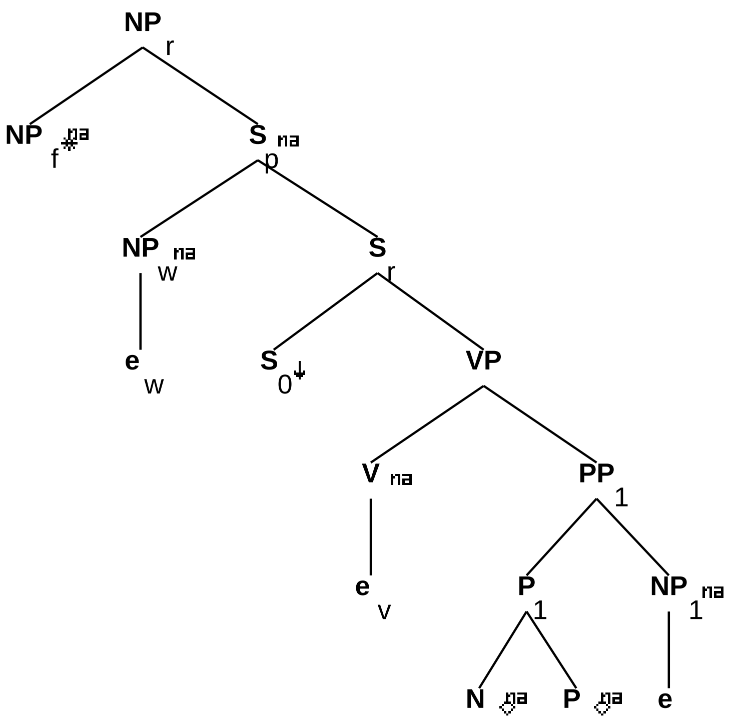
4.3 features

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



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

```
S_0.t:<extracted> = -
S_0.t:<mode> = ind/inf
S_0.t:<comp> = that/whether/for/nil
S_0.t:<assign-comp> = inf_nil
S_0.t:<inv> = -
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
PP_1.b:<assign-case> = P_1.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
P_1.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>
```

5 Tree "betaNc1s0NPnx1"

5.1 graphe



5.2 comments

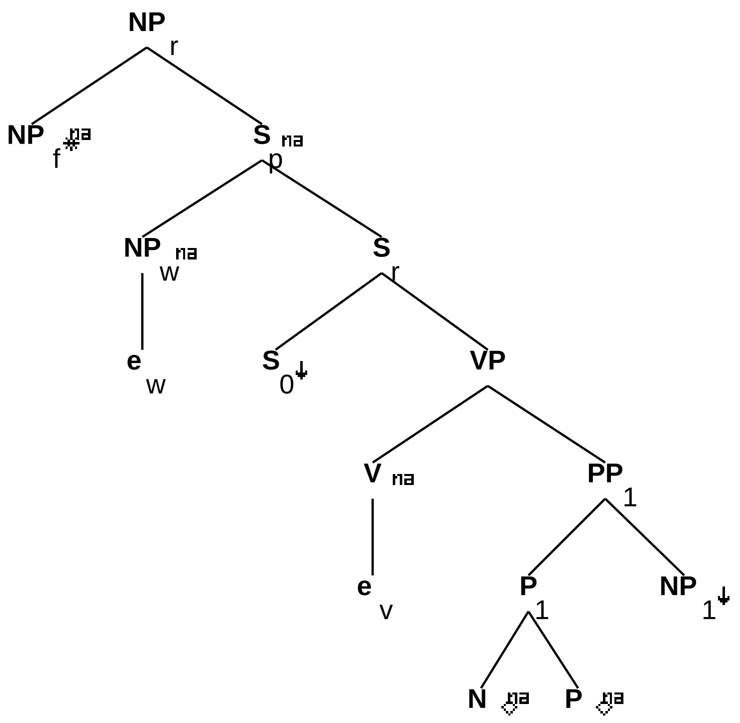
```
Relative clause tree for predicative PPs that take sentential subjects.
The NP inside the PP is what is extracted.
The sentential subjects can be indicative or infinitive with comps
of that/whether/for/nil, although nil can only co-occur with the infinitive.
This tree family, like other predicative tree families, is anchored by the
predicted object (here, the P), with the verb, if any, adjoining in.
EX: that she can walk is thanks to her crutches => I found her crutches that
that she can walk is thanks to
(these examples are stilted, but not so bad that we wanted to
exclude them)
```

5.3 features

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




VP.b:<compar> = -
S_r.b:<mode> = VP.t:<mode>
S_r.b:<tense> = VP.t:<tense>
S_r.t:<inv> = -
S_r.b:<inv> = -
S_0.t:<extracted> = -
S_0.t:<mode> = ind/inf
S_0.t:<comp> = that/whether/for/nil
S_0.t:<assign-comp> = inf_nil
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
PP_1.b:<assign-case> = P_1.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
S_r.t:<conj> = nil

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>
```

6 Tree "betaNcs0NPnx1"

6.1 graphe



6.2 comments

```
Declarative tree for predicative PPs that take sentential subjects.
The sentential subjects can be indicative or infinitive with comps
of that/whether/for/nil, although nil can only co-occur with the infinitive.
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: That John got in to medical school was thanks to his aunt's connections.
```

6.3 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_0.t:<extracted> = -
S_0.t:<mode> = ind/inf
S_0.t:<comp> = that/whether/for/nil
S_0.t:<assign-comp> = inf_nil
S_0.t:<inv> = -
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
PP_1.b:<assign-case> = P_1.t:<assign-case>
PP_1.b:<assign-case> = NP_1.t:<case>
P_1.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>
```