Formalisation d'expressions polylexicales au sein d'une méta-grammaire

Yoann Dupont & Éric de la Clergerie
<Prenom.Nom@inria.fr>

Journées PARSEME-FR
Blois, 13-14 Juin 2019

Motivations

Sujet de post-doctorat de Yoann Dupont

- faciliter la description de la structure syntaxique interne des MWE
  pb : diversité des structures, variabilité, contraintes (lexicales, accord, . . .)
- s'appuyer sur des descriptions modulaires (Méta-grammaire) et sur une grammaire existante (FRMG)
- obtenir des représentations opérationnelles efficaces au moment de l'analyse syntaxique
- (*optionnel*) dans la mesure du possible, induire les descriptions des MWE à partir de corpus

FRMG en bref

Depuis 2004, FRMG, une grammaire TAG à large couverture pour le français

- générée à partir d'une meta-grammaire
  - avec des arbres élementaires assemblés à partir de contraintes
  - les contraintes venant de classes, qui héritent d'ancêtres et qui se combinent
  - $\rightsquigarrow$ compacte grammaire, grâce à la factorisation d'arbres (383 arbres)
- en entrée, un treillis de mots (DAG) produit par SxPipe
  $\rightsquigarrow$ conserve **ambiguïtés** lexicales et de segmentation
- en sortie **toutes** les analyses (complètes ou partielles) comme forêts partagées
  (dérivations puis dépendances)
- Phase de désambuisation vers un arbre de dépendances, avec
  - règles et poids manuellement définis
  - mais aussi des poids appris sur corpus (FTB)
    et affinités d'attachement apprises sur grands corpus (hyp. distributionnelle)
- $\sim$ 97% couverture complète sur FTB, et $\sim$ 88% LAS sur FTB test
- à tester sur le wiki FRMG : `http://alpage.inria.fr/frmgwiki`

Méta-Grammaires (version SMG)

Definition (Méta-Grammaire)

Description modulaire par classes regroupant des contraintes, avec héritage

Méta-Grammaires (version SMG)

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 (=>)

Exemple d'héritage (pour les adverbes)

```
class categories { %% base for anchored tree
    node Anchor : [type:anchor];
    desc.@htcat = node(Anchor).cat;
    node(Anchor).id = node(Anchor).cat;
    desc([ht:@ht_fs]); }

class adv { %% Adverbs
    <: categories;
    node Adv : [cat: adv, bot: [degree: -]]; Adv=Anchor;
    desc.ht = value([...]); }

class adv_modifier {   %% Adverbs as modifier
    <: adv;
    - shallow_auxiliary;
    Root >> Incise;
    Incise >> Adv;
    node Incise : [cat:incise,id:incise, type:std];
    node(Root).bot = node(Foot).top;  }
```

Exemple pour adv (suite)

```
class adv_v { %% Adverbs on verbs: may only occur after verb
    <: adv_modifier;
    node Root : [cat: v];  Foot < Anchor;
    node(Adv).bot.adv_kind = value(~très); }
```

```
class adv_before_v { %% Adverbs on non tensed verbs
  <: adv_modifier;
  node Root : [cat: v];  Anchor < Foot;
  node(Adv).bot.adv_kind = value(~très);
  node(Foot).top.mode = value(infinitive|participle|gerundive);
  desc.@kind0 = value(-); }
```

```
class adv_adv { %% Adverbs on adverbs: très très petit
    <: adv_modifier;
    node Root : [cat: adv];  Adv < Foot;
    node(Incise).bot.incise_kind = value(dash|par);
    node(Adv).bot.adv_kind = value(très|intensive); }
```

```
class adv_adj { %%  Adverbs on adjectives: très petit
    <: adv_modifier;
    node Root : [cat: adj]; Adv < Foot;
    node(Incise).bot.incise_kind = value(dash|par); }
```

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)

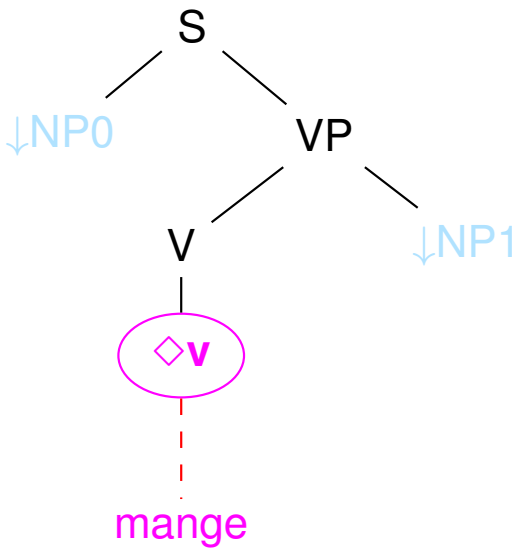
Étape 2 : Classes neutres

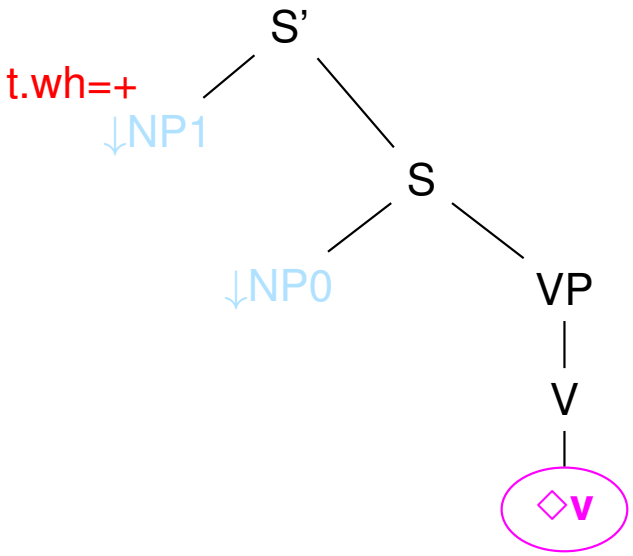
- Croisement des classes terminales pour neutraliser ressources & besoins
  - $C_1[-R \cup \mathcal{K}_1] x C_2[+R \cup \mathcal{K}_2] = (C_1 x C_2)[=R \cup \mathcal{K}_1 \cup \mathcal{K}_2]$
  - (Espace de nom) $\Rightarrow$ import classe productrice avec renommage
    $C_1[-N::R \cup \mathcal{K}_1]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

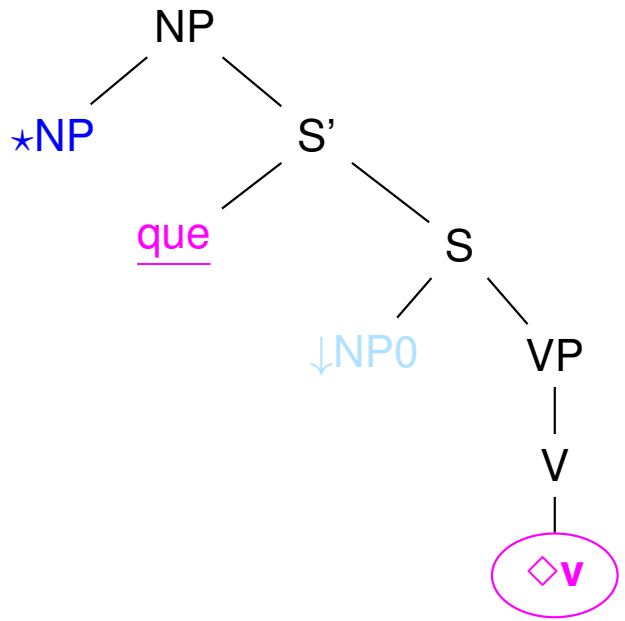
Étape 3 : Arbres TAG/TIG

Utilisation des contraintes des classes neutres pour construire les arbres

Arbres TAG

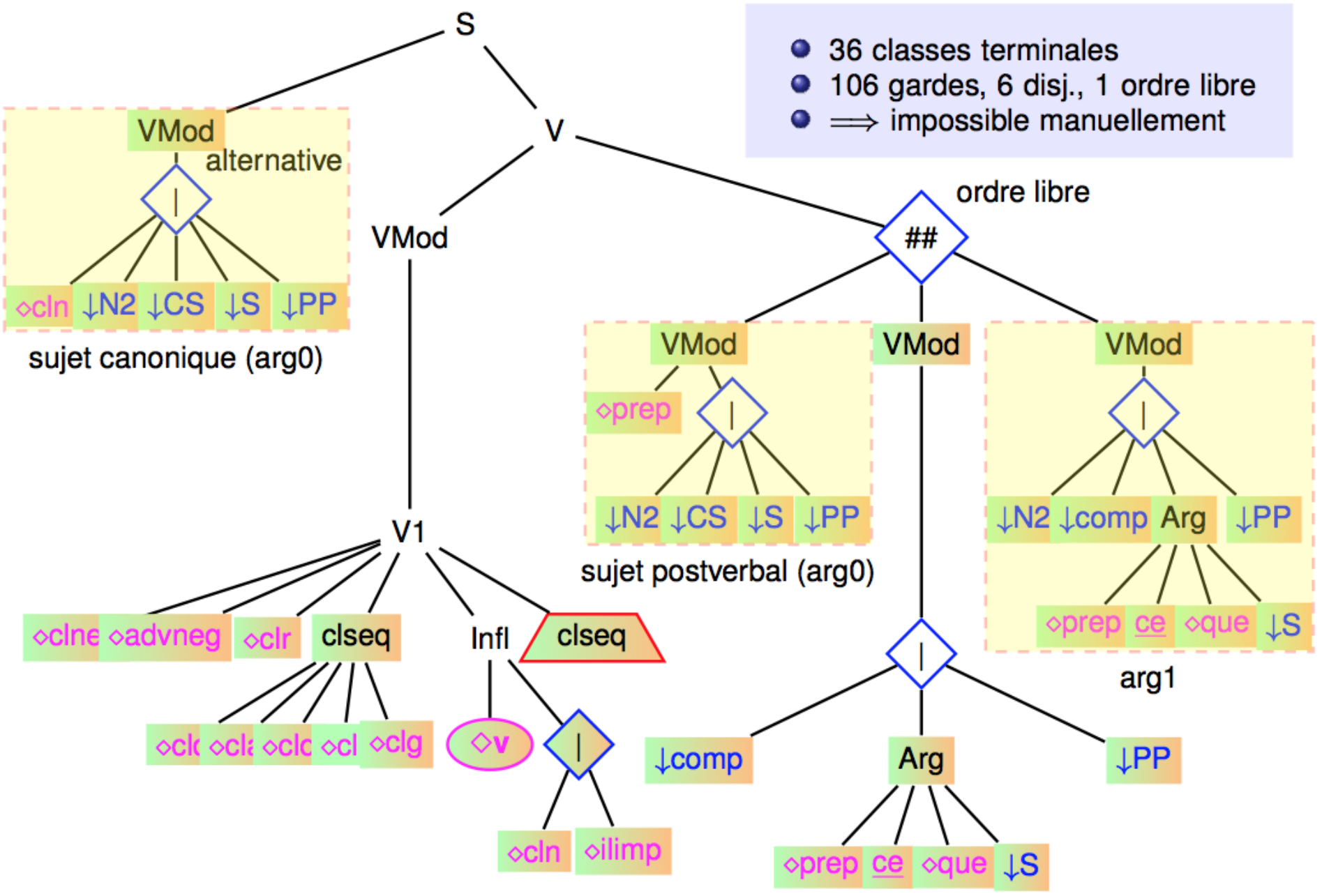






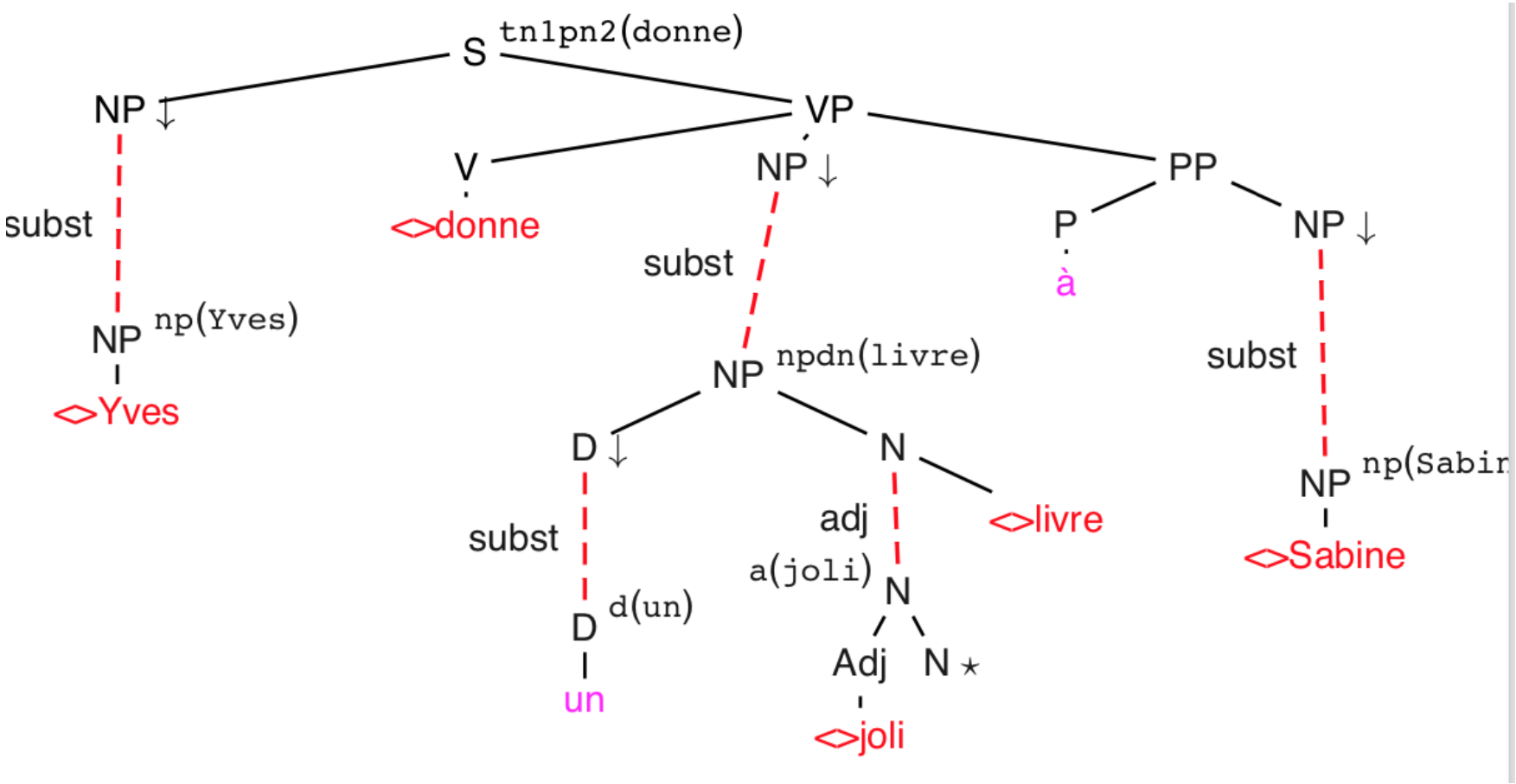
Les arbres offrent un domaine de localité étendu et sont normalement associé à un prédicat (sémantique) et ses arguments

Arbre #198 : verbe canonique (simplifié)



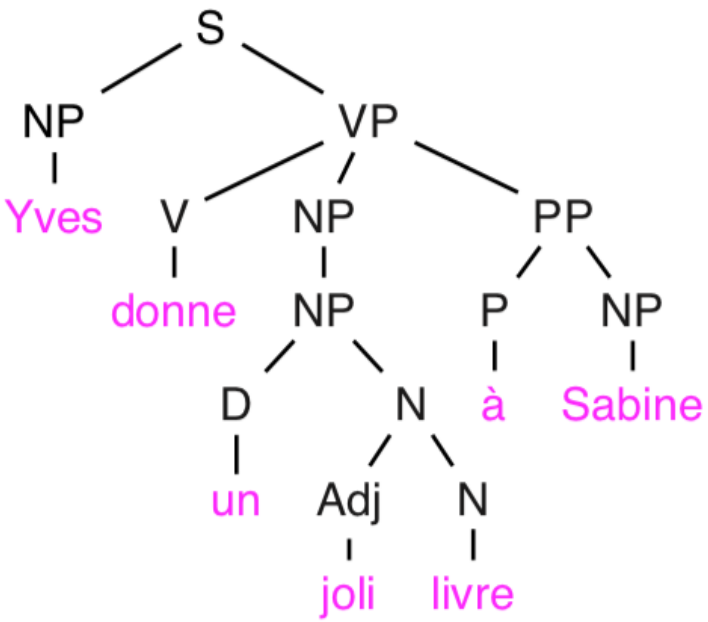
Opérations TAG

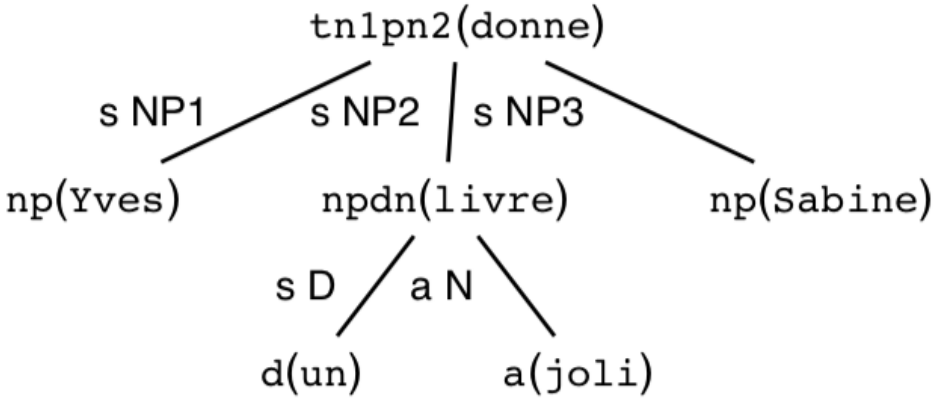
Les arbres élémentaires TAG sont combinés grâce aux 2 opérations de substitution et d'adjonction



Dérivations TAG

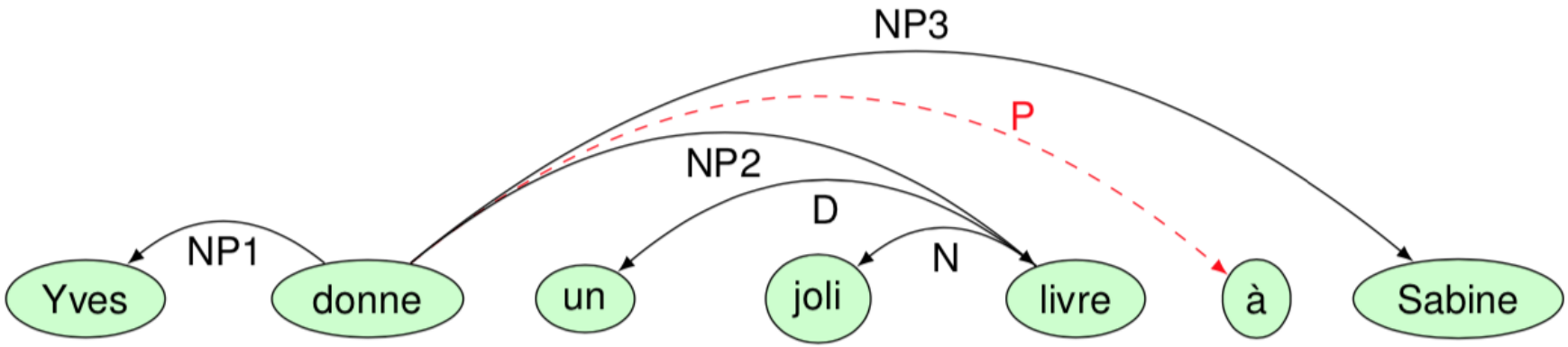
La trace des opérations TAG se traduit par un arbre de dérivation





Des dérivations vers les dépendances

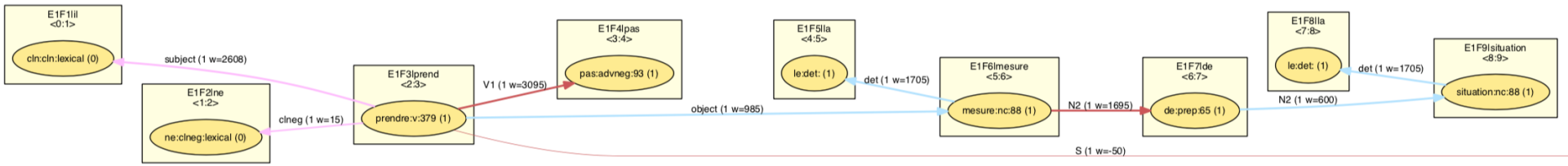
Conversion facile des dérivations vers des dépendances
(pour les TAG lexicalisées)



pour chaque étape de dérivation (subst, adj)
qui applique un arbre τ_1 sur un noeud N d'un arbre τ_2,
$\Rightarrow$ on ajoute un arc $\text{anchor}(\tau_1) \longrightarrow_N \text{anchor}(\tau_2)$

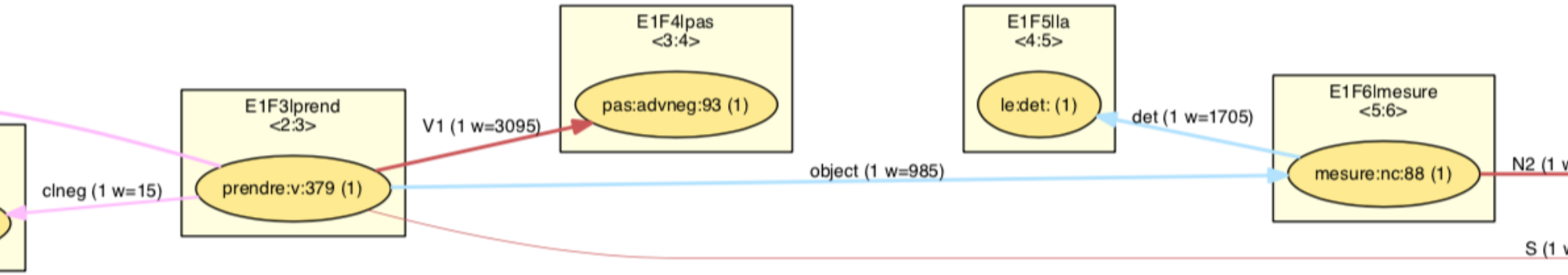
Sortie DepXML (FRMG)

Arbre de dépendances donnant accès aux arbres ancrés par les mots et les labels des arcs en relation avec des noeuds des arbres



Egalement accès aux span des arbres, aux traits des mots, aux cadres de sous-catégorisation, . . .

Sortie DepXML (FRMG)



ccès aux span des arbres, aux traits des m
risation, . . .

Arbres associés aux classes

Un arbre est la résultante du croisement d'un ensemble de classes

```
381
V1VMod:agreement arg1:caimp:agreement clsubj:agreement clsubj_alt:a
clsubj_il:agreement clsubj_ilimp:agreement arg1:ilimp:agreement
arg1:imp_subj_alt:agreement prel_arg:agreement
ante:clitic_sequence post:clitic_sequence clitics
arg0:collect_real_arg arg2:collect_real_arg arg1:collect_real_subje
arg0:real_group_comp arg2:real_group_comp
arg1:PP:true_subject arg1:cl:true_subject arg1:noun:true_subject
arg1:post_PP:true_subject arg1:post_noun:true_subject
arg1:post_s:true_subject arg1:post_v:true_subject
arg1:s:true_subject arg1:v:true_subject
v_with_subcat Infl:verb_agreement
V:verb_agreement v:verb_agreement
V1:verb_agreement_ancestor
arg0:verb_argument_other arg2:verb_argument_other
arg1:verb_argument_subject
verb_categorization_passive
verb_extraction_relative
```

MWE et MG : deux approches possibles

1. décrire des arbres étendus pour les MWE
   en s'appuyant sur les classes de base existantes
   - $\oplus$ respect convention arbre $\equiv$ domaine sémantique
   - $\ominus$ potentiellement beaucoup de nouveaux arbres
   - $\ominus$ ambiguïté accrue des analyses (multi-analyse)
   - $\rightsquigarrow \ominus$ coût algorithmique important

2. décrire une MWE comme un ensemble de liens entre un paquet d'arbres existants plus contraintes
   - $\oplus$ pas de nouveaux arbres (ou très peu)
   - $\oplus$ pas de modifications de l'algorithme de parsing (ou légères)
   - identification des MWE en post-parsing sur la forêt de dérivation/dépendances
   - $\oplus$ proximité conceptuelle avec la notion de Multi-Component TAG
     ($\ominus$ pas de miracle sur la complexité pire des cas !)

Approche 2 retenue !

Notion de classe MWE

Extensions du formalisme SMG pour :

- dénoter des paquets d'arbres $\tau_1, \ldots \tau_n$ héritant de certaines classes
- dénoter des liens TAG entre entre 2 arbres τ_h et τ_d (par substitution ou adjonction) sur un noeud N de τ_h
- nommer un noeud N d'un arbre τ (d'un paquet)
- (optionnel) autoriser de la factorisation au niveau des paquets d'arbres
- (optionnel) autoriser des gardes sur la sélection d'un arbre en cas de disjonction

$\rightsquigarrow$ réécriture par Yoann du parseur SMG pour faciliter l'ajout des nouvelles notations

Exemples

- Verbe + objet nominal

```
class VerbNoun {
  % tree variable T_v bound to a tree inherited from class verb
  tree T_v <: verb;
  tree T_obj <: cnoun_leaf;

  subst(T_v/arg1::N2) = T_obj;
  T_v/desc.ht.arg1.function = value(obj);
  T_v/node(arg1::N2).id = value(object)
}
```

- Propriétés supplémentaires sur l'objet

```
class VerbNounDef {
  %% serrer les dents
  <: VerbNoun
  T_obj/node(det).top.def = value(+);
}
class VerbNounPossSuj {
  %% faire ses preuves
  <: VerbNoun
  T_obj/node(det).top.poss: value(+);
  T_obj/node(det).top.numberposs = T_v/node(anchor).number
  T_obj/node(det).top.persposs = T_v/node(anchor).person
}
```

Que décrire ?

- Structure interne (arbres et liens)
- Flexibilité interne (ordre, transformation, optionalité, remplacement)
  se faire une idée fausse / **se faire une fausse idée** ; **des coups furent échangés**
- Sous-catégorisation
  prendre la mesure de X
- Bloquer/autoriser des points de modification
  prendre (rarement) la (pleine) mesure
- Vue externe (changement de catégorie)
  il accepte **n'importe quel** *travail* ; *il a mangé* **je ne sais quel** *fruit qui l'a rendu malade*

Flexibilité interne (exemple)

En fait, les classes les plus générales offrent le plus de flexibilité interne
L'ajout de contraintes bloque la flexibilité

```
class VerbNounCanonical {
  <: VerbNoun;
  tree T_v <: verb_canonical;    % no extraction allowed
  tree T_v <: verb_categorization_active % only active mode
}

class VerbNounCanonicalAlt {
  <: VerbNoun;
  T_v/desc.ht.arg1.extracted = value(-);
  T_v/desc.ht.diathesis = value(active);
}

class NounAdjAnte {
  <: NounAdj;
  T_adj/anchor < T_noun/anchor;
}
```

Points de modification (exemple)

Idem que flexibilité : par défaut, modifieurs autorisés.
ajout de contraintes pour bloquer des modifications

```
class VerbNounNoModif {
  <: VerbNoun;
  T_obj/node(N).adj = value(-);          % no adjective on object
  T_obj/node(N2).adj = value(-);         % no post-noun modifiers (
    PP, relatives, ...)
}
```

Classe vs lexique

Le formalisme SMG permet de définir des classes MWE très précises et contraintes

Mais pas raisonnable d'inclure dans une classe MWE l'ensemble des contraintes (lexicales, sous-catégorisation, . . .)
$\rightsquigarrow$ trop de classes !

Équilibre à trouver entre :

- un ensemble restreint de classes MWE
  couvrant les grands types d'expressions
- des contraintes à exprimer (dans LEFFF) au niveau d'entrées lexicalese
  nécessite des extensions des contraintes ($f = v$) de LEFFF

Lexique LEFFF (Sagot)

Les entrées fournissent des informations

- morpho-syntaxiques (sous forme de traits)
- et syntaxiques (traits + cadre de sous-catégorisation)

Entrée LEFFF pour **promets**

```
pred="promettre<Suj:(cln|scompl|sinf|sn),
                Obj:(cla|de-sinf|scompl|sn),
                Objà:(cld|à-sn)>",
ctrsubj = suj,
imp = -,
cat=v,
@P12s # tense = present, mode = indicative, person=1|2, number = sg
```

Ajout des contraintes de précédence et des variables d'arbres, de noeuds, ... de SMG + mention d'une classe MWE

```
faire son cinéma
mwe_class=VerbNounPossSuj,T_v/anchor.lemma=faire,T_obj/anchor.lex=cinéma|ciné
```

Compilation & Exploitation

La compilation des classes MWE ne donne pas des arbres mais plutôt une clause Prolog (DyALog)

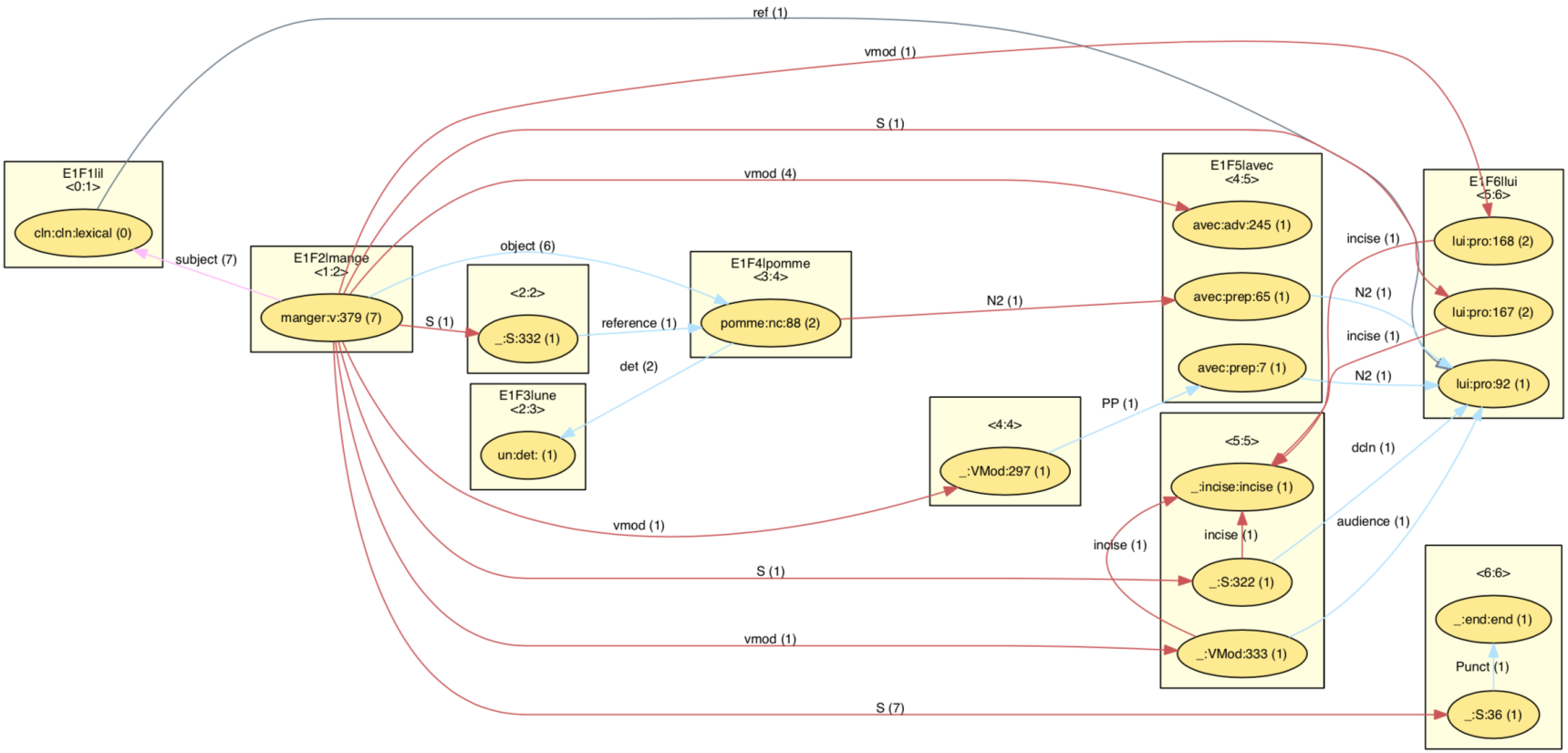
```
mwe_check_VerbNoun_1(V,N,V^N^ExtraConstraints) :-
    V::node{ tree => VTree, deriv => DId }, member(verb,VTree),
    E1::edge{ id => EId1,
              label => object,
              source => V,
              target => N,
              type => subst
    },
    N::node{ tree => NTree }, member(cnoun_leaf,NTree ),
    hypertag(DId,ht{ arg1 => arg{ function => obj }}),
    check_extra_constraints(ExtraConstraints)
    .
```

Les clauses sont

- activées par les entrées lexicales MWE potentielles dans la phrase
- vérifiées sur la forêt de dépendances
- + vérification des contraintes ExtraConstraints de l'entrée lexicale

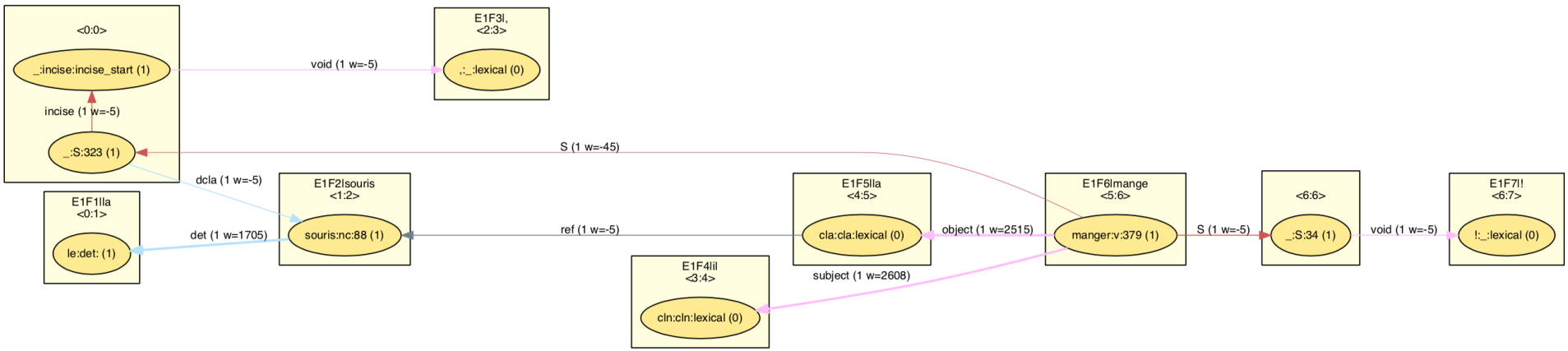
Forêt partagée (DepXML)

il mange une pomme avec lui



Modèle (similaire) des dépendances secondaires

Déjà parcours de la forêt partagée pour l'ajout de dépendances secondaires utilisées par exemple pour des verbes à contrôle ou les disloquées



Disloquée (MG)

```
class dislocated_on_s
{
    %% a dislocated Nominal Phrase (N2) related to some clitic and attached to S
    %% example: le livre , il l'a lu
    node N2: [cat: N2, type: subst, id: $id, top: [time: -, sat: +, wh: -]];
    - s_modifier; N2 = Modifier;
    node(N2).id = value(dcln|dcla|dcld|dclg|dcll);
    N2 +
        node(N2).dummy.nodeid = value(dcla),
        node(Foot).bot.xcla = node(N2).top {.@ngp},
        node(Foot).bot.xcla.person = value(~-),
        desc.secondary.dislocated.gov = value(cla)
        |
          ...
        ;
    desc.secondary.dislocated = value([ref: $id, label2: ref, axis: up]);
    ...
}
```

Disloquée (Grammaire)

```
tag_tree{ name => '323 dislocated_on_s_ante ...',
          family => '323 dislocated_on_s_ante ...',
          tree => auxtree id= 'Root'
                  at - 'S'(
                      id= incise
                      at ++ incise(
                          id= _id_1925::nodeid[dcla,dcld,dclg,dcll,dcln]
                          at 'N2',
                          ({'$tagop'(_id_1925,secondary(_id_1925,_gov_1939,ref,up))},
                           id= 'Foot'
                           and top= _bot_229
                           and bot= _bot_229
                           at * 'S')
                        )
        }.
```

- La traversée de l'arbre pendant le parsing enregistre dans la forêt de dérivations des informations sur un `secondary`
- lors de la conversion de la forêt de dérivation en forêt de dépendances, l'enregistrement est utilisé pour trouver les têtes et dépendants des dépendances secondaires

Exploitation sur corpus analysés (format DepXML)

Dans la lignée de l'exposé à venir de Yoann

- Extraction de motifs de dépendances (+ contraintes) à partir d'une liste de MWE, puis conversion motifs
  - pour induire des classes MWE de la MG
  - pour créer des entrées lexicales MWE liés aux classes

- Compilation des classes MWE vers des motifs d'extraction DPath

Cas délicats

Tout ne peut être géré en post-parsing

- constructions syntaxiques non standard
  oblige à ajouter de nouvelles classes (non MWE) $\rightsquigarrow$ nouveaux arbres
  mais ces arbres **doivent** être activés seulement en présence des MWE

- utilisation externe d'une MWE incompatible avec sa structure interne
  changement de catégorie : « S interne » vu comme Det
  il a mangé **je ne sais quel** *fruit qui l'a rendu malade*

Conclusion

Formalisation et implantation en cours

- Modifications minimales du formalisme SMG
- Réutilisation maximale de FRMG
  mais potentiellement quelques classes pour des constructions non standard
- Pas de surcoût pendant l'analyse pour la majorité des cas
  sauf contrôle (activation) des arbres non standard
- Coût d'identification et vérification des MWE en post-parsing sur forêts
  mais beaucoup de configurations déjà élaguées par le parsing
- Liens entre arbres (constructions), dérivations et dépendances
  $\rightsquigarrow$ induction de classes et production de motifs d'extraction (DPath)
- Examiner des rapprochements avec XMG,
  comme autre formalisme de méta-grammaire