Génération d'un lexique syntaxique

Une application Perl pour le Traitement Automatique des Langues

Ingrid Falk

CNRS, Atilf, Nancy
en collaboration avec
Claire Gardent (CNRS), Guy Perrier (Université), Bruno Guillaume (INRIA)
travail financé dans le cadre du Projet « Ingénierie des Langues » CPER Lorrain
(Région Lorraine, Université, CNRS et INRIA)

Les Journées Perl 2006

Plan

Plan

- un lexique est une liste qui associe des infos à chaque mot d'une langue :

Exemple : l'entrée d'un dictionnaire pour bénéficier.

BÉNÉFICIER, verbe.
A. Bénéficier de qqc. [Le suj. désigne un animé ou un inanimé]
2. Jouir, profiter de quelque chose. Bénéficier d'une aide; faire bénéficier qqc. ou qqn de :

- un lexique syntaxique liste pour chaque usage d'un mot les constructions syntaxiques admises par cet usage.

Exemple : bénéficier

admets la construction : `suj v à obj`
mais n'admets pas : `suj v`

- un lexique est une liste qui associe des infos à chaque mot d'une langue :

Exemple : l'entrée d'un dictionnaire pour bénéficier.

BÉNÉFICIER, verbe.
A. Bénéficier de qqc. [Le suj. désigne un animé ou un inanimé]
2. Jouir, profiter de quelque chose. Bénéficier d'une aide; faire bénéficier qqc. ou qqn de :

- un lexique syntaxique liste pour chaque usage d'un mot les constructions syntaxiques admises par cet usage.

Exemple : bénéficier

admets la construction : `suj v à obj`
mais n'admets pas : `suj v`

Le cadre syntaxique (la construction syntaxique).

Définition

Une suite a0, v, a1, a2, etc. d'arguments, représentant chacun un ensemble de paires trait–valeur.

Un cadre syntaxique `[a0 v a1]` pour bénéficier

Que Max parte

```
a0[cat=p, fonc=sujet, mode=subj, comp='que']
```

bénéficie

```
v[cat=v, aux=avoir]
```

à Luc

```
a1[cat=n, fonc=datif, type-sem=humain,
prep='à']
```

Plan

Applications dans le Traitement Automatique des Langues :

- Analyse syntaxique et ses applications :
  - Traduction automatique.
  - Extraction d'information
  - . . .

Plan

Pour le Français . . .

Lefff production sémiautomatique,
disponible librement,
format directement compatible avec le TAL,

Dicovalence production manuelle,
librement utilisable depuis peu,
pas directement utilisable pour le TAL,

Ladl production manuelle,
disponible partiellement sous LGPL-LR,
format pas directement utilisable pour le TAL.

Plan

Objectif : réaliser un lexique syntaxique pour le Français

- à partir des tables du Ladl ($\approx$ 5000 verbes)
- dans un format compatible avec le TAL
- dans un format normalisé
- libre d'accès

Plan

Les tables du Ladl

- $\approx$ 50 tables
- chaque table regroupe des verbes ayant un comportement commun.
- un verbe par ligne
- infos détaillée sur l'usage par les $+$ et $-$
- conversion tables $\rightsquigarrow$ lexique syntaxique n'est pas évidente !

Les tables du Ladl

N0 =: Nhum	N0 =: Qu P	N0 =: le fait Qu P	N0 = V1 W	N0 = V1c W	5	aux =:avoir	aux =: être	N0 V		N1 =: Nhum	Prép Nhum = Ppv	N1 =: N-hum	N1 =: le fait Qu P	Loc N1 = à Nq	Loc N1 = dans Nq	Loc N1 = de Nq	N1 =: Ppv	N2 =: Qu P	N2 =: Qu Psubj	N2 = de V1 W	N2 = de V1c W	Tc =: passé	Tc =: futur	Vc =: devoir	Vc =: pouvoir	Vc =: savoir	N0 V de ce Qu P	
-	+	-	+	-	advenir	-	+	-	à	+	+	-	-	-	-	-	-	-	+	+	-	-	-	+	+	-	-	Il est advenu à Max que Léa a divorcé
+	+	+	+	+	agir	+	-	+	sur	+	-	+	+	-	-	-	-	-	+	+	+	+	-	+	+	+	-	Que Max l'ait grondé a agi sur Luc
-	+	+	+	-	bénéficier	+	-	-	à	+	+	-	-	-	-	-	-	-	+	+	-	+	+	+	+	+	-	Que Max parte bénéficie à Luc

Figure: Échantillon de la table 5

Plan

Le Format cible

- Pas de format normalisé pour les lexiques syntaxiques.
- $\rightsquigarrow$ format de sortie Perl

Exemple : un cadre syntaxique en Perl

```
[
 { arg => 'a0',
   fonc => 'sujet',
   cat => 'n' },

 { arg => 'v',
   lemme => "b\x{e9}n\x{e9}ficier",
   cat => 'v',
   aux => 'avoir' },

 { arg => 'a1',
   fonc => 'oblique',
   cat => 'n',
   prep => "\x{e0}",
   type_sem => 'humain',
   cliticisable => 'vrai'}
]
```

Plan

Comment ?

En Perl à l'aide de :

- `XML::LibXML`
- `Graph::ReadWrite`
- `Graph`
- `Parse::RecDescent`
- `...`

Pourquoi Perl ?

Plan

Résultats

Un lexique syntaxique pour le Français

- $\approx$ 5000 verbes avec $\approx$ 28000 constructions.
- $\approx$ 550 cadres syntaxiques.
- nombre moyen de cadres par verbe : $6,8$.

En

- $\approx$ 10 mois/ingénieur.
- 373 de scripts et modules Perl.
- $\approx$ 90000 lignes de code.

Plan

Le déroulement dans la production du lexique

1. tables en format excel $\rightsquigarrow$ format xml (gnumeric)
2. Pour chaque table :
   - Modélisation : production manuelle du graphe de la table (graphviz dot)
   - Génération du lexique (Perl) :
     - Interpréter le graphe : quels cadres syntaxiques peut-il engendrer ?
     - Pour chaque verbe ($\approx$ ligne de la table) générer ses cadres syntaxiques.

Le déroulement dans la production du lexique

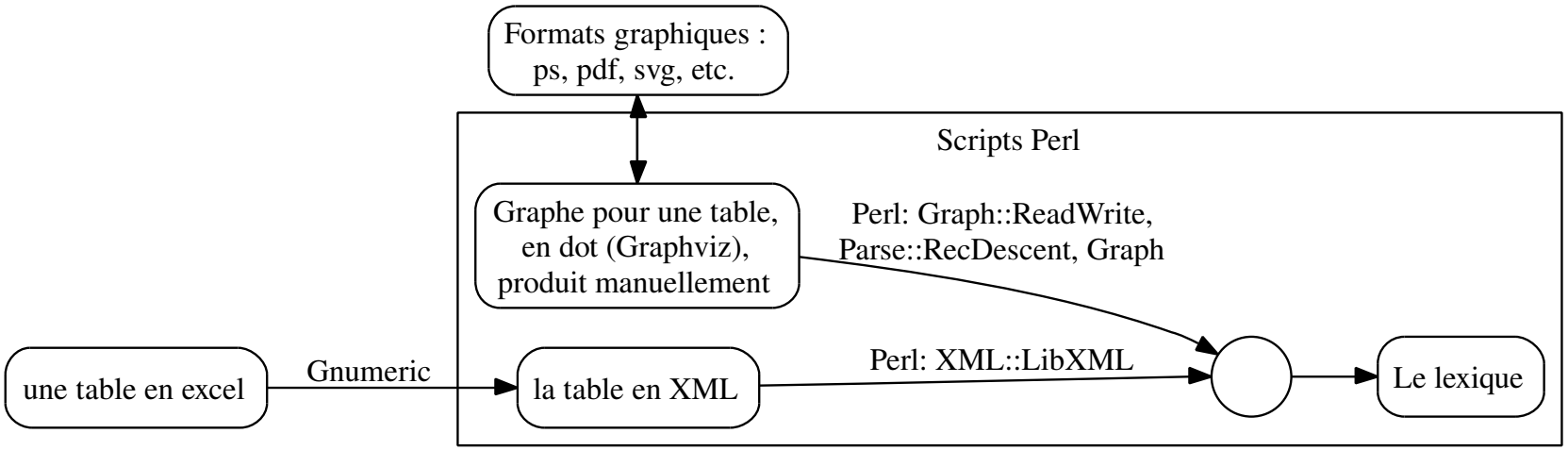


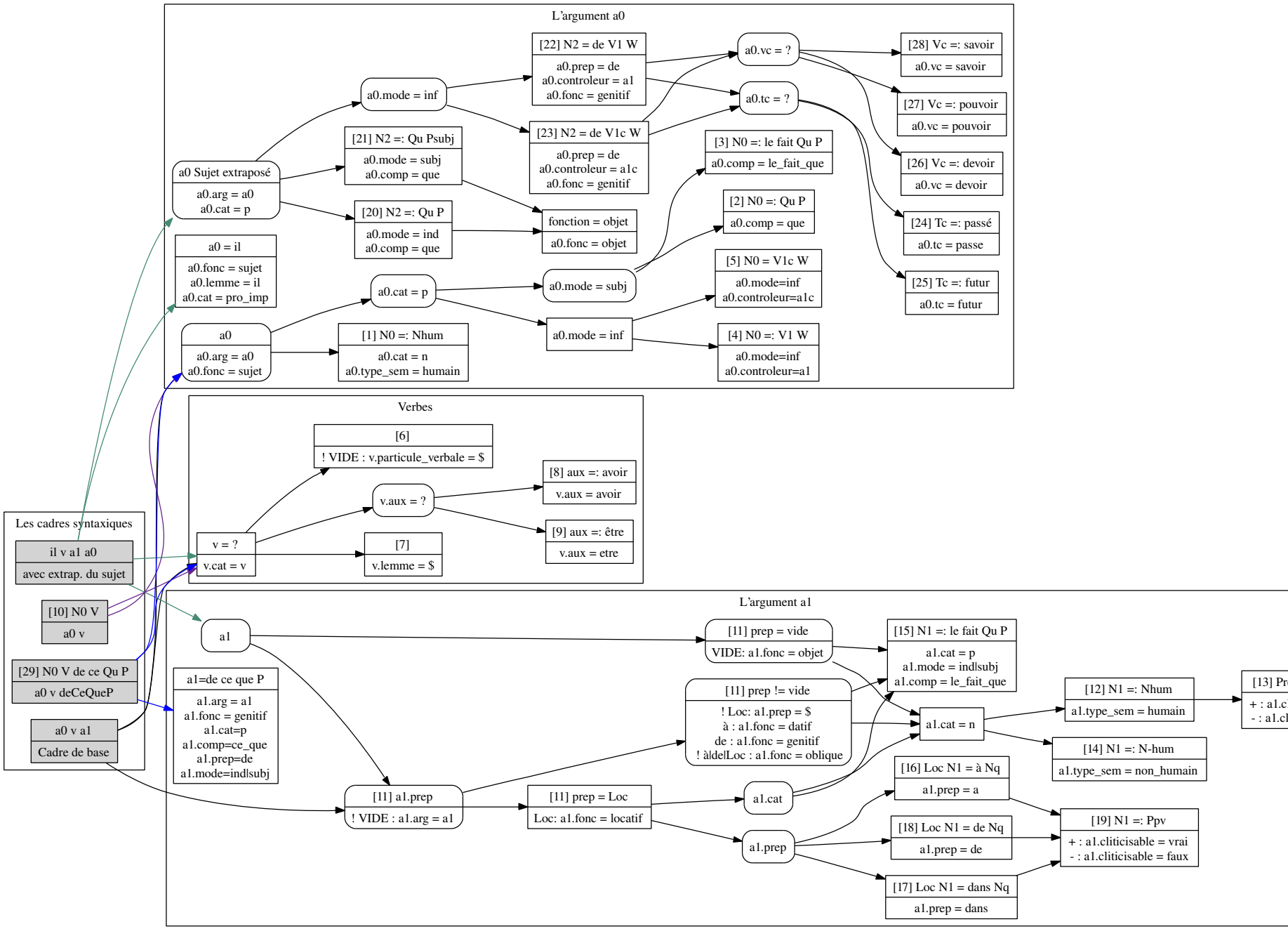
Figure: Le déroulement

Plan

Le graphe d'une table

- Vise à rendre explicite
  - dépendences implicites entre les colonnes de la table.
  - connaissances provenant de la description de la table, d'autres publications, de discussions.
- produit manuellement, format dot

LADL table 5



▸ Description du graphe

Description du graphe

Graphe orienté, acyclique, *ET/OU*

Les nœuds

OU	ovales, disjonction d'informations entre leurs nœuds fils.
ET	carrée, conjonction d'informations
cadre	racines, boîtes grisés, cadres syntaxiques associées à une table ▸ exemple
arguments	les nœuds fils des nœuds *cadre* ▸ exemple

Les nœuds

Les informations associés aux nœuds sont codées dans leurs étiquettes :

Les étiquettes des nœuds

▸ exemple

[c] une colonne de la table.

conditions sur les valeurs d'une ligne de la table pour la colonne [*c*].

paires traits - valeurs à rajouter à l'argument si la *condition* est satisfaite pour une ligne de la table.

Plan

Calcul des cadres syntaxiques

Cadres syntaxiques

Nœuds racine du graphe. ▸ exemple

Calculer les arguments

Calculer les chemins partant des nœuds arguments. ▸ exemple

Calculer les paires trait–valeur des arguments

Décoder les étiquettes des nœuds sur le chemin. ▸ exemple

En Perl

dot ⟶ `Graph`

`Graph::ReadWrite` ⇝ `Graph`

Calculer les chemins

`Graph`

Décoder les étiquettes des nœuds

`Parse::RecDescent`

En Perl

Décoder les étiquettes des nœuds

▸ exemple

pour chaque ligne dans la partie inférieure générer un objet Traits avec une fonction :

```
sub {
      @_ = valeurs d'une ligne de la table
      return  (trait => 'valeur');
    }
```

Après le parcours du graphe

Résultat

- les cadres engendrées par la table
- chaque cadre est une suite d'arguments
- chaque argument est une liste d'objets Traits

Plan

Rallier le graphe et la table

On parcours les lignes de la table

En Perl

- `XML::LibXML` avec `XPath`
- pour chaque ligne de la table (= verbe) et pour chaque Trait on instancie la paire trait–valeur correspondante, si approprié.

Et. . . on y est !

++++++++++++++++ bénéficier ++++++++++++++++

```
5  v[pred='bénéficier<suj:scompl,obja:à-sn>', @avoir, cat=v]
5  v[pred='bénéficier<suj:sinf,obja:à-sn>', @avoir, cat=v]
5  v[pred='bénéficier<suj:sn,obja:à-sn,obj:scompl>', @avoir, @il
5  v[pred='bénéficier<suj:sn,obja:à-sn,objde:de-sinf>', @avoir,
8  v[pred='bénéficier<suj:scompl,objde:de-scompl>', cat=v]
8  v[pred='bénéficier<suj:scompl,objde:de-sinf>', cat=v]
8  v[pred='bénéficier<suj:scompl,objde:de-sn>', cat=v]
8  v[pred='bénéficier<suj:sinf,objde:de-scompl>', cat=v]
8  v[pred='bénéficier<suj:sinf,objde:de-sinf>', cat=v]
8  v[pred='bénéficier<suj:sinf,objde:de-sn>', cat=v]
8  v[pred='bénéficier<suj:sn,objde:de-scompl>', cat=v]
8  v[pred='bénéficier<suj:sn,objde:de-sinf>', cat=v]
8  v[pred='bénéficier<suj:sn,objde:de-sn>', cat=v]
```

Beaucoup de travaux après la création du lexique :

en cours

- Analyse et validation des résultats
- Comptages, statistiques, représentations graphiques
- Comparaisons avec d'autres lexiques

. . . et pour l'avenir

- mieux définir le format de sortie
- des interfaces pour gérer le(s) graphe(s)
- des interfaces pour modifier et maintenir le lexique

Beaucoup de travaux après la création du lexique :

en cours

- Analyse et validation des résultats
- Comptages, statistiques, représentations graphiques
- Comparaisons avec d'autres lexiques

. . . et pour l'avenir

- mieux définir le format de sortie
- des interfaces pour gérer le(s) graphe(s)
- des interfaces pour modifier et maintenir le lexique

Pourquoi Perl s'est avèré un bon choix

- Souplesse : les méthodes n'étaient pas claires dès le début
- La richesse et puissance des modules, en particulier
  - `Graph`,
  - `XML::LibXML` et
  - `Parse::RecDescent`

Conclusion

- Grace à Perl nous avons rendu possible une tache assez complexe et difficile !

Références I

Théorie

SynLex `http://www.loria.fr/~gardent/ladl/index.php`

Ladl `http://infolingu.univ-mlv.fr/DonneesLinguistiques/Lexiques-Grammaires/Presentation.html`

Implémentation : modules et scripts Perl spécifiques (non CPAN)

- `http://www.loria.fr/~falk/Ladl.html`
- `http://www.loria.fr/~falk/Ladl.html#tools`

Implémentation : modules du CPAN utilisés

Références II

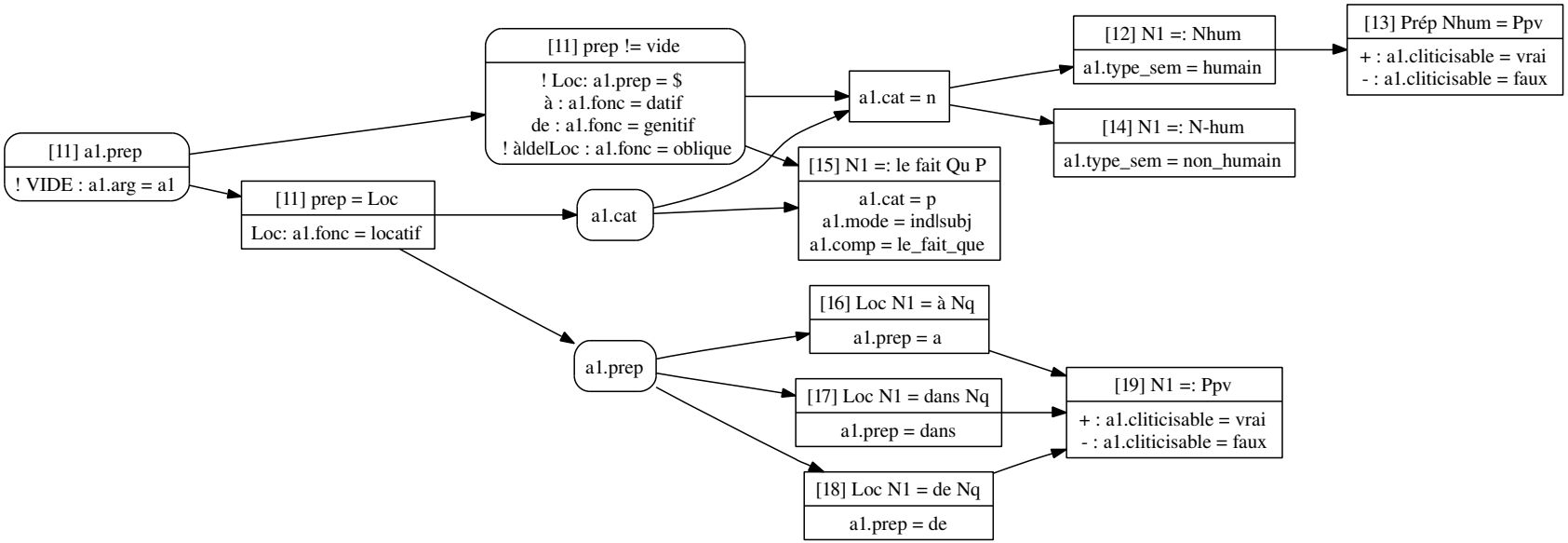
manipuler des graphes `Graph`, `http://search.cpan.org/author/JHI/Graph-0.80/lib/Graph.pod`

dot ⇝ `Graph` `Graph::ReadWrite`, `http://search.cpan.org/~neilb/Graph-ReadWrite-2.00/`

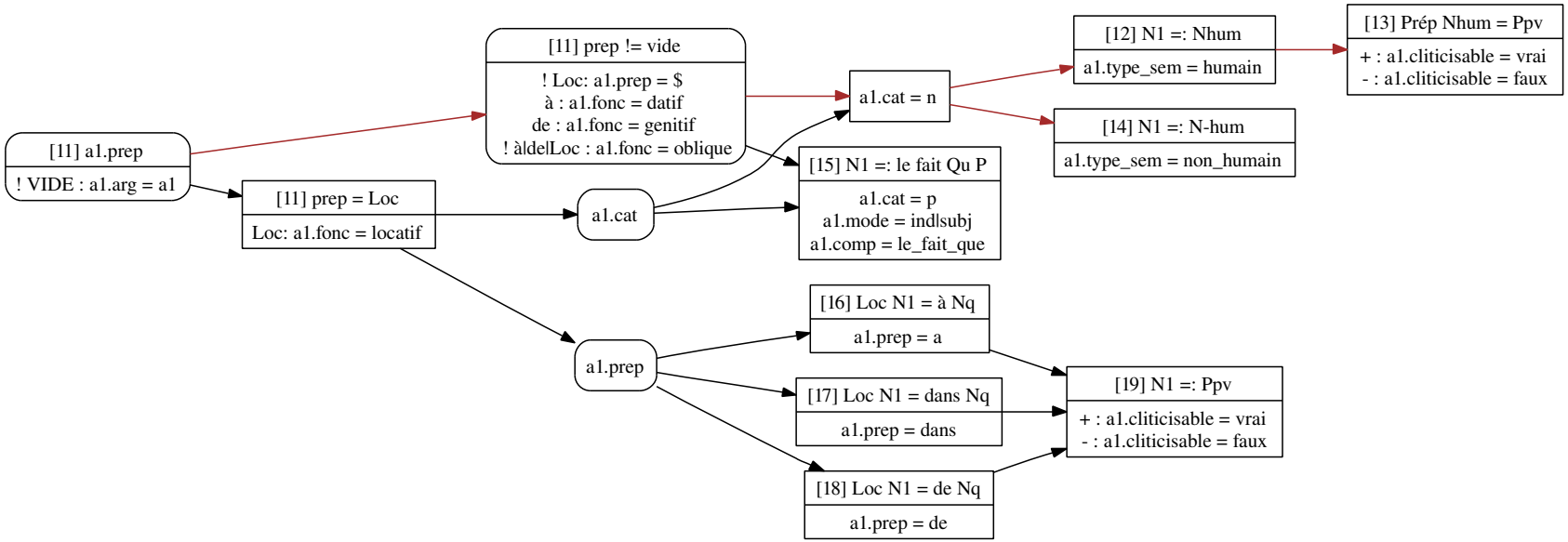
analyse `Parse::RecDescent`, `http://search.cpan.org/~dconway/Parse-RecDescent-1.94/lib/Parse/RecDescent.pod`

XML `XML::LibXML`, `http://search.cpan.org/author/PAJAS/XML-LibXML-1.62001/LibXML.pod`

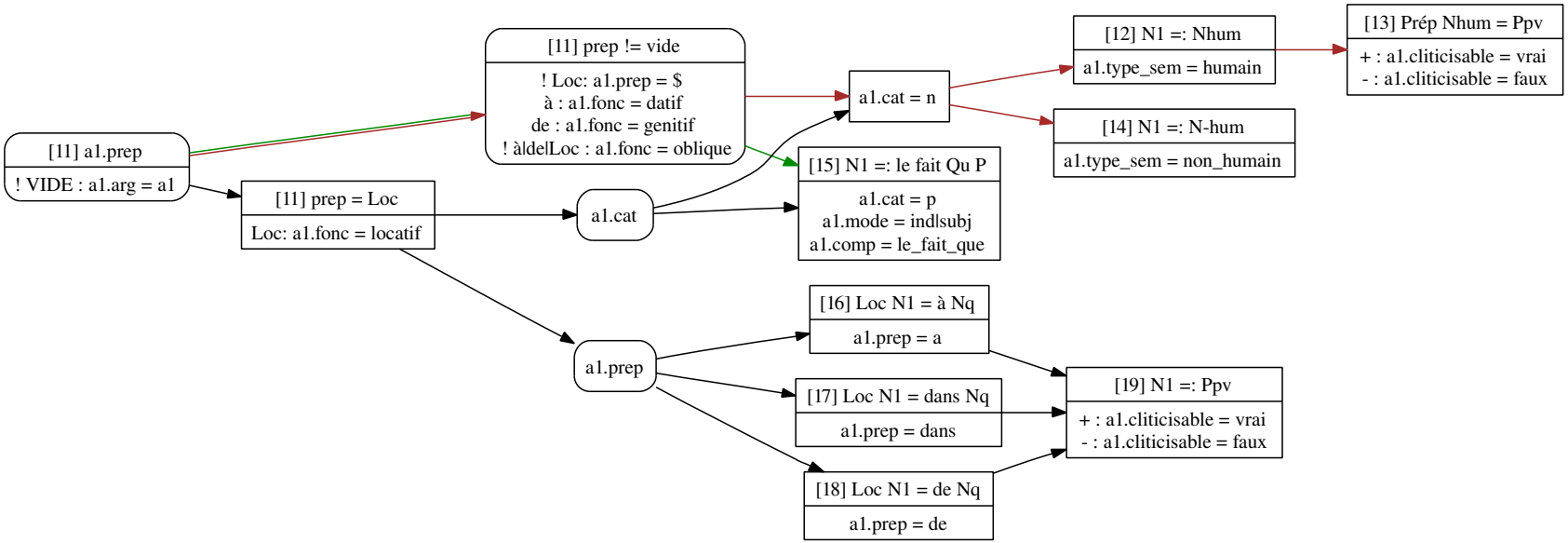
Exemple : chemins dans un graphe *ET/OU*



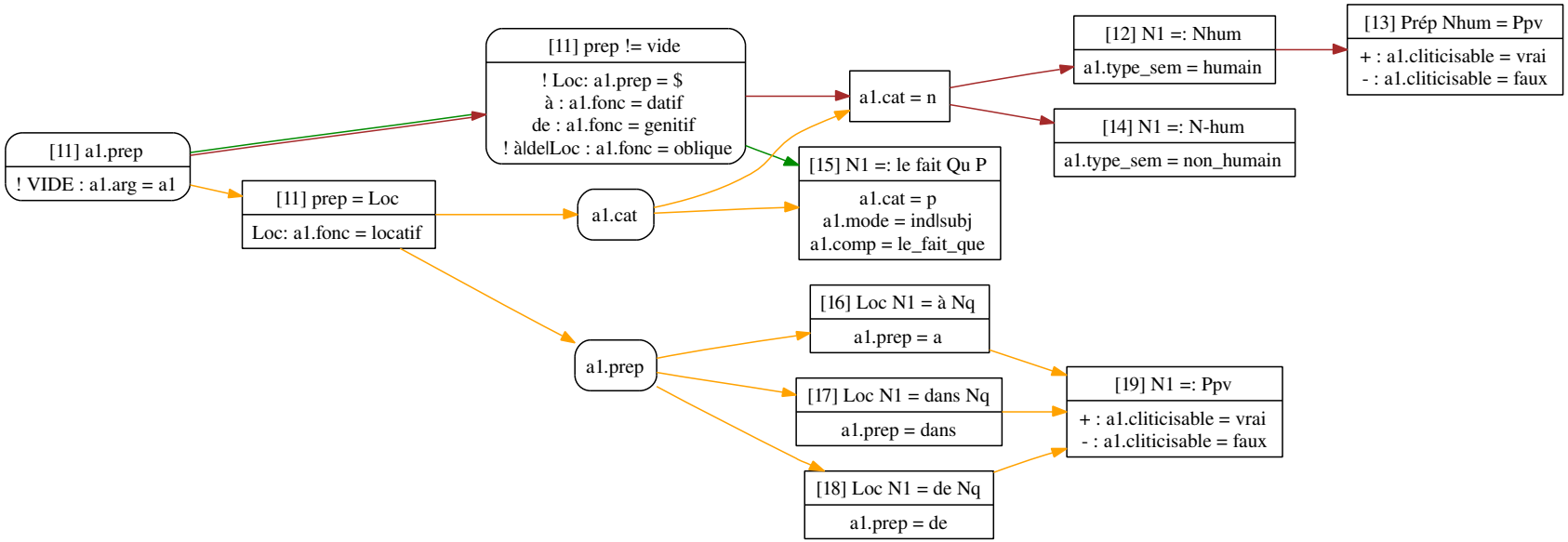
Exemple : chemins dans un graphe *ET/OU*



Exemple : chemins dans un graphe *ET/OU*

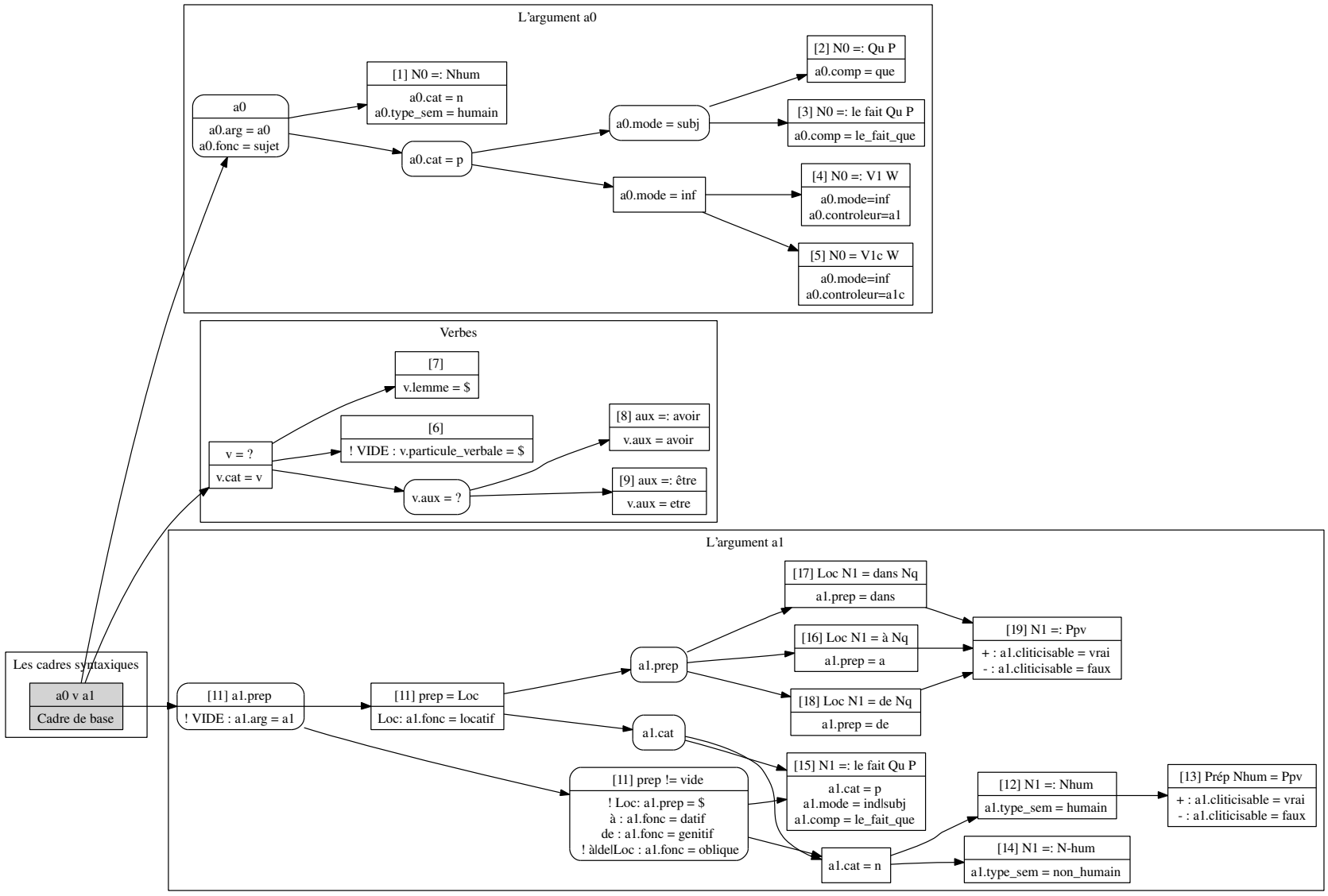


Exemple : chemins dans un graphe *ET/OU*

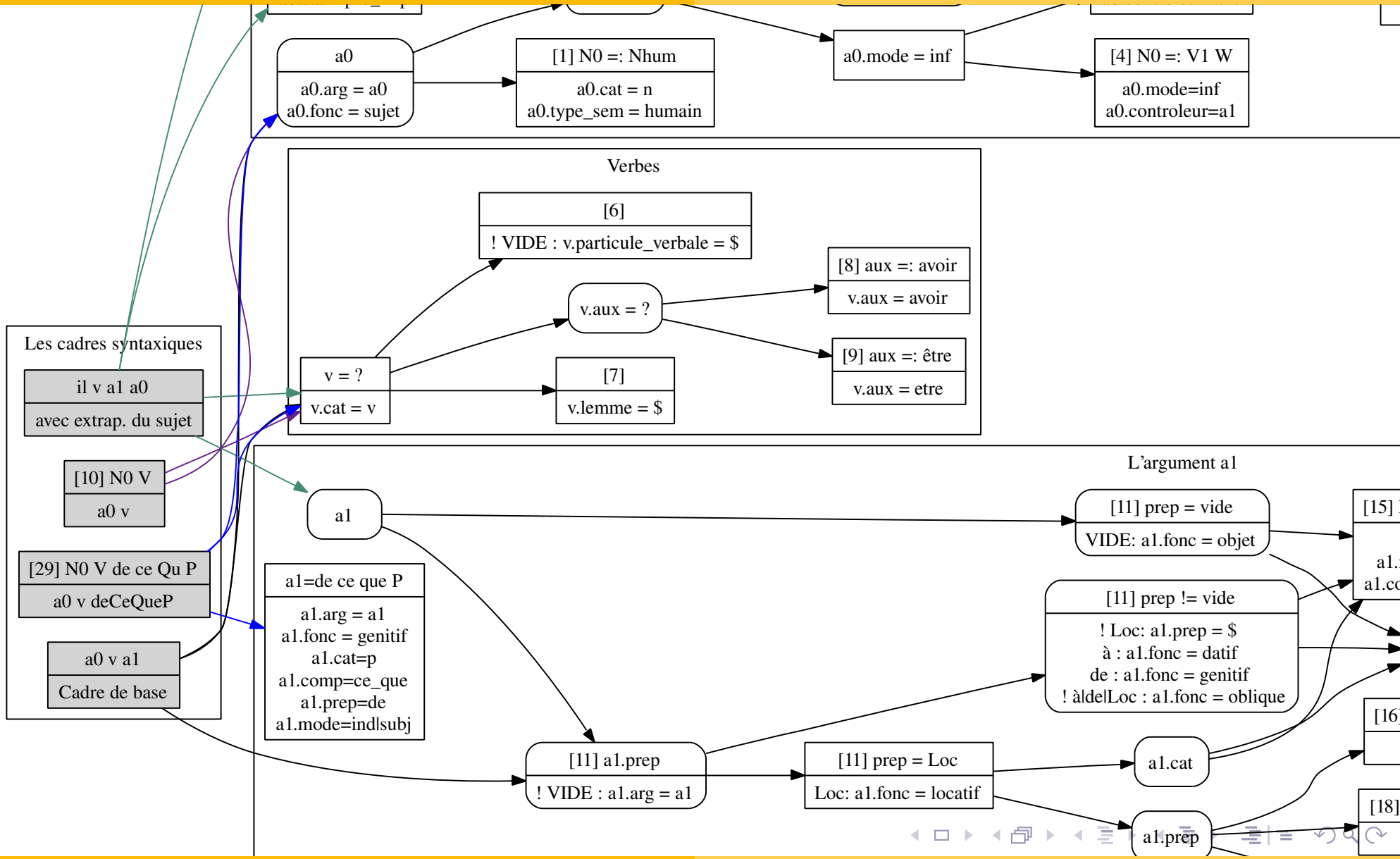


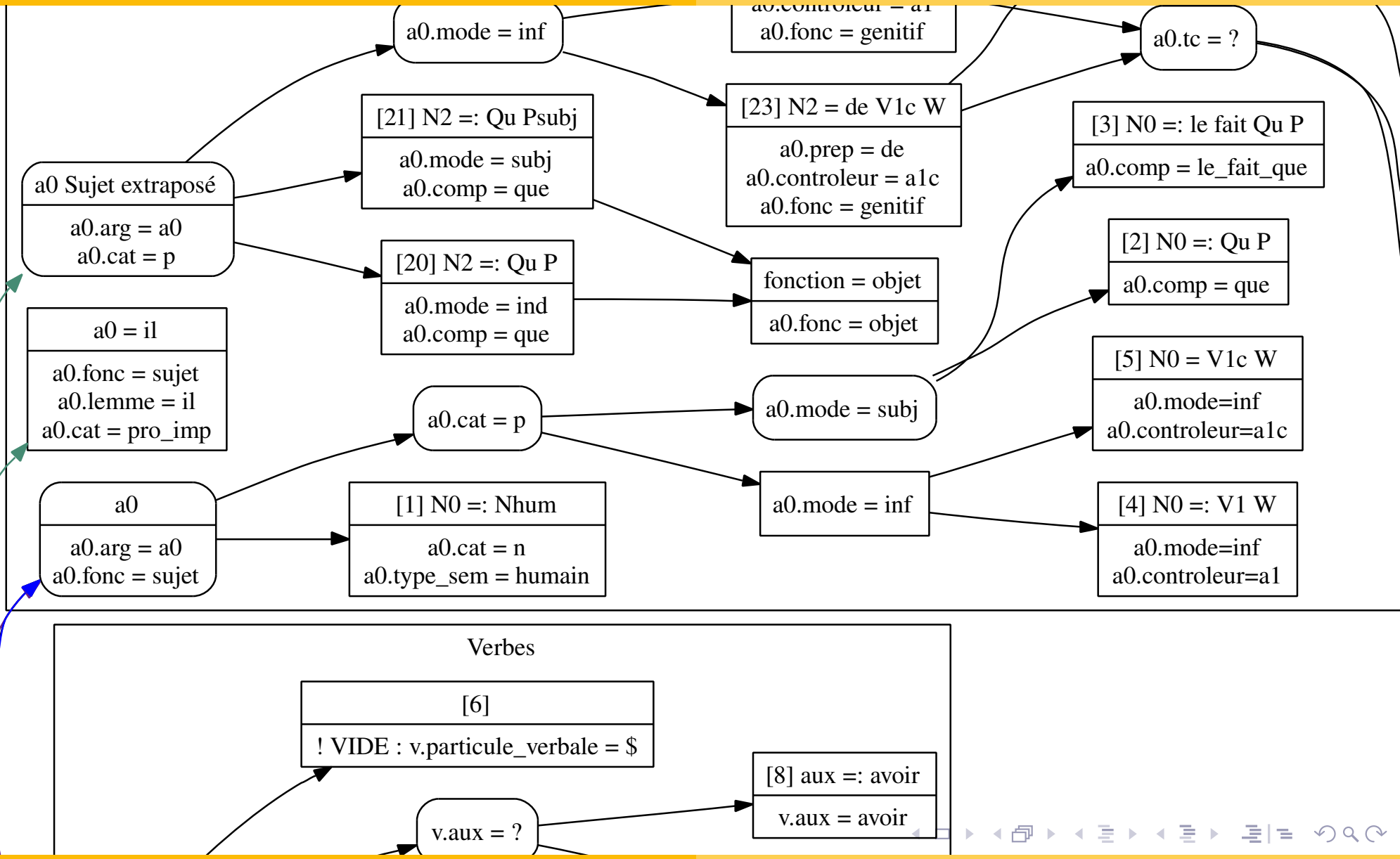
retour

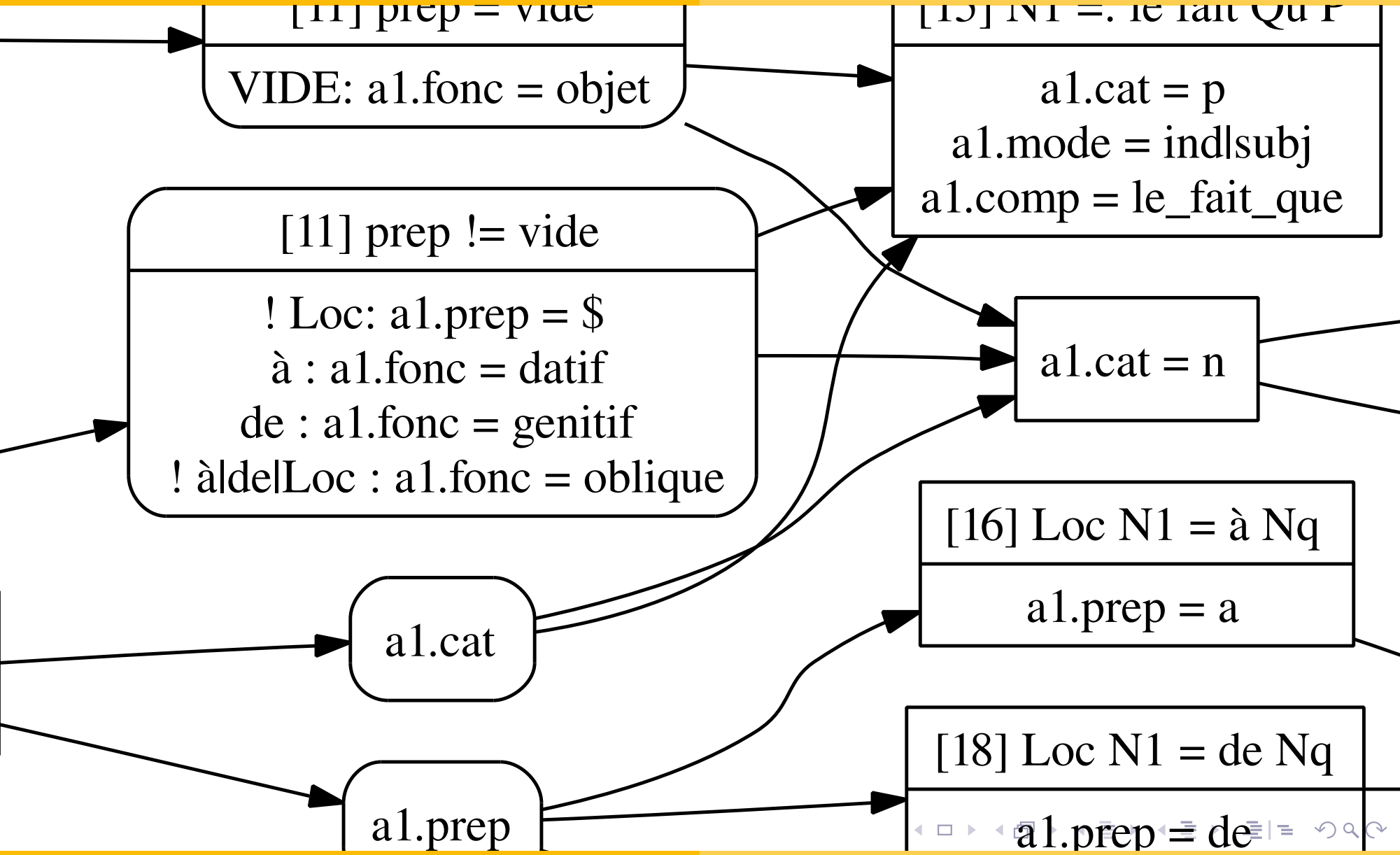
LADL table 5



retour


a0
a0.arg = a0
a0.fonc = sujet
[1] N0 =: Nhum
a0.cat = n
a0.type_sem = humain
a0.mode = inf
[4] N0 =: V1 W
a0.mode=inf
a0.controleur=a1
Verbes
[6]
! VIDE : v.particule_verbale = $
v.aux = ?
[8] aux =: avoir
v.aux = avoir
[9] aux =: être
v.aux = etre
v = ?
v.cat = v
[7]
v.lemme = $
Les cadres syntaxiques
il v a1 a0
avec extrap. du sujet
[10] N0 V
a0 v
[29] N0 V de ce Qu P
a0 v deCeQueP
a0 v a1
Cadre de base
L'argument a1
a1
a1=de ce que P
a1.arg = a1
a1.fonc = genitif
a1.cat=p
a1.comp=ce_que
a1.prep=de
a1.mode=ind|subj
[11] prep = vide
VIDE: a1.fonc = objet
[11] prep != vide
! Loc: a1.prep = $
à : a1.fonc = datif
de : a1.fonc = genitif
! à|de|Loc : a1.fonc = oblique
[11] a1.prep
! VIDE : a1.arg = a1
[11] prep = Loc
Loc: a1.fonc = locatif
a1.cat
a1.prep


a0.mode = inf
a0.fonc = genitif
a0.tc = ?
[23] N2 = de V1c W
a0.prep = de
a0.controleur = a1c
a0.fonc = genitif
[21] N2 =: Qu Psubj
a0.mode = subj
a0.comp = que
[3] N0 =: le fait Qu P
a0.comp = le_fait_que
a0 Sujet extraposé
a0.arg = a0
a0.cat = p
[20] N2 =: Qu P
a0.mode = ind
a0.comp = que
fonction = objet
a0.fonc = objet
[2] N0 =: Qu P
a0.comp = que
a0 = il
a0.fonc = sujet
a0.lemme = il
a0.cat = pro_imp
a0.cat = p
a0.mode = subj
[5] N0 = V1c W
a0.mode=inf
a0.controleur=a1c
a0
a0.arg = a0
a0.fonc = sujet
[1] N0 =: Nhum
a0.cat = n
a0.type_sem = humain
a0.mode = inf
[4] N0 =: V1 W
a0.mode=inf
a0.controleur=a1
Verbes
[6]
! VIDE : v.particule_verbale = $
[8] aux =: avoir
v.aux = avoir
v.aux = ?


[11] prep = vide
VIDE: a1.fonc = objet
[13] N1 =: le fait Qu P
a1.cat = p
a1.mode = ind|subj
a1.comp = le_fait_que
[11] prep != vide
! Loc: a1.prep = $
à : a1.fonc = datif
de : a1.fonc = genitif
! à|de|Loc : a1.fonc = oblique
a1.cat = n
[16] Loc N1 = à Nq
a1.prep = a
a1.cat
[18] Loc N1 = de Nq
a1.prep
a1.prep = de