Cours sur le traitement automatique des langues (IV)

Violaine Prince
Université de Montpellier
LIRMM-CNRS

Grammaire non restreinte

- Toute règle de production de la forme :
  - α -> β
- Équivalente à une machine de Türing

Algorithme de Markov (*)

- Règle élémentaire : $\alpha \rightarrow \beta$
- Application : Si $w = w_1 \alpha w_2$ et w_1 ne contient pas d'occurrence de α alors :

$$w \Rightarrow w' \text{ avec } w' = w_1 \beta w_2$$

- (*) Cours J. Chauché 2005

Algorithme

- Défini sur un vocabulaire fini V
- Ensemble fini et ordonné de règles
- Chaque règle a un type : ordinaire ou finale.
- Un algorithme définit une fonction sur un ensemble des mots définis sur un vocabulaire V' $\subseteq$ V. Si E est l'ensemble de définition de l'algorithme alors :

$$E \subseteq V'^* \subseteq V^*$$

Application de l'algorithme

- Application d'une seule règle à la fois.
- Choix strict de la règle à appliquer : la règle applicable la plus prioritaire.
- Application transitive des règles : une nouvelle règle s'appliquera sur le résultat de l'application d'une première règle.
- L'arrêt du processus : aucune règle applicable ou application d'une règle terminale.

Exemple: génération d'un mot miroir

- Soit V = { a,b,c }
- L'algorithme sera construit sur le vocabulaire V' = { a,b,c,α,β } ε (mot vide) .
- Notion de règle terminale x $\rightarrow$ .w
- Règles :

1. $\alpha\ \alpha \rightarrow \beta$
2. β a $\rightarrow$ aβ
3. β b $\rightarrow$ bβ
4. β c $\rightarrow$ cβ
5. $\beta\ \alpha \rightarrow \beta$
6. $\beta \rightarrow .\varepsilon$
7. αaa $\rightarrow$ aαa
8. αab $\rightarrow$ bαa
9. αac $\rightarrow$ cαa
10. αba $\rightarrow$ aαb
11. αbb $\rightarrow$ bαb
12. αbc $\rightarrow$ cαb
13. αca $\rightarrow$ aαc
14. αcb $\rightarrow$ bαc
15. αcc $\rightarrow$ cαc
16. $\varepsilon \rightarrow \alpha$

Application

- aabc => αaabc (16. ε → α)
- αaabc =>aαabc (7. αaa → aαa)
- aαabc =>abαac (8. αab → bαa)
- abαac =>abcαa (9. αac → cαa)
- abcαa =>αabcαa (16. ε → α)
- αabcαa =>bαacαa (8. αab → bαa)
- bαacαa =>bcαaαa (9. αac → cαa)
- bcαaαa =>αbcαaαa (16. ε → α)
- αbcαaαa =>cαbαaαa (12. αbc → cαb)
- cαbαaαa => αcαbαaαa (16. ε → α)
- αcαbαaαa =>ααcαbαaαa (16. ε → α)
- ααcαbαaαa =>βcαbαaαa (1. α α → β)

- βcαbαaαa =>cβαbαaαa (4. β c → cβ)
- cβαbαaαa => cβbαaαa (5. β α → β)
- cβbαaαa =>cbβαaαa (3. β b → bβ)
- cbβαaαa =>cbβaαa (5. β α → β)
- cbβaαa =>cbaβαa (2.β a → aβ)
- cbaβαa=>cbaβa (5. β α → β)
- cbaβa =>cbaaβ(2.β a → aβ)
- cbaaβ=>cbaa (6. β → .ε)

L'algorithme se termine grâce à la règle 6.

Propriétés

- On peut faire des calculs à l'aide d'un tel algorithme.
- Algorithmes fermés : Un algorithme est dit fermé s'il possède une règle de la forme
  - $\varepsilon \rightarrow .w$
- Il existe toujours un algorithme normé équivalent: On ajoute la règle en fin d'algorithme : $\varepsilon \rightarrow .\varepsilon$
- Ce qui est intéressant : la composition d'algorithmes.

Composition

- U sur l'alphabet V et U' sur l'alphabet V'
- $\alpha,\beta \notin V \cup V'$
- Soit V'' = $\{\sum_\sigma \mid \forall\sigma \in V'\}$
- Soit U'' l'algorithme U où chaque point est remplacé par α
- Soit U''' l'algorithme U' où chaque symbole σ est remplacé par $\sum_\sigma$ et chaque point par β et chaque règle de la forme $\varepsilon \rightarrow w$ par $\alpha \rightarrow \alpha w'$ où w' correspond à w et chaque règle de la forme $\varepsilon \rightarrow w$ par $\alpha \rightarrow \alpha\ \beta w'$

Composition

Exemple :

U:

1. $a \rightarrow .\varepsilon$
2. $\varepsilon \rightarrow ab$

Alors :

U' :

1. $ab \rightarrow .\varepsilon$
2. $\varepsilon \rightarrow ac$
3. $\varepsilon \rightarrow .ac$

U'':

1. $a \rightarrow \alpha$
2. $\varepsilon \rightarrow ab$

U''' :

1. $\sum_a \sum_b \rightarrow \beta$
2. $\alpha \rightarrow \alpha \sum_a \sum_c$
3. $\alpha \rightarrow \alpha \beta \sum_a \sum_c$

Composition

- L'algorithme suivant calcule U'(U(w)):
  1. $\sigma\alpha \rightarrow \alpha\sigma \ \forall \ \sigma \in V$
  2. $\alpha\sigma \rightarrow \alpha\Sigma_\sigma \ \forall \ \sigma \in V$
  3. $\Sigma_\sigma\sigma' \rightarrow \Sigma_\sigma\Sigma_{\sigma'} \ \forall \ \sigma,\sigma' \in V$
  4. $\Sigma_\sigma\beta \rightarrow \beta\Sigma_\sigma \ \forall \ \sigma \in V$
  5. $\beta\Sigma_\sigma \rightarrow \beta\sigma \ \forall \ \sigma \in V$
  6. $\sigma\Sigma_{\sigma'} \rightarrow \sigma\sigma' \ \forall \ \sigma,\sigma' \in V$
  7. $\sigma\beta \rightarrow .\varepsilon$

  U'''

  U''

Approche de la récurrence : juxtaposition

- Soit U et U’ deux algorithmes sur le même vocabulaire V
- Il existe un algorithme U’’ qui produit la juxtaposition des algorithmes U et U’ :

$$U''(w) = U(w)U'(w)$$

Même principe que pour la composition :

- W est doublé et le double transposé.
- U’ fonctionne sur un alphabet transposé

Réccurence

- Si U et U’ sont les deux fonctions définissant la récurrence :
- Les paramètres sont doublés jusqu’à ce que le paramètre de réccurence soit nul.
- On applique alors U sur la dernière valeur
- On applique ensuite U’ sur les valeurs résultantes successives

Application à la génération de structure

- Soit le langage $a^n b^n$ et sa structure parenthèsée avec le symbole S.
- Construction valable que sur ce langage :

1. $\alpha a \rightarrow S(a\alpha$ aabb => αaabb =>
2. $\alpha b \rightarrow b)\alpha$ S(aαabb => S(aS(aαbb
3. $\alpha \rightarrow .\varepsilon$ => S(aS(ab)αb =>
4. $\varepsilon \rightarrow \alpha$ S(aS(ab)b)α => S(aS(ab)b)

Remarques

- L’algorithme précédent donne la structure correcte sur le langage choisi.
- Il donne également une structure correcte sur le langage sur {a, b} formé des mots tel que tout préfixe a un nombre supérieur ou égal de a par rapport au nombre de b.
- Dans les applications en LN il ne s’agit pas d’écrire des algorithmes qui vérifient si la phrase est correcte mais pour une phrase correcte de donner la bonne structure.

Exemple d ’un analyseur syntaxique théorique : SYGMART

- Analyse morphosyntaxique
- Moteur de réécriture sur des objets cibles : structures syntaxiques
- Extension des algorithmes de Markov
- Puissance de la machine de Turing

Eléments de base pour un analyseur

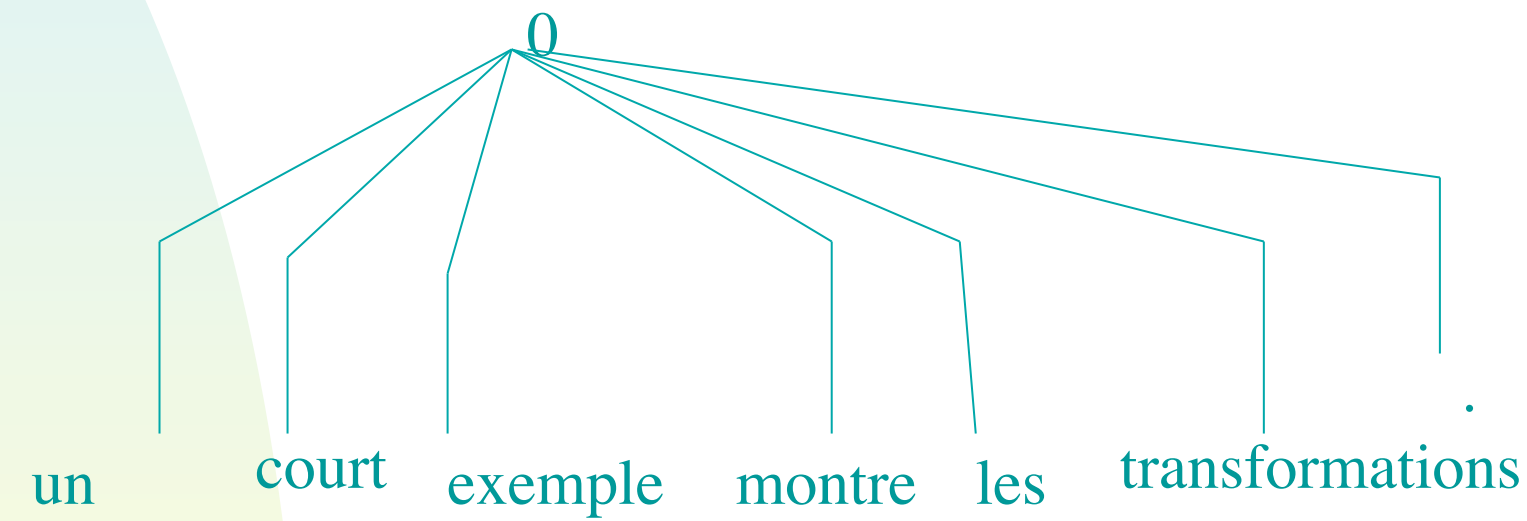
- Arborescences
  - Structures neutres.
  - Représentation :
    - chaîne de caractères sur le vocabulaire V = {(,)}
- Propriétés
  - transformation d ’arborescence : transformation de chaîne
  - Elément structuré : ensemble d ’arborescences.
  - Identification d ’arborescence : numéro (dimension)

- Etiquettes
  - Définies par une valeur
  - Variables typées : arithmétique (entière), flottant, chaîne, à valeur définie exclusive et non exclusive, potentielle (valeur définie mais non explicitée), référence (pointeur sur une autre étiquette).
- Fonction d ’étiquetage :
  - application d ’un noeud d ’une arborescence à une étiquette associée.
    - étiquetage morphologique : catégories grammaticales, nombre, genre

- VARIABLES UTILISEES
- DEFINIT Analyse
  - DECLARE variables_lexicales
    - CHAINE : frm.
    - POT: unite-lexicale.
  - FIN variables_lexicales.
  - DECLARE variables_grammaticales
    - EXC : categorie (PH,GN,GV,ART,ADJ,NM,PONCT)
    - NEX:nombre(sing,plur); genre(mas,fem).
  - FIN variables_grammaticales
- FIN analyse.

Exemples de règles « à la sygmart »

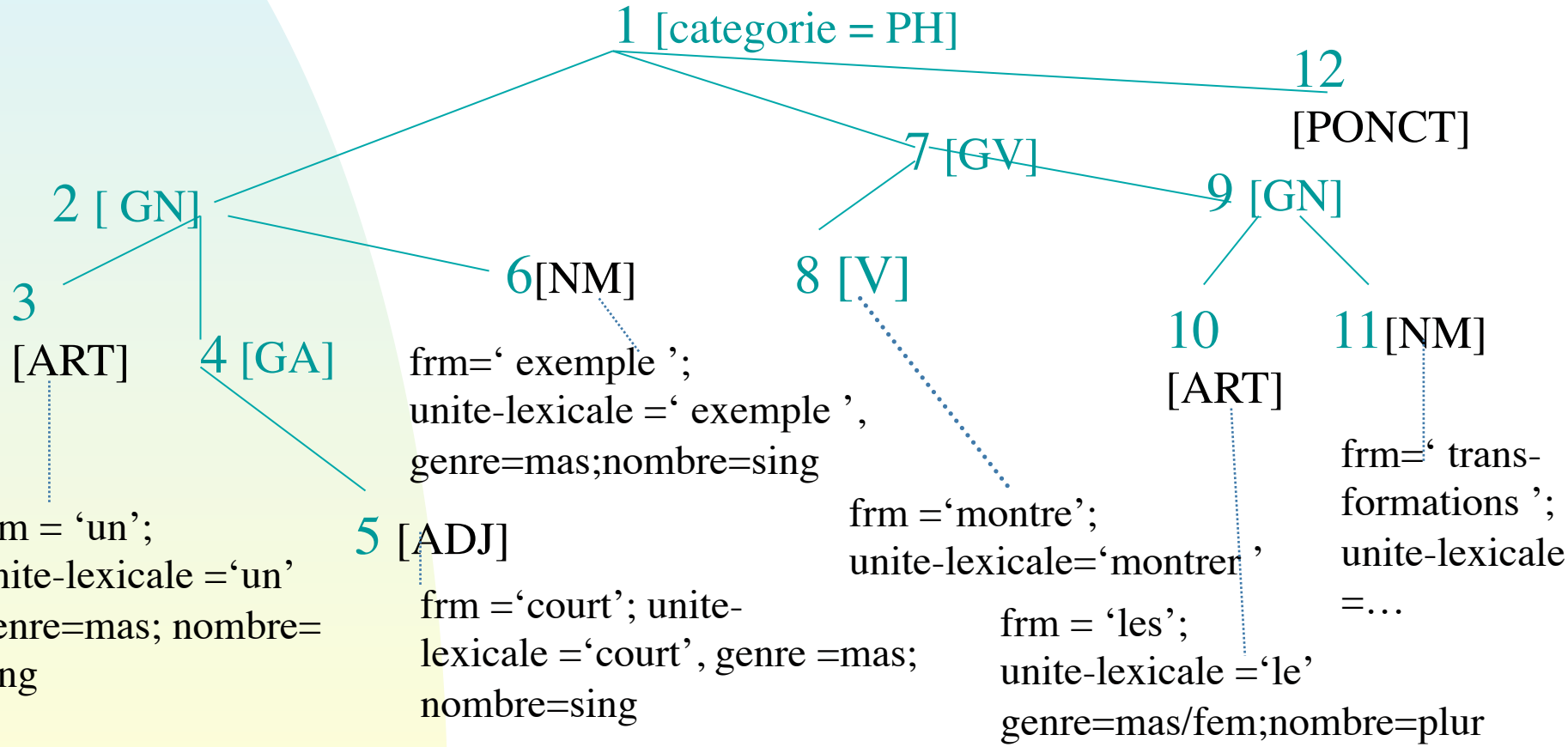
- La racine d ’un arbre d ’un texte à analyser est à 0.



noeuds de niveau 1

Ce à quoi on veut arriver

- « un court exemple montre les transformations.»

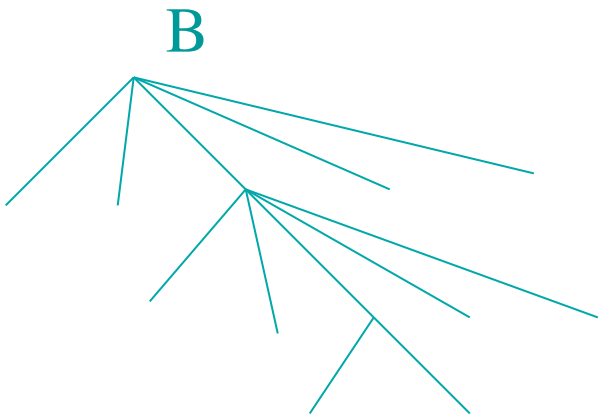


Transformations d ’élements structurés

- Schéma
  - ensemble d ’éléments structuré recherché dans l ’élément traité. Chaque élément a globalement la même structure, les variations portent sur l ’étiquetage, la présence (ou non) de certaines parties, l ’ordre entre les éléments.
- schéma d ’arborescence
    - soient deux arborescences A et B. A est une sous-arborescence de B s ’il existe une projection de A dans B.

A

B

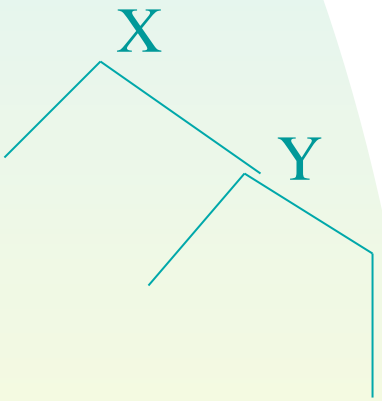


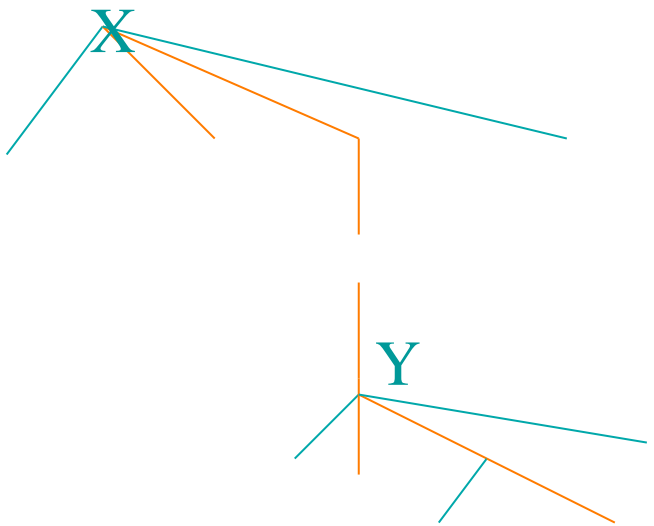
A: (() (() (()))) B: (() () (() () (() ()) () ()) () ())

A est une sous-arborescence de B. La décomposition correspond à un traitement d'un ensemble d 'infixes (un infixe est un mot contenu dans un autre).

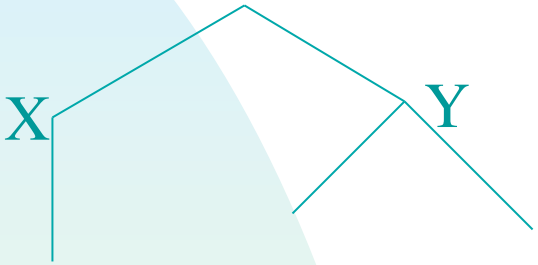
Contraintes sur la décomposition

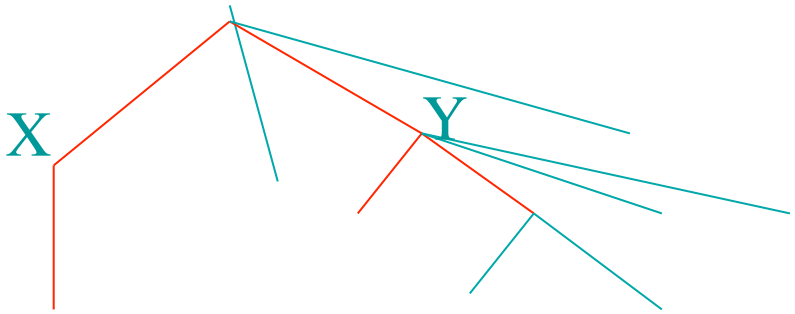
- Dépendance : immédiate ou généralisée
  - dans la reconnaissance d ’une sous-arborescence, chaque point différent de la racine, dépend directement d ’un autre point.





- Contrainte de continuité





S ’il y a une contrainte de continuité entre X et Y, l ’arbo-rescence de gauche ne peut pas être une sous-arborescence de l ’arborescence de droite.

- Contrainte d ’ordre

- Contrainte de présence
  - un noeud (et ses descendants) peut être déclaré optionnel
    - Le schéma A sera reconnu dans l ’arborescence B si le noeud X peut être déclaré optionnel.

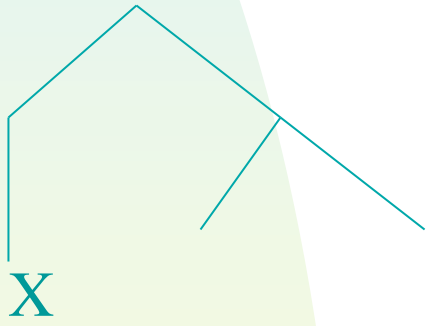


schéma A

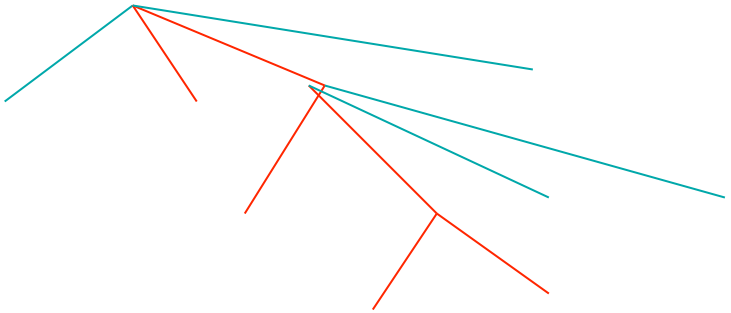
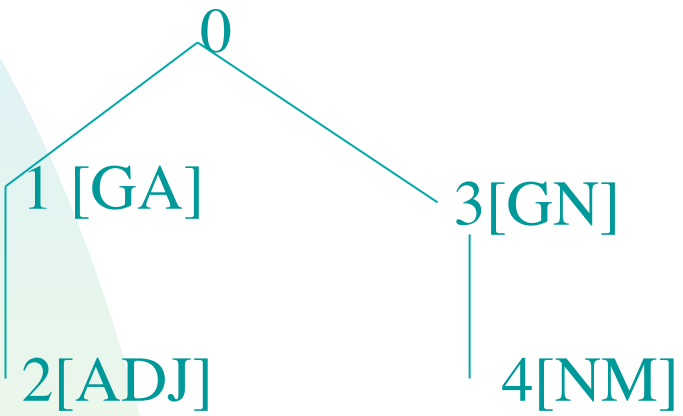


schéma B

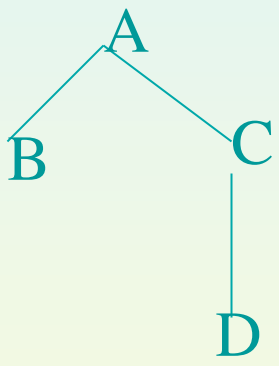
- Schéma d ’élément structuré
  - Il est avant tout défini par des conditions sur les étiquettes associées aux noeuds par la fonction d ’étiquetage.
  - Conditions :
    - expressions booléennes portant sur les valeurs des variables.
    - deux types
      - conditions propres : l ’expression booléenne ne peut faire référence qu ’à une seule étiquette du schéma
      - conditions inter-sommets : elle peut faire référence à tout ou partie de l ’ensemble des noeuds du schéma.

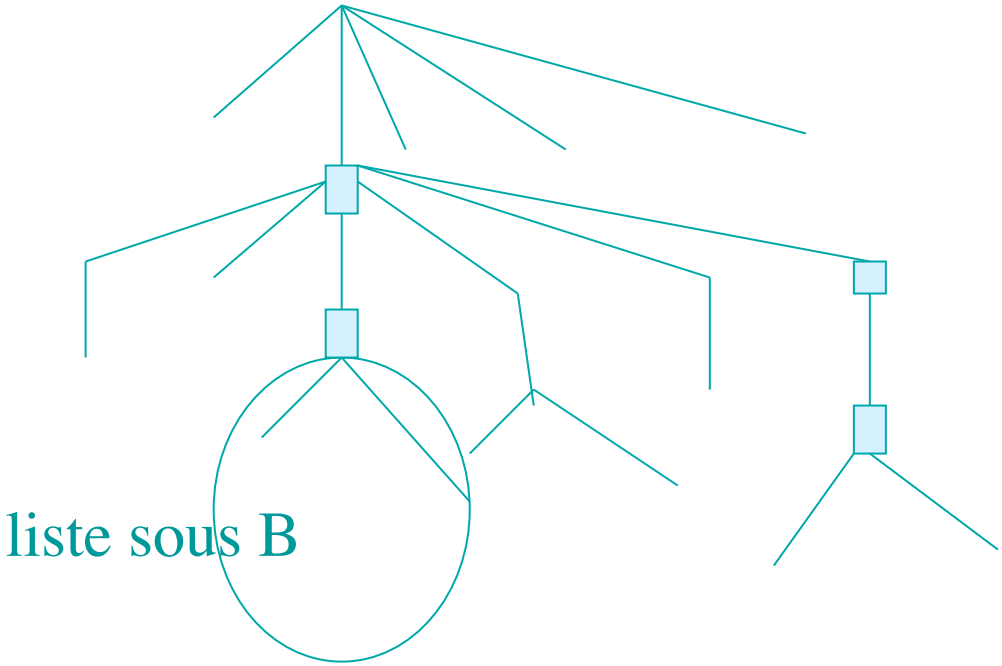
- Exemple de schéma
- (0(1(2)3(4)))



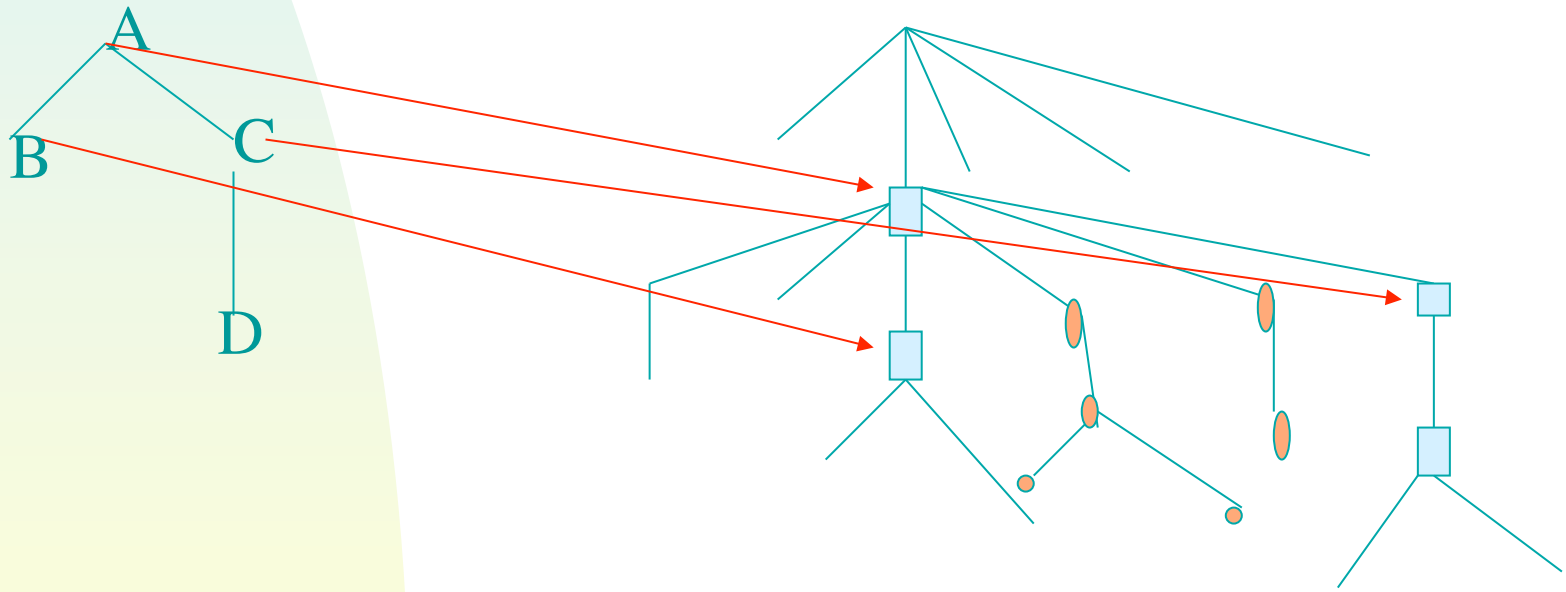
genre(2)&genre(4) != 0
Le nom et l'adjectif doivent être du même genre

- Identification d ’un schéma
  - lorsqu’un schéma est reconnu dans une arborescence, il identifie un ensemble de noeuds. Cette identification sépare les différents éléments de l ’arborescence en listes.





- liste d ’arborescence
  - A <B,C> : ensemble des éléments qui se trouvent sous le point associé à A, à droite du point associé à B et à gauche du point associé à C. A est obligatoire, B et C sont facultatifs.

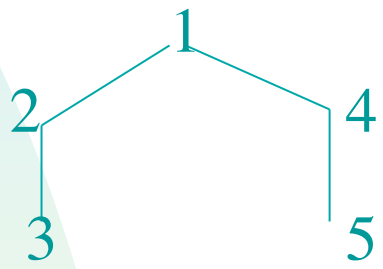


Réalisation des transformations

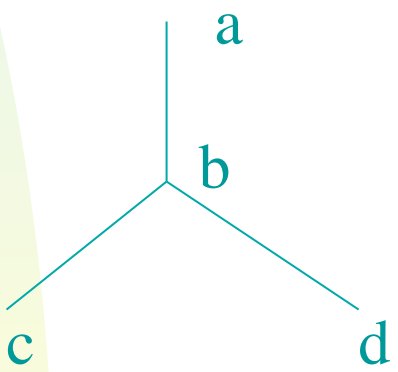
- Transformation d ’arborescence
  - on remplace la sous-arborescence, reconnue par le schéma de reconnaissance, par l ’arborescence de transformation.
- Transformation d ’élément structuré.
  - Une transformation est définie par un quadruplet (A, B, f, O)
    - A est un schéma de reconnaissance
    - B est un schéma de transformation
    - f est une fonction de l ’ensemble des listes de A dans l ’ensemble des listes de B.
    - O est une relation d ’ordre partielle telle que :
      - x et y deux listes de A telles que f(x)=f(y), alors 0(x,y) est défini.

Exemple

- On définit la transformation suivante :
  - schéma de reconnaissance

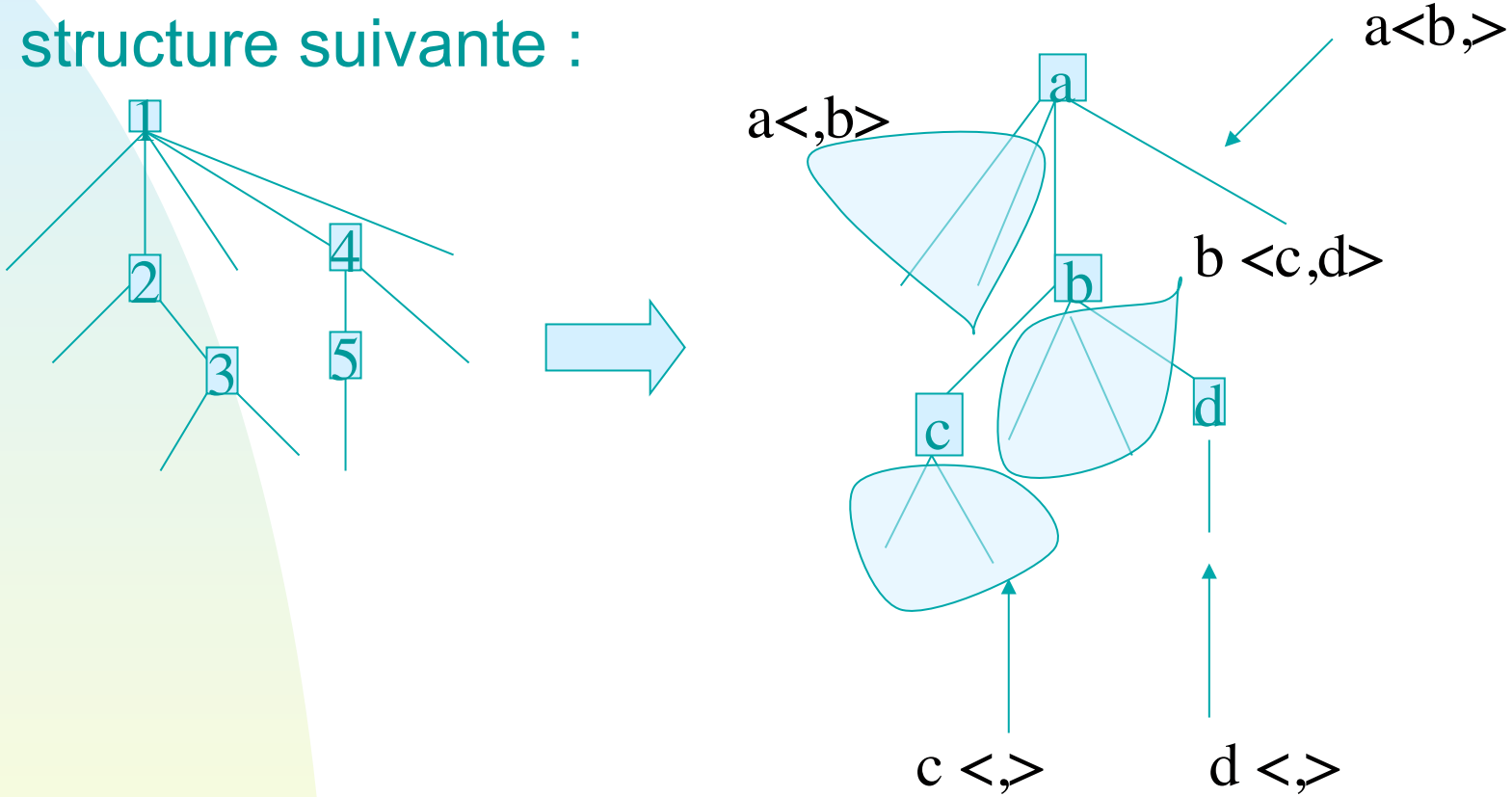


  - Schéma de transformation



- Des applications permettant de définir f et O:
  - 1<,2> —> a<,b>
  - 1<2,3> —> a<,b>
  - 1<3,> —>a<b,>
  - 2<,> —>b<c,d>
  - 3<,> —>b <c,d>
  - 4 <,> —>c <,>
  - 5 <,> —> d <,>
  - soit :
  - (1(2(3) 4(5))) —>(a[* 1<,2> *,* 1<2,3>*](b [*1<3,>*] (c [*2<,>*, *3<,>*, *4 <,>*] d [2<,>*, *3<,>*, *5 <,>*])))

- Cette transformation peut s ’appliquer sur la structure suivante :



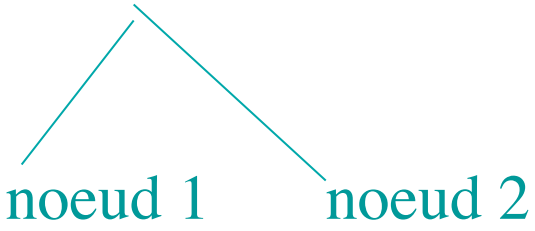
Grammaire

- Algorithme de Markov étendu aux éléments structurés.
  - un ensemble ordonné de règles de transformation.
  - modes de fonctionnement :
    - itératif (mode des algorithmes de Markov)
    - exhaustif (lorsqu ’une règle est appliquée elle est éliminée de la grammaire)
    - unitaire (mode itératif borné par une valeur numérique)

règle de transformation

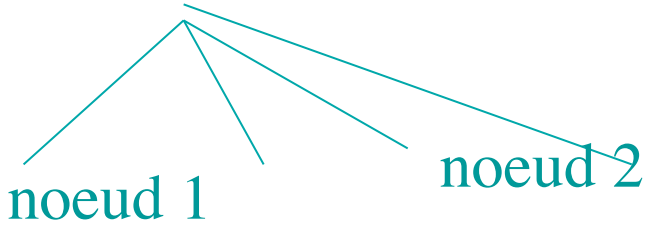
- Eléments de la règle (transformation)
  - forme de la structure à transformer :
    - noeud1 (noeud2)

    - noeud1, noeud 2



      - les noeuds doivent être contigus (contrainte de continuité)

- noeud1, *, noeud2



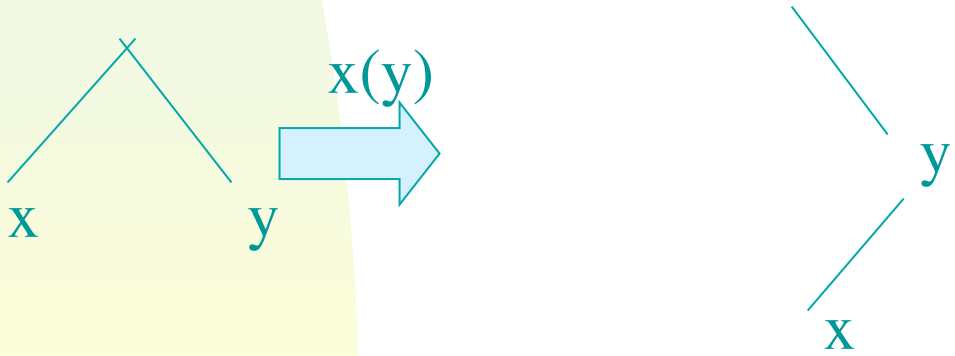
- conditions sur la structure à transformer :/ condition noeud1;condition noeud2;…;condition noeud n

- schéma de transformation:
  - insertion de noeud,
  - transformation de hiérarchie.

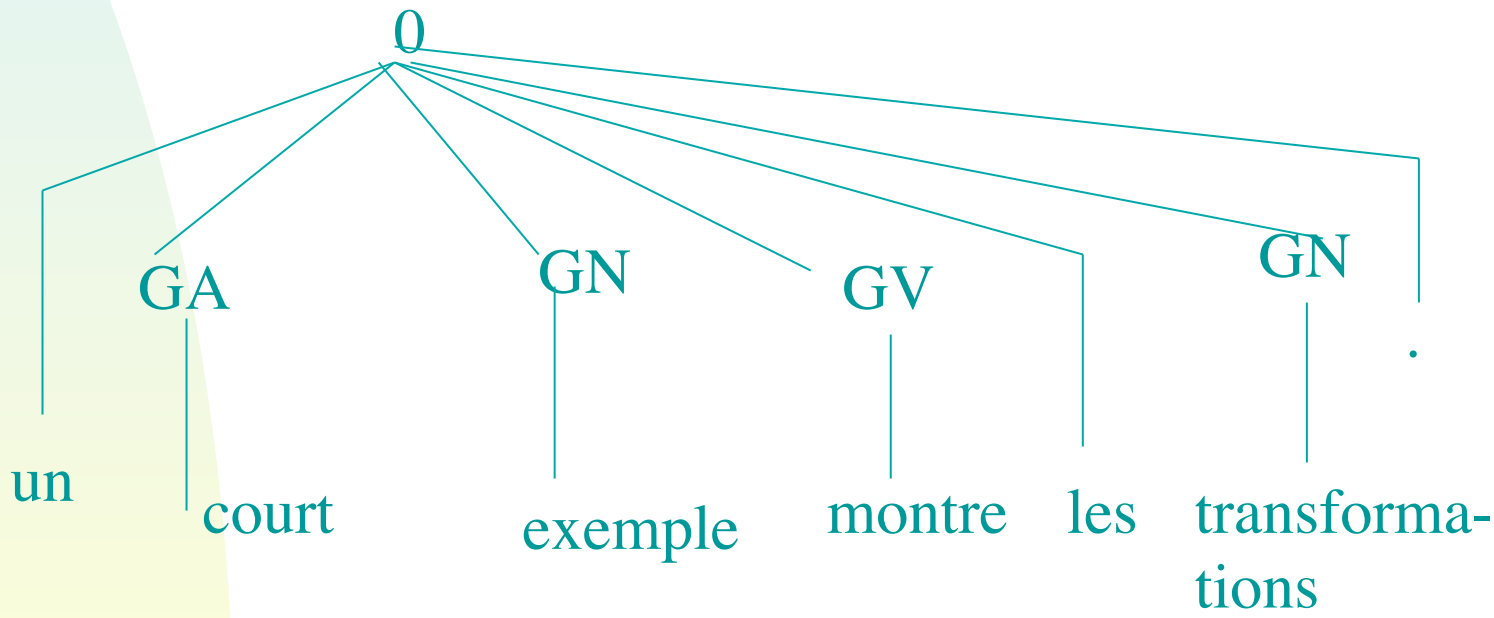
Type d'action

- insertion :
  - noeud-père:noeud-père (propriétés noeud-fils)

- modification de hiérarchie

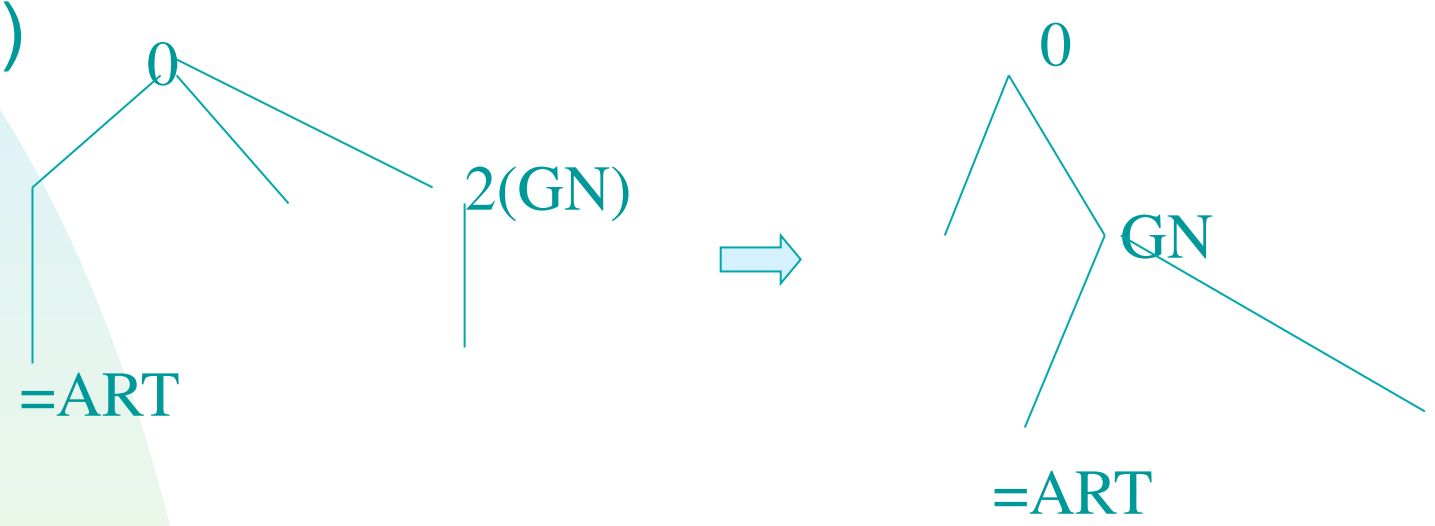


- R1 :0 (1),0:categorie = 0, 1: categorie = ADJ => 0:0: categorie = GA (insertion d ’un noeud sous 0)
- R2 : 0 (1), /0 :categorie = 0, 1: categorie = NM=>0:0: categorie = GN
- R3: 0 (1), 0 :categorie = 0, 1: categorie = V=> 0:0 : categorie = GV

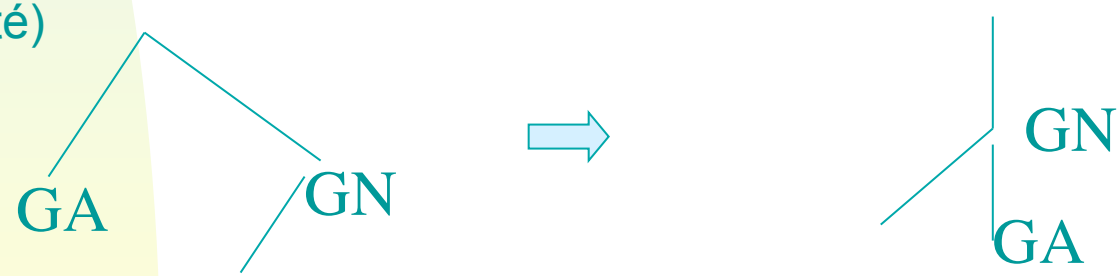


Règles de transformation

- R4:(0(1),*,2) /1:categorie =ART; 2: categorie GN =>2(1)

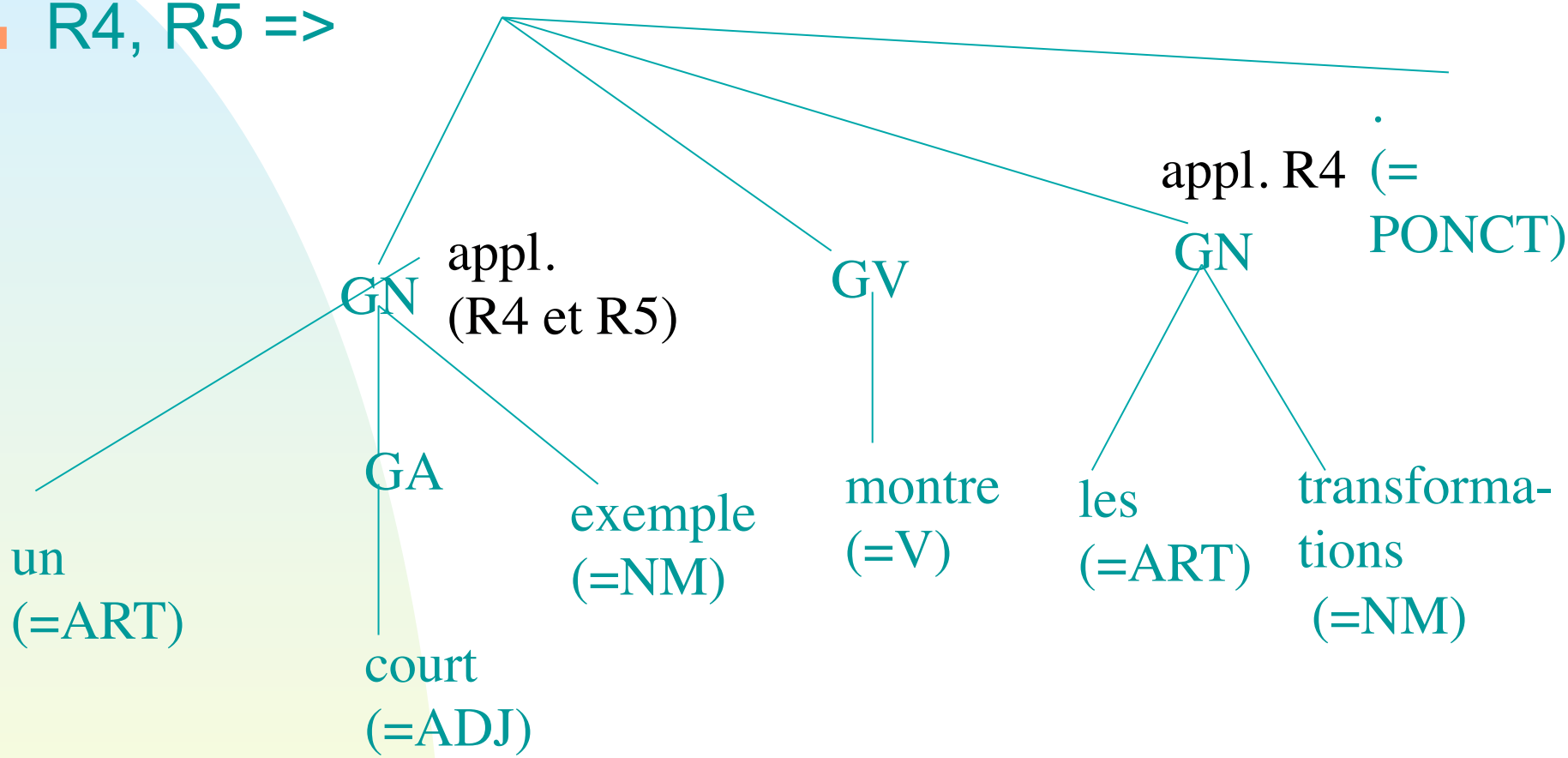


- R5:
  - (0,1)/0: categorie =GA, 1: categorie = GN => 1(0)
  - (continuité)

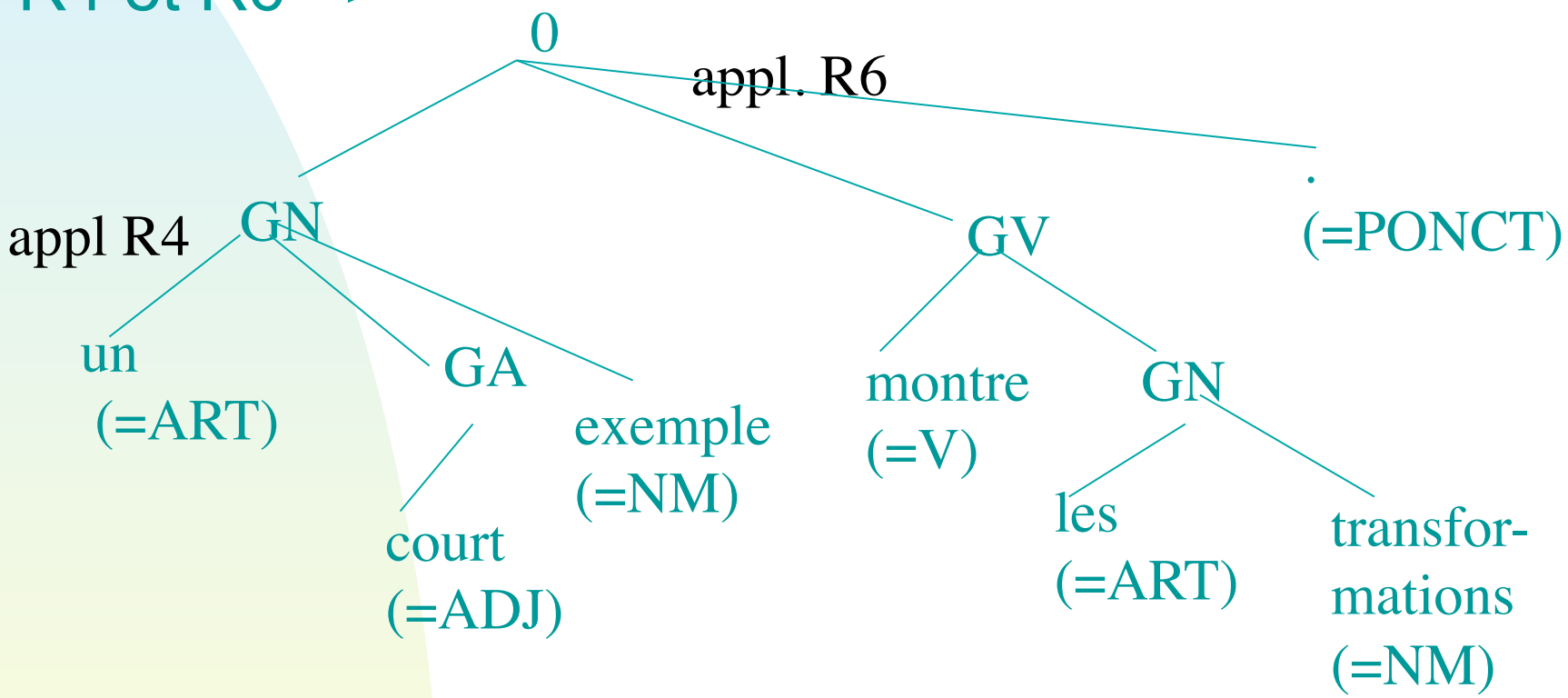


Application des règles

- R4, R5 =>



- R6 : (0,*,1)/0: categorie=GV; 1:categorie =GN => 0(1)
- R4 et R6 =>



Introduction de règles de dépendance

- Si une phrase est de la forme GN GV alors le GN est groupe sujet.
- R7: 0(1,2) /0: (K=PH); 1:(CAT=GN); 2: (CAT=GV) => 0(1,2)/ 1:1& FSUJ(2).
- Si un groupe nominal non prépositionnel est après un verbe transitif alors il est complément d'objet de ce verbe et doit lui être attaché.

Introduction de règles de dépendance

- R8: 0(1,2(3),*,4)/ 0: (K!=PH), 1:(CAT=GN); 2:(CAT=GV), 3:(CAT=V) & (TYP=TRANS), 4: (CAT=GN)=> 0(1,2(3, 4))) / 0: K<-PH; 4:4&FCOD(2)
- La règle R7 pourra être appliquée après la règle R8 (appel récursif à la grammaire).
- Il faut de préférence les mettre dans le bon ordre…

Conclusion sur les grammaires

- Les grammaires structurelles peuvent être efficaces.
- Les transformations doivent être préférentiellement localistes (ne pas mettre trop de conditions…) pour être plus performantes.
- L'ordre est important pour la complexité.
- Les transformations structurelles peuvent à la fois traiter des constituants comme des dépendances.
- On peut introduire des étiquettes sémantiques et aborder l'interface syntaxe-sémantique.