Comment obtenir plus des Méta-Grammaires

ou comment construire des grammaires TAG/TIG compactes avec des arbres factorisés

François Thomasset & Éric de la Clergerie
Eric.De_La_Clergerie@inria.fr

ATOLL – INRIA

TALN 2005
Dourdan – Mardi 7 Juin 2005

Motivations

⇒ Des systèmes pour construire des analyseurs syntaxiques,
dont DyALog ⇒ formalismes hybrides TAG+TIG, DCG, RCG, ∼ Prolog

- Mais manque de ressources : lexiques et grammaires à large couverture
  ⇒ pb pour la campagne d'évaluation EASy
- Développement de grammaires
  - processus complexe et long avec pb de cohérence et maintenance
  - taille importante des grammaires *lexicalisables* (TAG : 1000 à 10000 arbres)
  - redondance importante dans les grammaires
- Solution : Méta-Grammaire [Candito]
  - Niveau plus abstrait et compact de description linguistique
  - Développement plus rapide, maintenance facilitée
  - Génération d'une grammaire, en général de grande taille
    en particulier à cause de sous-catégorisations x extractions

Motivations

- Des systèmes pour construire des analyseurs syntaxiques,
  dont DyALog $\Rightarrow$ formalismes hybrides TAG+TIG, DCG, RCG, $\sim$ Prolog

$\Rightarrow$ Mais manque de ressources : lexiques et grammaires à large couverture
$\Rightarrow$ pb pour la campagne d'évaluation EASy

- Développement de grammaires
  - processus complexe et long avec pb de cohérence et maintenance
  - taille importante des grammaires *lexicalisables* (TAG : 1000 à 10000 arbres)
  - redondance importante dans les grammaires

- Solution : Méta-Grammaire [Candito]
  - Niveau plus abstrait et compact de description linguistique
  - Développement plus rapide, maintenance facilitée
  - Génération d'une grammaire, en général de grande taille
    en particulier à cause de sous-catégorisations x extractions

Motivations

- Des systèmes pour construire des analyseurs syntaxiques,
  dont DyALog $\Rightarrow$ formalismes hybrides TAG+TIG, DCG, RCG, $\sim$ Prolog
- Mais manque de ressources : lexiques et grammaires à large couverture
  $\Rightarrow$ pb pour la campagne d'évaluation EASy

$\Rightarrow$ Développement de grammaires

- processus complexe et long avec pb de cohérence et maintenance
- taille importante des grammaires *lexicalisables* (TAG : 1000 à 10000 arbres)
- redondance importante dans les grammaires

- Solution : Méta-Grammaire [Candito]
  - Niveau plus abstrait et compact de description linguistique
  - Développement plus rapide, maintenance facilitée
  - Génération d'une grammaire, en général de grande taille
    en particulier à cause de sous-catégorisations x extractions

Motivations

- Des systèmes pour construire des analyseurs syntaxiques,
  dont DyALog $\Rightarrow$ formalismes hybrides TAG+TIG, DCG, RCG, $\sim$ Prolog
- Mais manque de ressources : lexiques et grammaires à large couverture
  $\Rightarrow$ pb pour la campagne d'évaluation EASy
- Développement de grammaires
  - processus complexe et long avec pb de cohérence et maintenance
  - taille importante des grammaires *lexicalisables* (TAG : 1000 à 10000 arbres)
  - redondance importante dans les grammaires

$\Rightarrow$ Solution : Méta-Grammaire [Candito]

- Niveau plus abstrait et compact de description linguistique
- Développement plus rapide, maintenance facilitée
- Génération d'une grammaire, en général de grande taille
  en particulier à cause de sous-catégorisations x extractions

Motivations

- Des systèmes pour construire des analyseurs syntaxiques,
  dont DyALog $\Rightarrow$ formalismes hybrides TAG+TIG, DCG, RCG, $\sim$ Prolog
- Mais manque de ressources : lexiques et grammaires à large couverture
  $\Rightarrow$ pb pour la campagne d'évaluation EASy
- Développement de grammaires
  - processus complexe et long avec pb de cohérence et maintenance
  - taille importante des grammaires *lexicalisables* (TAG : 1000 à 10000 arbres)
  - redondance importante dans les grammaires
- Solution : Méta-Grammaire [Candito]
  - Niveau plus abstrait et compact de description linguistique
  - Développement plus rapide, maintenance facilitée
  - Génération d'une grammaire, en général de grande taille
    en particulier à cause de sous-catégorisations x extractions

Question : Obtention de grammaires compactes à partir de MG ?

Des opérateurs pour des arbres factorisés

DyALog propose des opérateurs réguliers pour factoriser les arbres :

- disjonctions $T[t_1;t_2] \equiv T[t_1] \cup T[t_2]$
- répetitions (Étoile de Kleene) $t@* \equiv \text{kleene}_t(\epsilon) \cup \text{kleene}_t(t, \text{kleene}_t)$
- entrelacements (ordre libre entre séquences de noeuds)
  $(t_1, t_2)\#\#t_3 \equiv (t_1, t_2, t_3; t_1, t_3, t_2; t_3, t_1, t_2)$
- optionalité (noeud optionel) $t? \equiv (t; \epsilon)$
- gardes (noeuds gardés) $T[G_+, t; G_-] \equiv T[t].\sigma_+ \cup T[\epsilon].\sigma_-$
  gardes : formules booléennes sur des équations entre chemins

Des opérateurs pour des arbres factorisés

DyALog propose des opérateurs réguliers pour factoriser les arbres :

- disjonctions $T[t_1;t_2] \equiv T[t_1] \cup T[t_2]$
- répetitions (Étoile de Kleene) $t@* \equiv \text{kleene}_t(\epsilon) \cup \text{kleene}_t(t, \text{kleene}_t)$
- entrelacements (ordre libre entre séquences de noeuds)
  $(t_1, t_2)\#\#t_3 \equiv (t_1, t_2, t_3; t_1, t_3, t_2; t_3, t_1, t_2)$
- optionalité (noeud optionel) $t? \equiv (t; \epsilon)$
- gardes (noeuds gardés) $T[G_+, t; G_-] \equiv T[t].\sigma_+ \cup T[\epsilon].\sigma_-$
  gardes : formules booléennes sur des équations entre chemins

Ces opérateurs

- ne changent pas le pouvoir d'expression ou la complexité
- peuvent être élimés par expansion
  mais taille résultante exponentielle en le nombre d'opérateurs
- plus efficace de les évaluer sans expansion
  $\Rightarrow$ analyses plus naturelles
- opérateurs très génériques (non spécifiques TAG ou TIG)

Les Méta-Grammaires par l'exemple

Definition (Méta-Grammaire)

Description modulaire par classes regroupant des contraintes, avec héritage

Les Méta-Grammaires par l'exemple

Definition (Méta-Grammaire)

Description modulaire par classes regroupant des contraintes, avec héritage

```
class collect_real_subject_canonical {
 <: collect_real_subject;
 $arg.extracted = value(~cleft);
 S >> VSubj; V >> psubj;
 VSubj < V; VMod < psubj;
 node psubj: [cat:N2, id:subject,
              top:[wh:-,sat:+]];
 - psubj::agreement; psubj = psubj::N;
 psubj =>
  node(Infl).bot.inv = value(+),
  $arg.extracted = value(-),
  $arg.real = value(N2),
  desc.extraction = value(~-),
  node(V).top.mode= value(~inf|imp|...);
 ~psubj=> node(Infl).bot.inv = value(~+);
}
```

- Héritage (<:)
- Contraintes
  - dominance (>> et >>+)
  - précédence (<)
  - égalité (=)
  - Décorations (FS)
    - ★ noeuds
    - ★ classe
  - Éq. entre chemins (.)
    - ★ noeuds (node psubj)
    - ★ classe (desc)
    - ★ variable ($arg)
- Ressources + / Besoins -
  - Espace de noms (::)
- Gardes (=>)

Compiler la méta-grammaire

Compilateur MGCOMP, développé avec DYALOG

Compiler la méta-grammaire

Compilateur MGCOMP, développé avec DYALOG

Étape 1 : Classes terminales

Héritage des contraintes par les classes terminales (+ vérif contraintes)

Compiler la méta-grammaire

Compilateur MGCOMP, développé avec DYALOG

Étape 1 : Classes terminales

Héritage des contraintes par les classes terminales (+ vérif contraintes)

Étape 2 : Classes neutres

- Croisement des classes terminales pour neutraliser ressources & besoins
  - $C_1[-R \cup \mathcal{K}_1] x C_2[+R \cup \mathcal{K}_2] = (C_1 x C_2)[=R \cup \mathcal{K}_1 \cup \mathcal{K}_2]$
  - (Espace de nom) $\Rightarrow$ import classe productrice avec renommage
    $C_1[-N::R \cup \mathcal{K}_1] x C_2[+R \cup \mathcal{K}_2] = (C_1 x N::C_2)[=N::R \cup \mathcal{K}_1 \cup N::\mathcal{K}_2]$
- Réduction des gardes (quand possible)
- Vérification des contraintes

Compiler la méta-grammaire

Compilateur MGCOMP, développé avec DYALOG

Étape 1 : Classes terminales

Héritage des contraintes par les classes terminales (+ vérif contraintes)

Étape 2 : Classes neutres

- Croisement des classes terminales pour neutraliser ressources & besoins
  - $C_1[-R \cup \mathcal{K}_1] x C_2[+R \cup \mathcal{K}_2] = (C_1 x C_2)[=R \cup \mathcal{K}_1 \cup \mathcal{K}_2]$
  - (Espace de nom) $\Rightarrow$ import classe productrice avec renommage
    $C_1[-N::R \cup \mathcal{K}_1] x C_2[+R \cup \mathcal{K}_2] = (C_1 x N::C_2)[=N::R \cup \mathcal{K}_1 \cup N::\mathcal{K}_2]$
- Réduction des gardes (quand possible)
- Vérification des contraintes

Étape 3 : Arbres TAG

Utilisation des contraintes des classes neutres pour construire les arbres

- précédence sous spécifiée entre noeuds frères $\Rightarrow$ entrelacement

La grammaire en quelques tables

Classes	Arbres	Init.	Aux.	Aux. Env.	Aux. Gauches	Aux. Droits
191	133=126+7	44	89	12	29	48

Distribution par catégories d'arbres

non ancrés	v	coo	adv	adj	csu	prep	aux	np	nc	det	pro
50	27	12	10	8	4	3	2	2	1	1	1

Distribution par la catégorie de l'ancre

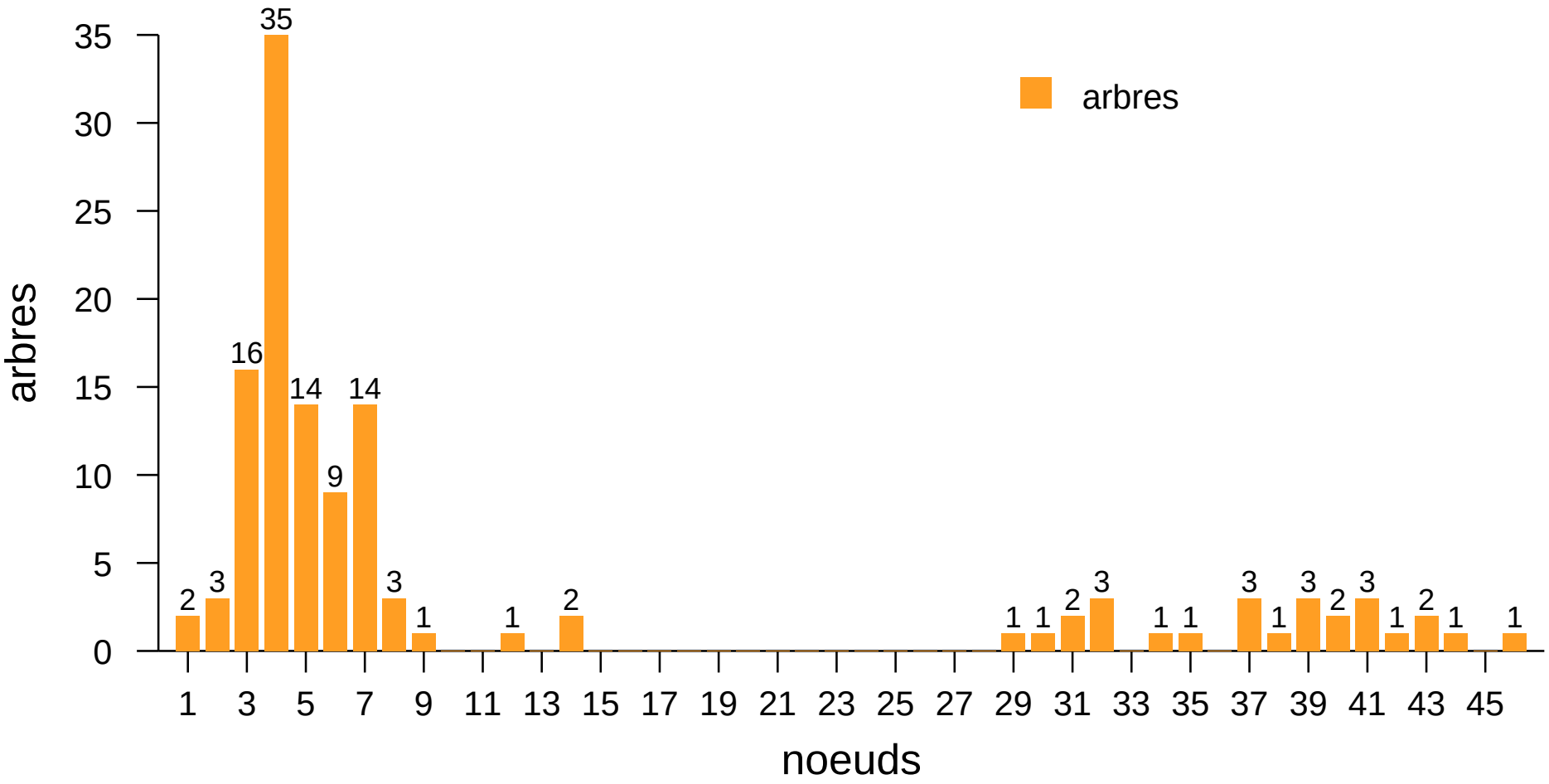
Canonique	Extr.	Actif	Passif	Quest.	Rel.	Clivées	Coord	Adv	Adj
7	19	19	6	4	4	11	12	14	11

Distribution par phénomènes syntaxiques

Gardes	Disjonctions	Entrelacement	Étoiles de Kleene
820	92	26	13

Utilisation des opérateurs de factorisation

La grammaire en quelques tables (suite)



Distribution par nombre de noeuds

Arbre #111

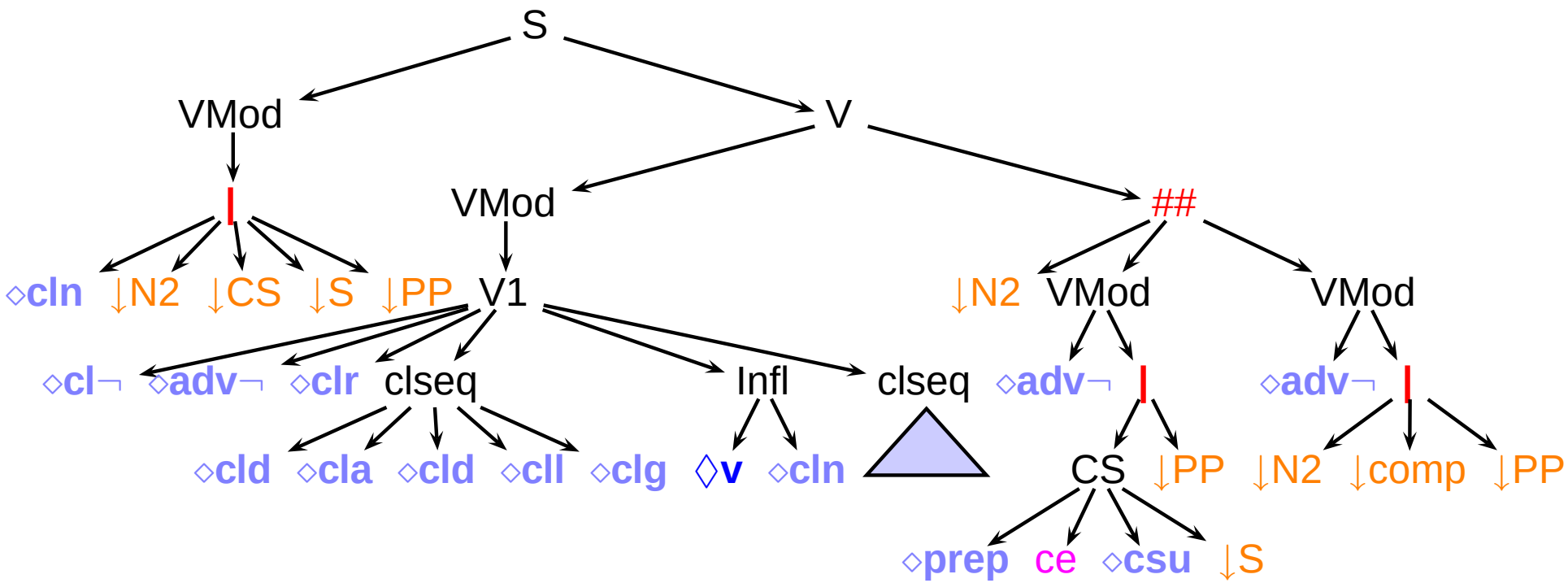
Arbre #111 : pour la construction canonique du verbe

- croisement de 25 classes terminales ;
- 43 nœuds plus 3 alternatives et 1 entrelacement ;
- 35 gardes ;
- +2000 lignes XML

Arbre #111

Arbre #111 : pour la construction canonique du verbe

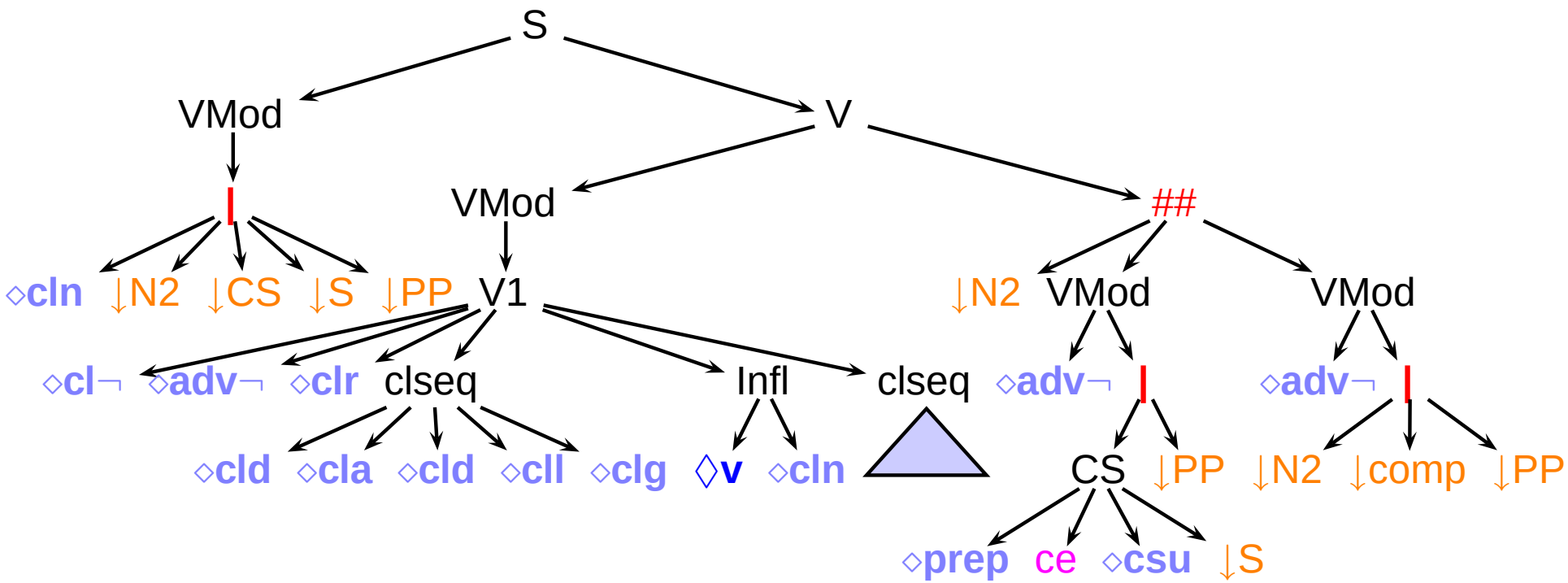
- croisement de 25 classes terminales ;
- 43 nœuds plus 3 alternatives et 1 entrelacement ;
- 35 gardes ;
- +2000 lignes XML



Arbre #111

Arbre #111 : pour la construction canonique du verbe

- croisement de 25 classes terminales ;
- 43 nœuds plus 3 alternatives et 1 entrelacement ;
- 35 gardes ;
- +2000 lignes XML

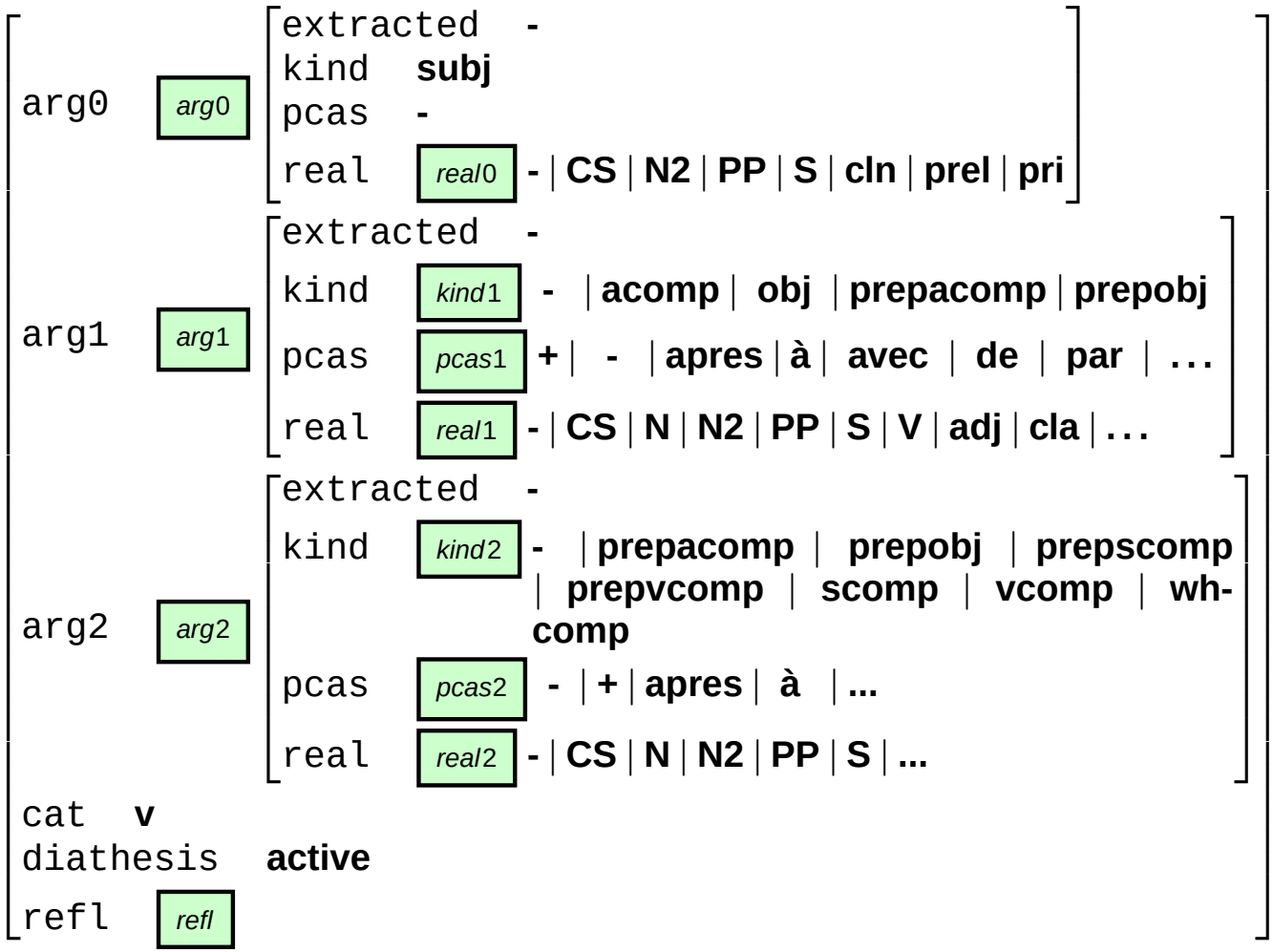


Taux de compaction : difficile à évaluer (10 000 à 100 000), mais + réaliste entre 100 et 1000

Lien lexique / Grammaire : ancrage par *hypertag*

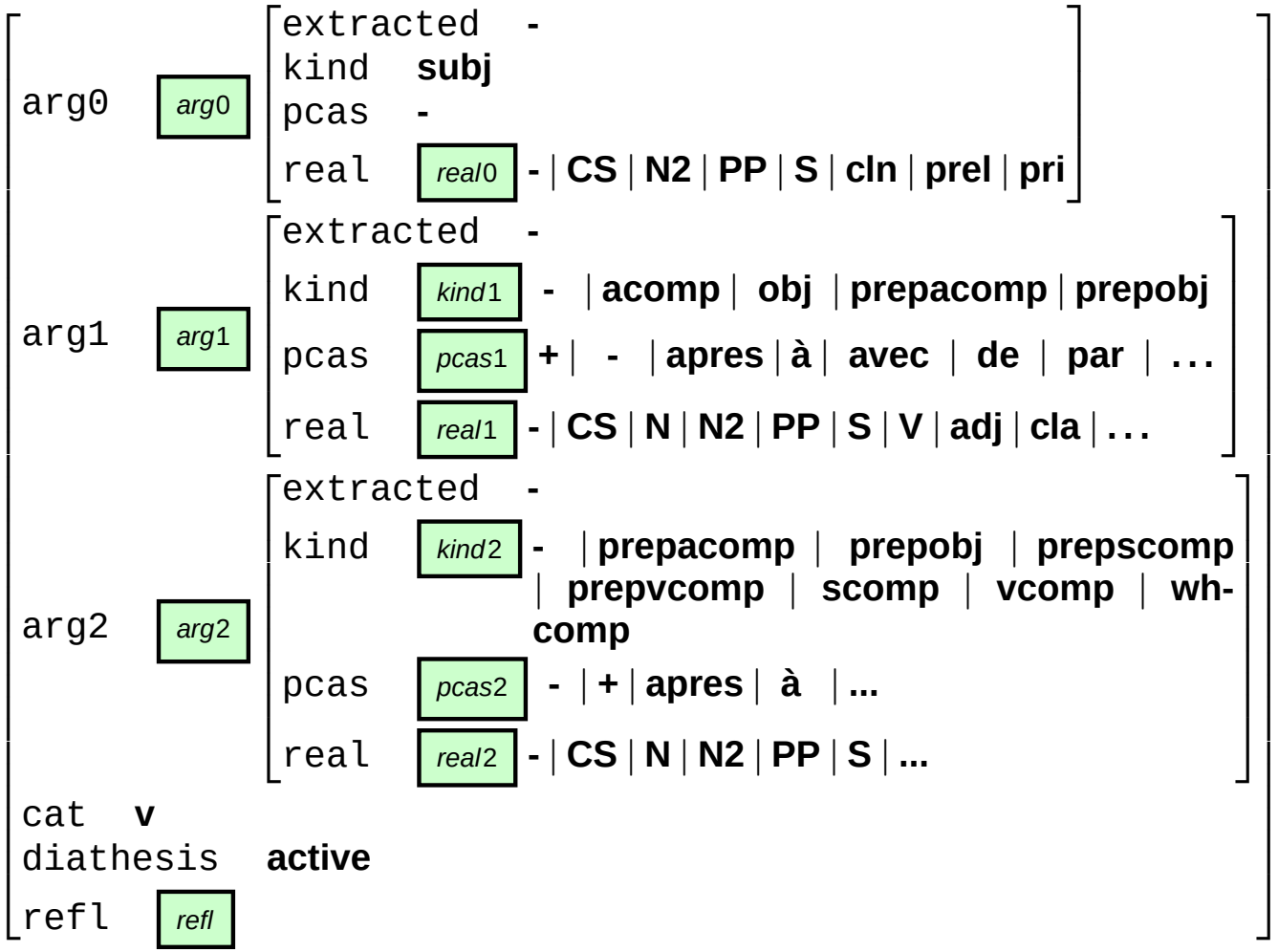
Grammaire

hypertag #111

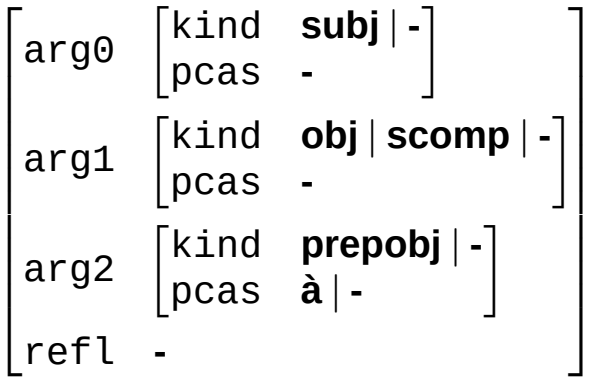


Lien lexique / Grammaire : ancrage par *hypertag*

Grammaire
hypertag #111



Lexique
hypertag «promettre»



Lien lexique / Grammaire : ancrage par *hypertag*

Grammaire
hypertag #111

```
arg0  arg0 = [ extracted - ; kind subj ; pcas - ; real real0 - | CS | N2 | PP | S | cln | prel | pri ]

arg1  arg1 = [ extracted - ; kind kind1 - | acomp | obj | prepacomp | prepobj ;
               pcas pcas1 + | - | apres | à | avec | de | par | ... ;
               real real1 - | CS | N | N2 | PP | S | V | adj | cla | ... ]

arg2  arg2 = [ extracted - ; kind kind2 - | prepacomp | prepobj | prepscomp
               | prepvcomp | scomp | vcomp | wh-comp ;
               pcas pcas2 - | + | apres | à | ... ;
               real real2 - | CS | N | N2 | PP | S | ... ]

cat  v
diathesis  active
refl  refl -
```

Lexique
hypertag «promettre»

```
arg0 [ kind subj | - ; pcas - ]
arg1 [ kind obj | scomp | - ; pcas - ]
arg2 [ kind prepobj | - ; pcas à | - ]
refl -
```

Les variables propagent les valeurs des hypertags aux noeuds et gardes

Environnement de développement

Principes :

- Développement de MG $\sim$ développement de programmes (objets)
- Utilisation de formats intermédaires XML $\rightsquigarrow$ vues multiples (XSL)

Édition et visualisation des MG : MGTOOLS

- Mode Emacs avec coloration syntaxique
- Interaction avec une visualisation graphique de la hiérarchie des classes
- Conversion format XML

Environnement de développement (suite)

Visualisation de la grammaire

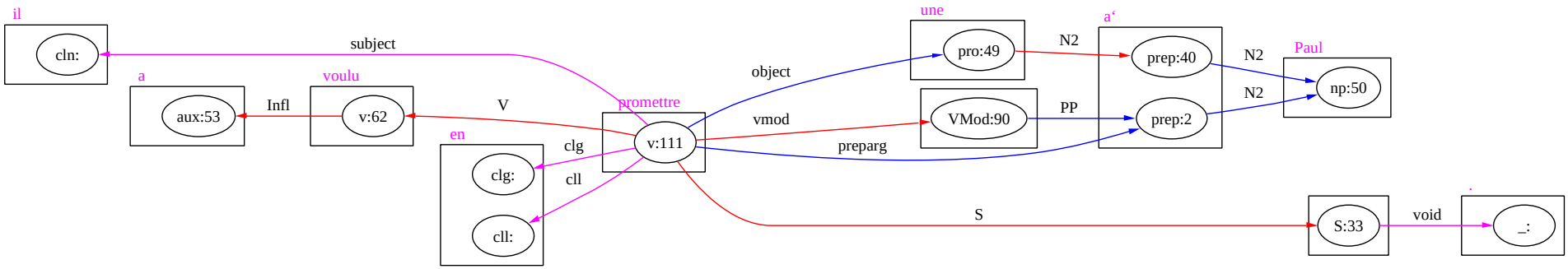
- conversion format TAGML (XML) vers vue HTML
- script CGI de recherche (par id et par mots clés)
  `http://atoll.inria.fr/perl/frmg/tree.pl`

Utilisation de la grammaire

- Serveur de parseur + visualisation des analyses sous différents formats (forme XML $\Rightarrow$ formes «grammaire», dérivations et dépendances)
  `http://atoll.inria.fr/parserdemo`
- Possibilité de visualisation des arbres utilisés dans une analyse
- Utilisation de jeux de tests avec calculs de statistiques (temps, couverture, taux d'ambiguité, ...) + modif round précédent
- Calcul de taux de «parsabilité» par mot $\Rightarrow$ amélioraton lexique

Sorties d'analyse syntaxique (format dépendances)

il a voulu en promettre une à Paul.



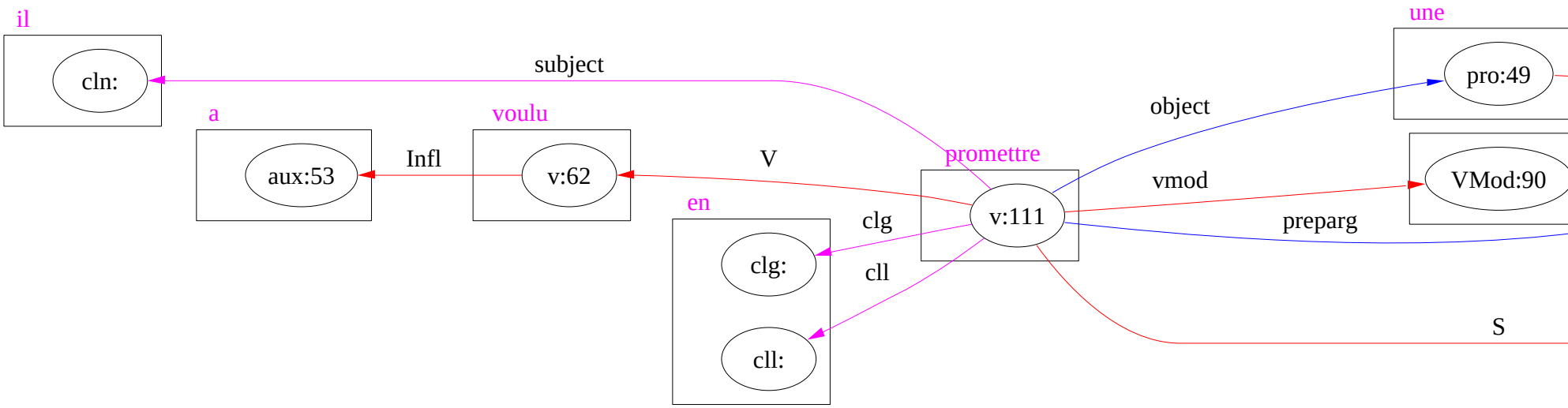
Definition (Ambiguité moyenne par mot α)

$$\alpha = \frac{\text{nombre d'arcs}}{\text{nombre mots} n} - 1$$

- Pour une phrase non ambiguë : $\alpha = 0$
- Pour une phrase ambiguë : nombre d'analyse $\sim (1 + \alpha)^n$

Sorties d'analyse syntaxique (format dépendances)

il a voulu en promettre une à



Definition (Ambiguité moyenne par mot α)

$$\alpha = \frac{\text{nombre d'arcs}}{\text{nombre de mots}} - 1$$

Couverture et performances

Utilisation de l'analyseur TAG/TIG pour des analyses complètes
note : l'analyseur peut aussi retourner des analyses partielles (mode robuste)

Corpus	#phrases	% couv.	temps (s)		ambiguïté
			moyen	médian	
EUROTRA	334	95.80	1.81	1.27	0.7
TSNLP	1661	93.38	0.72	0.56	0.4
MD10x20	5000	63.18	2.85	1.80	0.8
EASY	34438	42.45	5.55	1.61	0.6

Résultats (avec un *timeout* de 100s)

Couverture et performances

Utilisation de l'analyseur TAG/TIG pour des analyses complètes
note : l'analyseur peut aussi retourner des analyses partielles (mode robuste)

Corpus	#phrases	% couv.	temps (s)		ambiguïté
			moyen	médian	
EUROTRA / OLD	334	95.80 / 89.22	1.81 / 0.70	1.27 / 0.54	0.7 / 0.3
TSNLP / OLD	1661	93.38 / 86.15	0.72 / 0.43	0.56 / 0.33	0.4 / 0.2
MD10x20 / OLD	5000	63.18 / 43.06	2.85 / 1.97	1.80 / 1.30	0.8 / 0.5
EASY	34438	42.45 / -	5.55 / -	1.61 / -	0.6 / -

Résultats (avec un *timeout* de 100s)

Modif mal contrôlées de dernière minute pour améliorer la couverture

- $\rightsquigarrow$ dégradation temps et ambiguïté par rapport à une ancienne version OLD
- $\Rightarrow$ importance d'un suivi constant et régulier

Manque information sur la précision :

- Attente résultats EASy
- Attente corpus annoté en syntaxe

Conclusion

- MG + env. développement + factorisation $\Rightarrow$
  - grammaire compacte
  - temps de développement rapide (quelques mois)

- DyALog + factorisation : puissance et généricité

- Mais le dev. d'une MG reste une tâche complexe
  - Besoin important d'expertise linguistique
  - Besoin d'un lexique syntaxique riche
  - Discipline rigoureuse de tests
  - Contraintes plus expressives et plus puissantes
    exclusion, topologique, rang, ...
  - Formalisme cible plus expressif que les TAGs, comme les MC-TAGs
    $\Rightarrow$ réduire la distance entre MG et Formalisme Cible
  - Besoin de probabilités/heuristiques pour lever les ambiguïtés

- Outils et ressources librement disponibles :
  DyALog, FRMG, MGCOMP, MGTOOLS + Lefff + ...
  `http://atoll.inria.fr` rubrique Catalogue

Estimer le taux de compaction des arbres factorisés

Pour l'arbre #111

- subjreal = 5 : choix de réalisations pour un sujet pré-verbal : `cln N2 CS S PP`
- subjpos : choix entre sujet « absent », « pré-verbal », « pré-verbal et cl post-verbal », « absent et cl post-verbal », « post-verbal » ;
- argspos = 3 : entrelacements entre 2 args post-verbaux + un sujet post-verbal, tous optionnels ;
- clseqpos = *X* : choix entre 2 séquences de clitiques pré ou post verbaux optionnels : `cld`, `cla`, `cld`, `cll`, `clg` ;
- clopt = 2 : clitiques pré-verbaux optionnels : `clneg` (« ne »), `clr` (réflexif) ;
- neg = 3 : positions de négation (advneg « que » sur les arguments verbaux ou advneg pré-verbal pour les infinitives) ;
- arg1real = 3 : réalisations possibles pour arg1 : `N2 comp PP` (nominal, adjectival, infinitif, complétif, . . .) ;
- arg2real = 2 : réalisations possibles (au moins) pour arg2 (CS,PP) ;

Meta Grammaire du Français

- Sous-catégorisation des verbes : sujet, attribut, `object`, `prep-object`, `vcomp` et `scomp`, `prep-vcomp`, `wh-comp` `prep-scomp`.
  au plus 3 arg. verbaux (sujet inclus)
- Constructions auxiliaires, verbes à controle (sujet)
- Diverses réalisations (np, clitique, infinitive, complétive) et position (pre, post, post-clitique) du sujet
- extraction arguments et modifiers (questions, relatives, et clivées)
- construction passive et active
- coordination partielle (sans ellipse)
- modifieurs du verbe à diverses positions (participiales, PP, adv, . . .),

EASY – Détail

corpus	#sent	% coverage
general elda	806	50.25%
general lemonde	2659	38.70%
general mlcc	2435	40.78%
general senat	260	38.08%
litteraire 1	1102	38.93%
litteraire 2	1577	31.07%
litteraire 3	2036	39.98%
litteraire 4	3245	42.10%
mail 1	873	32.65%
mail 10	192	45.31%
mail 2	520	30.96%
mail 3	621	24.32%
mail 4	1230	28.78%
mail 5	556	31.65%
mail 6	695	24.46%
mail 7	769	35.37%
mail 8	1609	36.98%
mail 9	627	43.70%
medical 1	859	44.59%
medical 2	51	33.33%
medical 3	445	42.02%
medical 4	305	31.48%
medical 5	283	45.94%
medical 6	282	58.87%
oral delic 1	99	51.52%
oral delic 2	83	43.37%
oral delic 3	90	51.11%
oral delic 4	51	33.33%
oral delic 5	46	32.61%
oral delic 6	47	38.30%
oral delic 7	**37**	**16.22%**
oral delic 8	23	21.74%
oral delic 9	41	26.83%
oral elda 1	821	45.68%
oral elda 2	813	47.85%
oral elda 3	741	41.16%
oral elda 4	917	45.26%
oral elda 5	713	45.44%
oral elda 6	986	50.20%
oral elda 7	763	40.50%
oral elda 8	621	38.97%
questions amaryllis	1611	60.15%
questions trec	**1898**	**77.03%**