Outils formels pour la maitrise informatique de la langue naturelle

J. Chauché

Janvier 2006

Table des matières

INTRODUCTION

Le traitement informatique de la langue naturelle appartient au domaine désigné "intelligence artificielle". Dans le traitement automatique des langues naturelles sur ordinateurs, on retrouve tous les algorithmes classiques utilisés en IA :

– Techniques d'analyses des problèmes combinatoires.
– Recherche heuristique.
– Systèmes a base de règles et systèmes experts.
– Apprentissage.
– Démonstration automatique des théorèmes.

Chacun de ces aspects correspond à une application spécifique du traitement des langues naturelles. Ce traitement suppose une intérrogation constante par rapport aux techniques utilisées. Sans négliger les propriétés formelles des outils employés il est nécessaire de ne pas perdre de vue la finalité première de la formalisation qui est de fournir un moyen d'appréhender l'objet étudié.

L'utilisation en informatique de la langue naturelle suppose d'abord, comme pour tout langage, une reconnaissance de cette dernière. Cette reconnaissance correspond bien sûr à la fonction inverse de la génération. Parmis les modèles proposés pour cette reconnaissance certains formalimes sont directement issus d'un formalisme génératif. D'autres au contraire s'en écartent suffisament pour ne plus pouvoir définir de fa con précise le langage reconnu par ce formalisme. Le choix entre ces différentes approches dépend de l'application qui utilise la langue naturelle comme interface et aussi des applications disponibles à un certain moment. En effet une réalisation quelconque utilisant la langue naturelle ne peut pas négliger l'aspect quantitatif qui se trouve surtout concentré dans le lexique. Cet aspect est la premier frein au développement de projets informatiques qui utilisent la langue naturelle. La construction d'un dictionnaire n'étant pas encore normalisée toute application, surtout universitaire, se trouve bloquée devant cet obstacle : Une application qui est définie comme utilisant la langue naturelle (et non pas un sous-langage) doit pouvoir au moins appréhender le langage courant et donc posséder un lexique relativement important. L'aspect quantitatif ne se trouve d'ailleurs pas que dans le lexique. Certains formalismes, notamment ceux basés sur l'unification, ont beaucoup de mal à être maitrisés lorsque la couverture de la langue augmente et produisent souvent des ambiguités parasites qui pénalisent lourdement la communication. L'expérience de l'utilisation de l'algorithme de Cocke définissant un analyseur complet quelque soit la grammaire hors contexte employée montre que la difficulté dans la construction d'un analyseur se trouve aussi dans le nombre de règles à mettre en oeuvre. Cet algorithme est fondamental dans la mesure ou il permet formellement la construction d'un analyseur pour tout langage hors contexte. Il impose également la recherche de processus dépassant ce cadre des langages hors contextes.

Le but de la communication consiste à aquérir une représentation du sens. La logique formelle constitue un outil privilégié pour la représentation du sens. Tout les formalismes du traitement des langues naturelles sont capable d'intégrer cette composantes au moins dans la phase de reconnaissance. La séparation entre formalismes syntaxiques et formalismes sémantiques a été déterminée à partir d'un aspect opératoire. Les formalismes sémantiques ont vocation à être utilisés après une reconnaissance syntaxique. Les formalismes syntaxiques ont vocation à définir un traitement syntaxique et pour certains un traitement sémantique associé. Seul deux grandes familles d'analyse sont présentées. Les autres approches, comme les grammaires d'unification, les grammaires catégorielles ou les grammaires d'arbres adjoints font également l'objet d' étude. Les références bibliographiques permettent de combler cette lacune.

Dans cette présentation les notions fondamentales comme celle du "retour arrière" ou de la projection de deux graphes sont supposées connues.

Chapitre 1

Grammaires formelles

Les langues naturelles font partie des langages. Un langage est un dispositif qui permet une communication : langage des signes, langages de programmation, langue naturelle. Un des aspects fondamentaux de l'étude des langages consiste à définir un moyen de description des éléments de ce langage. Cette description peut s'accompagner d'une signification associée à chaque élément du langage. Par exemple la sémantique du signe '+' dans le langage mathématique sera différente suivant le contexte ou il est employé : addition de nombres entiers, de booléens, de matrice, etc... La définition d'un langage doit être finie. Cette définition peut être plus ou moins précise : La définition d'un langage de programmation est rigoureuse et ne permet pas de variations. La définition d'une langue naturelle ne suit pas les mêmes règles : l'historique de la langue et son évolution ne permettent pas d'obtenir un processus aussi statique que celui des langages artificiels.

Il existe deux grandes familles de procédés pour décrire un langage :

- La première consiste à définir un processus qui soit en mesure d'engendrer tous les éléments du langage. Ce processus est de type descriptif et les grammaires formelles, dérivées de la logique mathématique, en constituent l'axe principal.
- La seconde consiste à définir un processus de reconnaissance ou processus accepteur. Dans ce cas on est capable de déterminer si un élément donné correspond à un élément du langage. Ce processus est bien sûr indispenssable a toute application informatique puisqu'il permet d'attacher pendant la reconnaissance une signification au message reçu. Cette famille comprend comme outil de base la théorie des automates.

1.1 Définitions.

1.1.1 Vocabulaire, mots, monoïde.

Un langage est construit sur un ensemble de signes. L'ensemble de ces signes sera appelé "vocabulaire". Exemple :

- Le vocabulaire nécessaire à la construction du langage morse :

  { <espace>, ., - }.

- Le vocabulaire nécessaire à la construction du français :

{ a,...,z,A,...,Z,é,..,ç,',... }.
Le signe "#" correspond à un signe inconnu pour la construction du français.

La propriété importante que doit avoir un vocabulaire est sa finitude. On ne construit pas de langage sur des vocabulaires infinis. On a l'habitude formellement de définir un vocabulaire par les lettres minuscules de l'alphabet :

$$V = \{a, b, c, ...\}$$

ou

$$V = \{a_1, a_2, a_3, ..., b_1, b_2, ...\}$$

A l'aide d'un vocabulaire on construit un nouvel ensemble, V^*, qui est défini en même temps que l'opération de concaténation qui l'engendre :

- Les éléments de V^* sont appelés des mots.
- Il existe un mot ε appelé mot vide qui est élément neutre de la concaténation.
- $V \subset V^*$.
- L'opération de concaténation est une opération interne de V^* qui a deux mots de V^* associe le mot formé par la juxtaposition ordonnée des deux mots opérandes.
- Tout élément de V^* est la concaténation de deux éléments de V^*.

Ainsi du point de vue théorique les deux élements suivants sont des mots :

"un"

"ceci est un seul mot"

Du point de vue linguistique on parlera de mot dans le premier cas et de phrase dans le second.

Il sera aussi possible de parler de vocabulaire infini (les mots au sens linguistique du terme). Dans ce cas les mots correspondront à des phrases. On peut déjà remarquer que cette approche est un abus de langage qui n'altère pas la définition initiale puisqu'en définitif le vocabulaire utilisé pour construire les mots est fini.

L'ensemble V^* muni de la concaténation est un monoïde :

- La concaténation est une opération interne.
- La concaténation a un élément neutre : le mot vide ε.
- La concaténation est associative.

L'ensemble V^* est un monoïde libre :

Soit
- E un ensemble
- M un monoïde
- f une application de E dans M.

On dit que M est libre (dans E) si :
- Pour tout semi-groupe S
- Pour toute application g de E dans S

Il existe un seul homomorphisme h de M dans S tel que le diagramme suivant commute :

$$\begin{tikzcd}
E \arrow[rr, "f"] \arrow[dr, "g"'] & & M \arrow[dl, "h"] \\
& S &
\end{tikzcd}$$

V^* est libre :

Soit $f : V \to V^+ \quad (V^* - \varepsilon)$ définit par :

$f(a) = a \quad \forall a \in V$.

Soit g une fonction quelconque de V dans un semi-groupe S.

Soit alors h défini par :

$h(x_1 \cdots x_n) = g(x_1).g(x_2)\ldots g(x_n)$

$\forall x_1 \ldots x_n \in V^+$

L'associativité de l'opération dans S montre que h est un homomorphisme. $(f(x+y) = f(x) + f(y))$

$\forall a \in V$:

$(h \circ f)(a) = h(f(a))$

comme $f(a) = a : (h \circ f)(a) = g(a)$

donc : $g = h \circ f$

h est unique : si k est un autre homomorphisme :

$g = k \circ f$

alors :

$$\begin{aligned}
k(x_1 \cdots x_n) &= k(f(x_1) \cdots f(x_n)) \\
&= k(f(x_1)) \cdots k(f(x_n)) \\
&= g(x_1) \cdots g(x_n) \\
&= h(x_1 \cdots x_n)
\end{aligned}$$

1.1.2 Langages.

Un langage est une partie de V^*.

Les opérations ensemblistes sont définies sur les langages puisque ceux-ci sont des ensembles.

Les opérations complémentaires sur les langages correspondent à l'extension de l'opération de concaténation :

$L_1.L_2 = \{w_1.w_2 | w_1 \in L_1, w_2 \in L_2\}$

$$L^* = \bigcup_{i=0}^{\infty} L^i \quad L^0 = \{\varepsilon\}$$

Opération miroir :

$\forall a \in V, w \in V^* : \widetilde{a.w} = \tilde{w}.a$

1.1.3 Définition des grammaires formelles.

Une grammaire formelle est un quadruplet G = (V_N, V_t, S, P) où :
- V_N : est un ensemble fini le vocabulaire auxiliaire.
- V_t : est un ensemble fini le vocabulaire terminal.
- S : est l'axiome et appartient à V_N.
- P : est un ensemble fini de production :

$$P \in \{(V_N \cup V_t)^* - V_t^*\} \times (V_N \cup V_t)^*$$

On note "$x_1 \rightarrow x_2$" si $(x_1, x_2) \in P$.

Dérivation dans une grammaire (règle de réécriture) :

Soit w_1 et w_2 deux mots de $(V_N \cup V_t)^*$. $w_1 \underset{G}{\Rightarrow} w_2$ si et seulement si :
- $w_1 = x_1 x_2 x_3$
- $w_2 = x_1 x_4 x_3$
- $x_2 \rightarrow x_4 \in P$.

Soit $\overset{\star}{\underset{G}{\Rightarrow}}$ l'extension réflexive et transitive de $\underset{G}{\Rightarrow}$:

$w_1 \overset{\star}{\underset{G}{\Rightarrow}} w_2$ si et seulement si
- soit $w_1 = w_2$ (extension réflexive).
- soit $\exists w_3 \in (V_N \cup V_t)^*$ tel que :

$$w_1 \underset{G}{\Rightarrow} w_3 \overset{\star}{\underset{G}{\Rightarrow}} w_2 \text{ (extension transitive).}$$

Le langage engendré par la grammaire G est alors défini par :

$$L(G) = \{w \mid w \in V_t^* {}_\wedge S \overset{\star}{\underset{G}{\Rightarrow}} w\}$$

Exemple :

$V_N = \{A, B, C, S\}$
$V_t = \{a, b, c\}$

$$P = \{ \quad S \rightarrow ABC, S \rightarrow ASBC, CB \rightarrow BC, A \rightarrow a, aB \rightarrow ab, \\ bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc\}$$

$L(G) = \{a^n b^n c^n | n \geq 1\}$

S $\underset{G}{\Rightarrow}$ ASBC $\underset{G}{\Rightarrow}$ AASBCBC $\underset{G}{\Rightarrow}$ AAABCBCBC $\underset{G}{\Rightarrow}$ AAABBCCBC
$\underset{G}{\Rightarrow}$ AAABBCBCC $\underset{G}{\Rightarrow}$ AAABBBCCC $\underset{G}{\Rightarrow}$ aAABBBCCC $\underset{G}{\Rightarrow}$ aaABBBCCC
$\underset{G}{\Rightarrow}$ aaaBBBCCC $\underset{G}{\Rightarrow}$ aaabBBCCC $\underset{G}{\Rightarrow}$ aaabbBCCC $\underset{G}{\Rightarrow}$ aaabbbCCC
$\underset{G}{\Rightarrow}$ aaabbbcCC $\underset{G}{\Rightarrow}$ aaabbbccC $\underset{G}{\Rightarrow}$ aaabbbccc

1.1.4 Preuves dans les grammaires formelles.

Les principales preuves dans les grammaires formelles concernent l'équivalence de grammaires ou l'appartenance d'un mot au langage engendré.

Ces preuves s'appuient sur des démonstrations par récurrences sur la longueur des mots ou des dérivations. Dans le cas ou le raisonnement s'effectue sur une dérivation, la dérivation utilisée est souvent celle obtenue par le remplacement des symboles auxiliaires situés le plus à gauche à l'intérieur du mot.

Dérivation la plus à gauche.

Une dérivation est dite la plus à gauche si pour tout pas de dérivation :

$$w_1 \underset{G}{\Rightarrow} w_2 \text{ alors } w_1 = x_1x_2x_3,\ w_2 = x_1x_4x_3 \text{ et } x_1 \in V_t^*.$$

Une grammaire n'a pas toujours une dérivation la plus à gauche. Seules les grammaires hors contexte ont toujours ce type de dérivation.

Exemple :

- dérivation la plus à gauche de $a^2b^2c^2$:

  $$\mathrm{S} \underset{G}{\Rightarrow} \mathrm{ASBC} \underset{G}{\Rightarrow} \mathrm{aSBC} \underset{G}{\Rightarrow} \mathrm{aABCBC} \underset{G}{\Rightarrow} \mathrm{aaBCBC} \underset{G}{\Rightarrow} \mathrm{aabCBC}$$
  $$\underset{G}{\Rightarrow} \mathrm{aabBCC} \underset{G}{\Rightarrow} \mathrm{aabbCC} \underset{G}{\Rightarrow} \mathrm{aabbcC} \underset{G}{\Rightarrow} \mathrm{aabbcc}$$

- exemples de preuves :

  1. Propriété P : Dans la dérivation la plus à gauche les mots engendrés par la grammaire G précédente sont de la forme $a^n(BC)^n$:

     Preuve sur la longueur des mots (n)

     Propriété générale :

     Seules les deux premières règles sont telles que :

     $$w_1 \underset{G}{\Rightarrow} w_2 \text{ alors } |w_1| \neq |w_2|$$

     dans ce cas on a $|w_2| = |w_1| + 3$ règle 1
     $|w_2| = |w_1| + 2$ règle 2

     La longueur d'un mot engendré est donc :
     $1+3k+2$ avec k nombre de fois que la règle 1 est appliquée.
     donc $w \in L(G) \Rightarrow |w| = 3(k+1)$.

     La propriété P est vraie pour $n = 1 (k = 0)$:
     $S \underset{G}{\Rightarrow} ABC \underset{G}{\Rightarrow} aBC$
     Supposons la propriété vraie pour $n-1$.
     $S \underset{G}{\Rightarrow} ASBC \underset{G}{\Rightarrow} aSBC \overset{(n-1)}{\underset{G}{\Rightarrow}} aa^{(n-1)}(BC)^{(n-1)}BC = a^n(BC)^n$
     Pour qu'un mot puisse avoir une longueur supérieure à 3 il faut nécessairement que k soit non nul et donc que la règle 1 s'applique et la dérivation donnée est la seule possible.

  2. Soit la grammaire G definie par

     $$\begin{array}{rcl} S & \rightarrow & aSb \\ S & \rightarrow & ab \end{array}$$

     alors $L(G) = \{a^nb^n | n > 0\}$

     Preuve par récurrence sur le nombre de pas de dérivation :
     Propriété : a^nb^n est le seul mot produit en n pas de dérivation.
     La propriété est vraie pour 1 :

ab appartient à $L(G)$ et est engendré en 1 pas de dérivation :
$S \underset{G}{\Rightarrow} ab$
ab est le seul mot produit en 1 pas de dérivation.

Supposons la propriété est vraie pour $n-1$:

$S \underset{G}{\Rightarrow} aSb \overset{(n\text{-}1)}{\underset{G}{\Rightarrow}} aa^{n-1}b^{n-1}b = a^nb^n$

Une dérivation de longueur n commence nécessairement par la règle 1 ($n > 1$).

Donc la dérivation de longueur n donnée est la seule possible et

$L(G) = \{a^nb^n | n \geq 1\}$

1.1.5 Classification de Chomsky.

Les grammaires formelles et donc les langages engendrés sont classés suivant la forme des productions. Il existe quatre types de grammaires et donc quatre types de langages :

type 0 : aucune contraintes sur les productions.

type 1 : $w \to w' \in P \quad alors \quad |w| \leq |w'|$

type 2 : $w \to w' \in P \quad alors \quad |w| = 1$

type 3 : Toutes les règles de P sont de la forme :

$$\begin{aligned} \Sigma \to \quad & \sigma\Sigma' \quad où \\ & \Sigma \in V_N \\ & \sigma \in V_t \\ & \Sigma' \in V_N \cup \{\varepsilon\} \end{aligned}$$

Les différents langages engendrés sont alors dénommés :

type 0 : de type 0

type 1 : langage sous-contexte.

type 2 : langage hors-contexte.

type 3 : langage régulier, d'états finis.

Propriété de décidabilité :

Problème : Etant donné une grammaire G et un mot w :

Ce mot w appartient-il au langage L(G) engendré par G ?

type 0 : semi-décidable : Il existe un algorithme qui peut affirmer qu'un mot w appartient à l'ensemble des mots engendrés par G.

type 1,2,3 : décidable : Il existe un algorithme qui répond toujours à la question : w appartient-il à L(G) ?

Les applications informatiques sur le traitement des langues naturelles seront donc toujours réalisées en dehors du type 0.

1.1.6 Arborescence de dérivation.

Une arborescence de dérivation est définie pour les grammaires de type 2 et 3. Cette notion peut être parfois étendue au type 1.

Une arborescence de dérivation résume la production d'un mot dans une grammaire donnée. Elle permet ainsi d'associer plus aisément une signification à chaque élément composant le mot.

Une arborescence de dérivation est une arborescence étiquetée. Chaque point feuille de l'arborescence est étiqueté avec un symbole de V_t. Les autres points sont étiquetés avec des symboles de V_N.

> Si la règle "$S \to w$" est appliquée pour la génération d'un mot alors le point étiqueté par S aura comme descendants, et eux seuls, les points étiquetés avec les symboles de w.

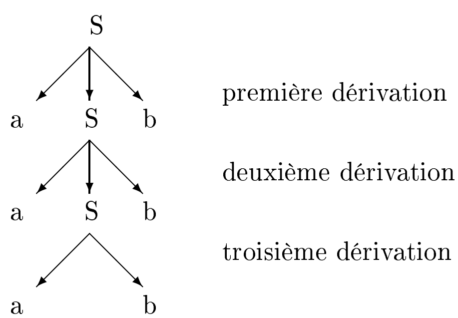
Exemple :

1. .

$$\begin{array}{lcll} S & \to & aSb & r_1 \\ S & \to & ab & r_2 \end{array}$$

$$S \underset{r_1}{\Rightarrow} aSb \underset{r_1}{\Rightarrow} aaSbb \underset{r_2}{\Rightarrow} aaabbb.$$

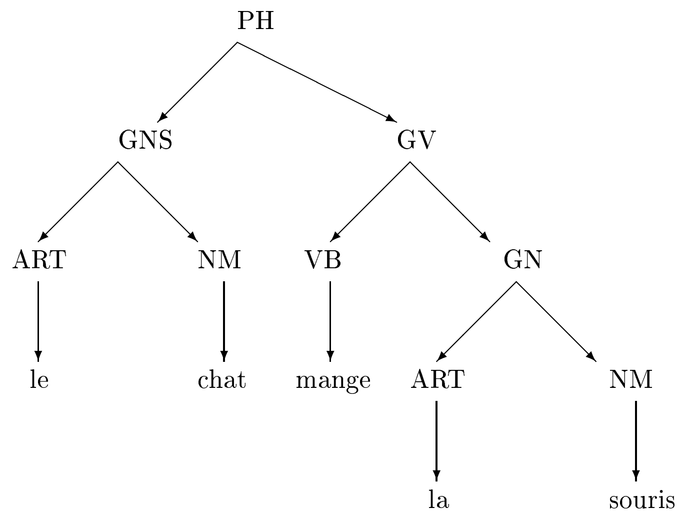
L'arborescence de dérivation est alors la suivante :



La lecture des feuilles donne le mot engendré.

2. .

$$\begin{array}{rcll} PH & \to & GNS\ GV & PH : \text{phrase} \\ GNS & \to & ART\ NM & GNS : \text{groupe nominal sujet} \\ GV & \to & VB\ GN & GV : \text{groupe verbal} \\ GN & \to & ART\ NM & GN : \text{groupe nominal} \\ ART & \to & le & ART : \text{article} \\ ART & \to & la & \\ NM & \to & chat & NM : \text{nom} \\ NM & \to & souris & \\ VB & \to & mange & VB : \text{verbe} \end{array}$$



L'arborescence de dérivation est appelée structure syntaxique.

1.1.7 Ambiguïtés.

Une grammaire est ambigüe si pour un mot qu'elle engendre il existe plusieurs arborescences de dérivations distinctes.

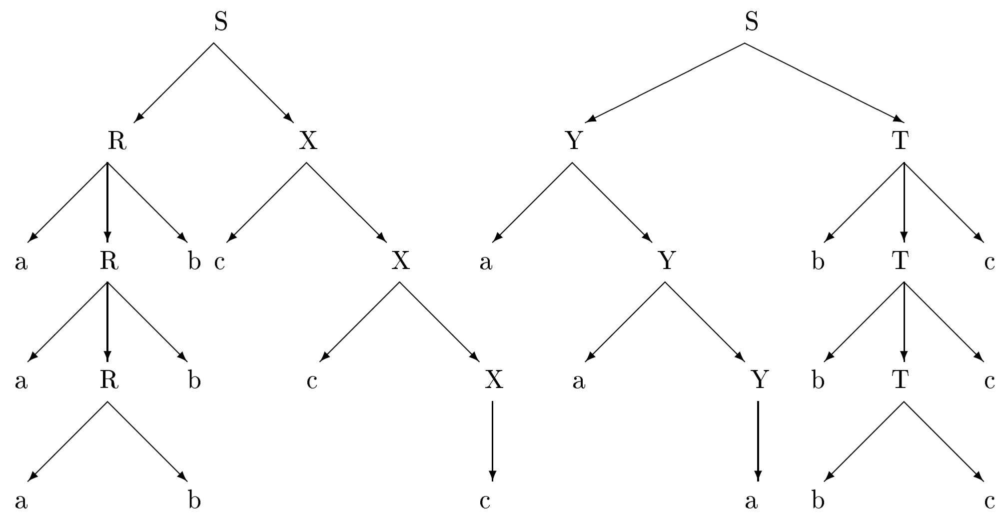
Un langage est ambigü si chaque grammaire qui l'engendre est ambigüe.

Exemple :

Soit la grammaire :

$$\begin{array}{ll} S \to RX & S \to YT \\ X \to cX & Y \to aY \\ X \to c & Y \to a \\ R \to aRb & T \to bTc \\ R \to ab & T \to bc \end{array}$$

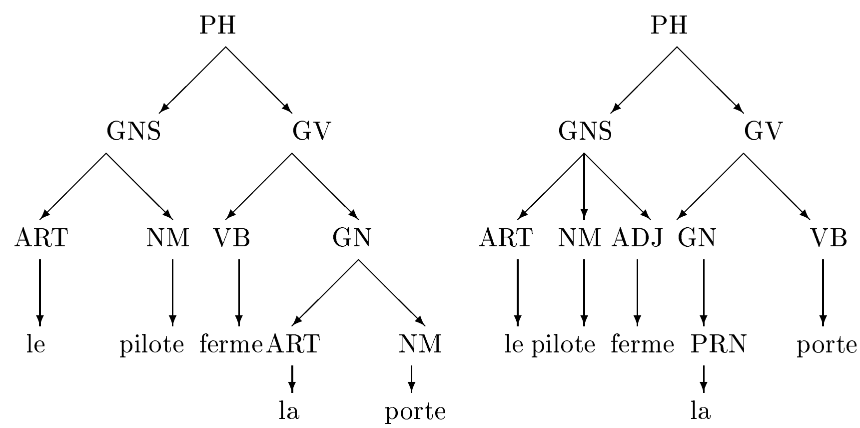
Le mot $a^3b^3c^3$ est engendré par cette grammaire de deux façons. A chaque dérivation correspond une structure syntaxique distincte :



Ces deux structures syntaxiques correspondent à deux intreprétations :

- $a^3b^3c^3$ est une suite bien parenthèsée de ab suivie d'un nombre quelconque de c.
- $a^3b^3c^3$ est une suite bien parenthèsée de bc précédée d'un nombre quelconque de a.

La langue naturelle est ambiguë.



il ferme la porte il la porte

La construction des langages de programmation doit éliminer les ambiguïtés.

Le traitement des langues naturelles doit traiter les ambiguïtés. Ce traitement se fera souvent par la définition d'un choix d'interprétation, qui correspond souvent au choix d'une structure syntaxique.

1.2 Analyse dans les grammaires formelles.

L'analyse dans les grammaires formelles consiste à rechercher si un mot donné appartient au langage engendré par cette grammaire.

Suivant le type de grammaire la réponse à ce problème est différente :

Pour les grammaires de type 1 les algorithmes utilisables sont des algorithmes basés sur des moteurs d'inférences (Une règle de grammaire est en fait une simple règle de réécriture ou une règle d'inférence). La recherche de l'appartenance d'un mot consiste à prouver que ce mot peut être déduit de l'application des règles.

Pour les grammaires de type 2 l'algorithme le plus général est l'algorithme de Cocke. Il permet d'obtenir toutes les structures syntaxiques d'une phrase d'un langage engendré par une grammaire d'un certain type. D'autres analyseurs peuvent être construits pour les langages hors-contexte. On peut citer par exemple l'analyseur basé sur la notion de chaînes développé au LADL (Salkoff). Pour les cas particuliers des langages de programmation les analyseurs sont déterministes et basés sur la notion d'automate à pile.

Pour les grammaires de type 3 la reconnaissance s'effectue par un automate d'états finis. Bien que l'équivalence entre un automate déterministe et non déterministe soit démontrée l'utilisation de ce formalisme pour les langues naturelles utilise toujours la forme non déterministe.

1.2.1 Analyse des langages de type 3.

1.2.1.1 Automate d'états finis.

Un automate d'états finis est un quadruplet $A = (V_t, Q, q_0, F, \mu)$ où

V_t est le vocabulaire terminal fini.
Q est l'ensemble fini des états.
q_0 est l'état initial, $q_0 \in Q$.
$F \subseteq Q$, est l'ensemble des états finals.
μ est la fonction de transition : $Q \times V_t \to \mathcal{P}(Q)$

La fonction de transtion induit une relation sur $Q \times V_t^*$:

$$(q, \sigma w) \underset{\mathrm{A}}{\vdash} (q', w) \quad \forall q, q' \in Q, \sigma \in V_t, w \in V_t^*,$$
$$\text{tel que } q' \in \mu(q, \sigma)$$

Soit $\overset{\star}{\underset{\mathrm{A}}{\vdash}}$ l'extension réflexive et transitive de la relation $\underset{\mathrm{A}}{\vdash}$:

$$(q, w) \overset{\star}{\underset{\mathrm{A}}{\vdash}} (q, w) \; \forall q \in Q, w \in V_t^*$$

$$(q, \sigma w) \overset{\star}{\underset{\mathrm{A}}{\vdash}} (q', w') \text{ si et seulement si } \exists q" \in Q \text{ tel que :}$$

$$(q, \sigma w) \underset{\mathrm{A}}{\vdash} (q", w) \text{ et } (q", w) \overset{\star}{\underset{\mathrm{A}}{\vdash}} (q', w')$$

1.2.1.2 Reconnaissance d'un mot par l'automate A :

Un mot w de V_t^* est reconnu par A si et seulement si :

$$\exists q \in F \text{ tel que : } (q_0, w) \overset{\star}{\underset{\mathrm{A}}{\vdash}} (q, \varepsilon)$$

$$L(A) = \{ \; w \mid \exists q \in F : (q_0, w) \overset{\star}{\underset{\mathrm{A}}{\vdash}} (q, \varepsilon)\}$$

1.2.1.3 Propriété :

Pour toute grammaire de type 3 il existe un automate d'états finis acceptant le langage engendré par cette grammaire :

$G = (V_N, V_t, S, P)$

$A = (V_t, V_N \cup \{f\}, S, \{f\}, \mu)$ où μ est défini par :
$\Sigma \in \mu(\Sigma', \sigma)$ si et seulement si $\Sigma' \to \sigma\Sigma \in P$.
$f \in \mu(\Sigma', \sigma)$ si et seulement si $\Sigma' \to \sigma \in P$.

Montrons par récurrence sur la longueur des mots la propriété :

$$(\Sigma, w) \overset{\star}{\underset{A}{\vdash}} (f, \varepsilon) \text{ si et seulement si } \Sigma \overset{\star}{\underset{G}{\Rightarrow}} w$$

La propriété est vraie pour les mots de longueur 1 :

$$\begin{aligned}
(\Sigma, \sigma) \overset{\star}{\underset{A}{\vdash}} (f, \varepsilon) &\Leftrightarrow (\Sigma, \sigma) \underset{A}{\vdash} (f, \varepsilon) \\
(\Sigma, \sigma) \underset{A}{\vdash} (f, \varepsilon) &\Leftrightarrow \Sigma \to \sigma \in P \text{ donc} \\
&\Leftrightarrow \Sigma \underset{G}{\Rightarrow} \sigma
\end{aligned}$$

Supposons la propriété vraie pour tous les mots de longueur n-1 :
Soit w de longeur n.
$w = \sigma w'$ avec $\sigma \in V_t$ et w' de longueur $n - 1$.

$$\begin{aligned}
(\Sigma, w) \overset{\star}{\underset{A}{\vdash}} (f, \varepsilon) &\Leftrightarrow (\Sigma, \sigma w') \overset{\star}{\underset{A}{\vdash}} (f, \varepsilon) \\
&\text{et par définition de } \overset{\star}{\underset{A}{\vdash}} : \\
&\Leftrightarrow (\Sigma, \sigma w') \underset{A}{\vdash} (\Sigma', w') \text{ et } (\Sigma', w') \overset{\star}{\underset{A}{\vdash}} (f, \varepsilon)
\end{aligned}$$

Par hypothèse de récurrence et comme $|w'| \neq 0$:
$(\Sigma', w') \overset{\star}{\underset{A}{\vdash}} (f, \varepsilon)$ si et seulement si $\Sigma' \overset{\star}{\underset{G}{\Rightarrow}} w'$ et $\Sigma' \in V_N$
Par définition de la relation $\underset{A}{\vdash}$: $(\Sigma, \sigma w) \underset{A}{\vdash} (\Sigma', w') \Leftrightarrow \Sigma \to \sigma\Sigma' \in P$.

$(\Sigma, \sigma w') \underset{A}{\vdash} (\Sigma', w')$ et $(\Sigma', w') \overset{\star}{\underset{A}{\vdash}} (f, \varepsilon)$ si et seulement si

$$\Sigma \to \sigma\Sigma' \in P \text{ et } \Sigma' \overset{\star}{\underset{G}{\Rightarrow}} w'.$$

donc si et seulement si

$$\Sigma \underset{G}{\Rightarrow} \sigma\Sigma' \overset{\star}{\underset{G}{\Rightarrow}} \sigma w' \text{ Soit } \Sigma \overset{\star}{\underset{G}{\Rightarrow}} \sigma w' = w$$

$w \in L(G)$ si et seulement si $S \overset{\star}{\underset{G}{\Rightarrow}} w$ donc si et seulement si $(S, w) \overset{\star}{\underset{A}{\vdash}} (f, \varepsilon)$
soit $w \in L(G)$ si et seulement si $w \in L(A)$.

Les analyseurs morphologiques utilisent le formalisme des automates d'états finis. Les éléments du vocabulaire sont alors des segments, c'est à dire des suites finies de caractères. Le nombre d'éléments est alors très important. Le non déterminisme est obligatoire car il correspond à plusieurs interprétations :

Segments :

'ferm'	racine du verbe 'fermer'
'ferme'	nom commun
'ferment'	nom commun
's'	suffixe du pluriel
'er'	suffixe de l'infinitif
'es'	suffixe de la conjugaison 2eme personne singulier
'e'	suffixe de la conjugaison 1 ou 3eme personne sing.
'ent'	suffixe de la conjugaison 3eme personne pluriel.

Grammaire (c'est à dire l'automate) :

S $\rightarrow$ ferm	SV
S $\rightarrow$ ferme	SN
S $\rightarrow$ ferme	
S $\rightarrow$ ferment	SN
S $\rightarrow$ ferment	
SV $\rightarrow$ er	
SV $\rightarrow$ es	
SV $\rightarrow$ e	
SV $\rightarrow$ ent	
SN $\rightarrow$ s	

Les mots ferme, fermes, ferment peuvent être engendrés de plusieurs façons. On repère ces différentes façons par leurs découpages, c'est à dire en repérant les différents segments qui forment un mot :

ferme	nom commun
ferm|e	verbe conjugué
ferme|s	nom commun
ferm|es	verbe conjugué
ferment	nom commun
ferm|ent	verbe conjugué

Par contre le mot ferments n'a ici qu'une seule génération possible :

ferment|s nom commun

La construction de l'automate déterministe équivalent ne pourrait pas associer les différentes interprétations définies précédemment.

1.3 Analyse des langages de type 2.

La reconnaissance d'un mot d'un langage hors contexte peut être définie de façon déterministe par plusieurs algorithmes. Chaque algorithme utilise une forme particulière de grammaire. L'algorithme de Cocke utilise les grammaires de forme normale de Chomsky.

1.3.1 Forme normale de Chomsky.

Une grammaire hors contexte est dite de forme normale de Chomsky si toutes ces règles sont de la forme :

soit $\Sigma \to \Sigma'\Sigma"$
soit $\Sigma \to \sigma$
$\Sigma, \Sigma', \Sigma" \in V_N, \sigma \in V_t$.

1.3.1.1 Propriètè :

Pour toute grammaire hors contexte G il existe une grammaire hors contexte G' de forme normale de Chomsky équivalente, c'est à dire tel que L(G) = L(G').

La preuve de cette équivalence comporte 3 étapes. A chaque étape une nouvelle grammaire équivalente est construite.

- Etant donné une grammaire G il existe une grammaire équivalente G_1 qui ne possède pas de production de la forme $\Sigma \to \Sigma'$, $\quad \Sigma, \Sigma' \in V_N$.

  Soit une grammaire $G = (V_N, V_t, S, P)$. Les productions de la forme $\Sigma \to \Sigma'$ sont appelées de type $'x'$ et les autres de type $'y'$.

  Soit le nouvel ensemble de productions P_1 construit de la façon suivante :

  1. P_1 contient toutes les productions de type $'y'$.
  2. Remarquons qu'il est facile de voir que $\Sigma \underset{G}{\overset{\star}{\Rightarrow}} \Sigma'$ puisqu'alors $\Sigma \underset{G}{\Rightarrow} \Sigma_1 \underset{G}{\Rightarrow} \Sigma_2 \cdots \underset{G}{\Rightarrow} \Sigma'$ et que si un non terminal apparait deux fois la dérivation peut être raccourcie. Il suffit donc de considérer toutes les séquences de longueur maximale $Card(V_N)$.
  3. Si $\Sigma \underset{G}{\overset{\star}{\Rightarrow}} \Sigma'$, Σ et $\Sigma' \in V_N$ alors on ajoute à P_1 toutes les productions de la forme $\Sigma \to w$ tel que $\Sigma' \to w$ est une production de type $'y'$ de P.

  Soit $G_1 = (V_N, V_t, S, P_1)$. Alors G_1 ne contient pas de production de type $'x'$ par construction et $L(G) = L(G_1)$.

  1. $L(G_1) \subseteq L(G)$.
     Si $\Sigma \to w \in P_1$ alors $\Sigma \underset{G}{\Rightarrow} w$ donc $L(G_1) \subseteq L(G)$.
  2. $L(G) \subseteq L(G_1)$.
     Soit $w \in L(G)$ et soit la dérivation la plus à gauche de w :
     $S \underset{G}{\Rightarrow} w_0 \underset{G}{\Rightarrow} w_1 \underset{G}{\Rightarrow} .... \underset{G}{\Rightarrow} w_n = w$

Si $0 <= i < n$ $w_i \underset{\mathrm{G}}{\Rightarrow} w_{i+1}$ par une production de type $'y'$ alors $w_i \underset{G_1}{\Rightarrow} w_{i+1}$.
Supposons que $w_i \underset{\mathrm{G}}{\Rightarrow} w_{i+1}$ par une production de type $'x'$ mais que $w_{i-1} \underset{\mathrm{G}}{\Rightarrow} w_i$ par une production de type $'y'$ ou $i = 0$. Alors $w_i, w_{i+1}, \cdots, w_j$ sont tous des mots de même longueur et puisque c'est la dérivation la plus à gauche le symbole remplacé est toujours à la même position. De plus $w_j \underset{\mathrm{G}}{\Rightarrow} w_{j+1}$ par une production de type $'y'$. Donc $w_i \underset{G_1}{\Rightarrow} w_{j+1}$ par une production de $P_1 - P$ et $S \underset{G_1}{\overset{\star}{\Rightarrow}} w$.

Donc $L(G) \subseteq L(G_1)$ et $L(G) = L(G_1)$.

– A partir de la grammaire G_1 on construit une grammaire G_2 possedant la propriété :

  si $\Sigma \to w \in P_2$ alors
  – soit $w \in V_t$
  – soit $w \in V_N^*$ et $|w| \geq 2$.

  Les règles de G_1 sont de deux formes :

  Soit $\Sigma \to w \quad |w| \geq 2$.
  Soit $\Sigma \to \sigma \quad \sigma \in V_t$

  L'ensemble des règles P_2 est construit de la façon suivante :

  1. Les règles du deuxième type appartiennent à P_2.
  2. Soit une règle du premier type :

     $\Sigma \to \Sigma_1 \Sigma_2 ... \Sigma_n \quad n \geq 2$

     Cette règle est transformée de la façon suivante :
     Si Σ_i est un symbole de V_t ($\Sigma_i = \sigma_j$) on crée un nouveau symbole non terminal Σ_{σ_j} et on ajoute à P_2 la production :

     $\Sigma_{\sigma_j} \to \sigma_j$

     Le symbole Σ_i est alors remplacé par Σ_{σ_j} dans la règle précédente.
     La production devient alors de la forme :

     $\Sigma \to \Sigma'_1 \Sigma'_2 ... \Sigma'_n$ où $\forall i\ \Sigma'_i \in V'_N$.

     Cette production est alors ajoutée à P_2.

  La grammaire G_2 devient alors :
  $G_2 = (V'_N, V_t, S, P_2)$ avec :

  $V'_N = V_N \cup \{\Sigma_{\sigma_i} | \sigma_i \in V_t\}$
  P_2 : ensemble des règles définies précédemment.

  $L(G_1) = L(G_2)$:

  1. Si $w \underset{G_1}{\Rightarrow} w'$ en une production alors $w \underset{G_2}{\overset{\star}{\Rightarrow}} w'$ en n productions donc $L(G_1) \subseteq L(G_2)$.
  2. Montrons par induction sur le nombre de pas de dérivation que si $\Sigma \underset{G_2}{\overset{\star}{\Rightarrow}} w$, $\Sigma \in V_N, w \in V_t^*$ alors $\Sigma \underset{G_1}{\overset{\star}{\Rightarrow}} w$.

- Le résultat est trivial pour un pas de dérivation : Les seules règles possibles sont des règles de G_1 du deuxième type.
- Supposons le résultat vrai pour n pas de dérivation :
  Soit $\Sigma \underset{G_2}{\overset{\star}{\Rightarrow}} w$ en $n+1$ pas
  Le premier pas est de la forme :
  $$\Sigma \underset{G_2}{\Rightarrow} \Sigma_1\Sigma_2...\Sigma_m \text{ avec } m \geq 2$$
  on peut alors écrire $w = w_1w_2...w_m$ avec $\forall i : \Sigma_i \underset{G_2}{\overset{\star}{\Rightarrow}} w_i$.
  Si $\Sigma_i \in V'_N - V_N$ on ne peut utiliser qu'une seule et nouvelle production de la forme :
  $$\Sigma_i \rightarrow \sigma_j \text{ avec } \sigma_j \in V_t$$
  Dans ce cas $w_i = \sigma_j$ et par construction de P_2 il existe une production de P_1 tel que
  $$\Sigma \rightarrow \Sigma'_1\Sigma'_2...\Sigma'_m \text{ où}$$
  $$\Sigma'_i = \Sigma_i \text{ si } \Sigma'_i \in V_N$$
  $$\Sigma'_i = \sigma_j \text{ si } \Sigma'_i \in V'_N - V_N$$
  Pour les Σ_i appartenant à V_N par hypothèse d'induction : $\Sigma_i \underset{G_2}{\overset{\star}{\Rightarrow}}$ w_i en moins de n pas
  donc $\Sigma_i \underset{G_1}{\overset{\star}{\Rightarrow}} w_i$

  Donc $\Sigma \underset{G_1}{\Rightarrow} \Sigma'_1\Sigma'_2...\Sigma'_m \underset{G_1}{\overset{\star}{\Rightarrow}} w_1w_2...w_m = w$ et $L(G_2) \subseteq L(G_1)$ soit $L(G_2) = L(G_1)$.
- A partir de G_2 on construit G_3 de forme Normale de Chomsky.
  G_2 à toutes ses règles de la forme :
  soit $\Sigma \rightarrow \sigma \quad \sigma \in V_t$
  soit $\Sigma \rightarrow w \quad w \in V_N^*$ et $|w| \geq 2$
  Toutes les règles de forme normale appartiennent à P_3.
  Soit une règle de P_2 qui ne soit pas de forme normale :
  $$\Sigma \rightarrow \Sigma_1\Sigma_2...\Sigma_m \quad m \geq 3$$
  On crée alors $m-2$ symboles supplémentaires $\Sigma'_1, \Sigma'_2, ...\Sigma'_{m-2}$ et la règle est remplacée dans P_3 par l'ensemble des règles :
  $$\Sigma \rightarrow \Sigma_1\Sigma'_1$$
  $$\Sigma'_1 \rightarrow \Sigma_2\Sigma'_2$$
  ....
  $$\Sigma'_{m-2} \rightarrow \Sigma_{m-1}\Sigma_{m-2}$$
  V_N" contient alors les symboles de V'_N et l'ensemble des symboles ainsi créés.
  $L(G_2) = L(G_3)$:
  - Si $\Sigma \underset{G_2}{\Rightarrow} w$ alors $\Sigma \underset{G_3}{\overset{\star}{\Rightarrow}} w$ donc $L(G_2) \subseteq L(G_3)$.
  - Montrons par induction sur le nombre de pas de dérivation que :

Pour tout symbole $\Sigma \in V'_N$, si $\Sigma \overset{\star}{\underset{G_3}{\Rightarrow}} w, w \in V_t^*$ alors $\Sigma \overset{\star}{\underset{G_2}{\Rightarrow}} w$.

- La propriété est évidente pour $n = 1$ (Les seule règles utilisables sont de la forme $\Sigma \rightarrow \sigma, \sigma \in V_t$).
- Supposons la propriété vraie pour n et soit $\Sigma \overset{\star}{\underset{G_3}{\Rightarrow}} w$ une dérivation de longueur $n+1$.
  Le premier pas de cette dérivation est de la forme

  $$\Sigma \underset{G_3}{\Rightarrow} \Sigma_1\Sigma_2 \overset{\star}{\underset{G_3}{\Rightarrow}} w$$

  Par application de la règle $\Sigma \rightarrow \Sigma_1\Sigma_2$. Si cette règle appartient à G_2 alors $\Sigma \overset{\star}{\underset{G_2}{\Rightarrow}} w$ car Σ_1 et Σ_2 appartiennent nécessairement à V'_N et par hypothèse de récurrence $\Sigma_1\Sigma_2 \overset{\star}{\underset{G_2}{\Rightarrow}} w$.
  Si cette règle n'appartient pas à G_2 alors $\Sigma_1 \in V'_N$, $\Sigma_2 \in V_N" - V'_N$ et il existe une seule règle dans G_2 ayant produit la règle $\Sigma \rightarrow \Sigma_1\Sigma_2$. Soit la règle de G_2 à l'origine de la règle $\Sigma \rightarrow \Sigma_1\Sigma_2$ alors cette règle est de la forme
  $\Sigma \rightarrow \Sigma_1\Sigma'_2...\Sigma'_m$
  et les seules règles de G_3 ayant comme partie gauche $\Sigma_2, ...\Sigma_{m-2}$ sont de la forme

  $\Sigma_2 \rightarrow \Sigma'_2\Sigma_3$
  ...
  $\Sigma_{m-2} \rightarrow \Sigma'_{m-1}\Sigma'_m$

  La dérivation la plus à gauche de w dans G_3 est donc de la forme :

  $$\Sigma \underset{G_3}{\Rightarrow} \Sigma_1\Sigma_2 \underset{G_3}{\Rightarrow} ... \underset{G_3}{\Rightarrow} w_1\Sigma'_2\Sigma_3 \underset{G_3}{\Rightarrow} .... \underset{G_3}{\Rightarrow} w_1w_2...\Sigma'_{m-1}\Sigma'_m \underset{G_3}{\Rightarrow} w_1w_2...w_m$$

  avec $\forall i \Sigma'_i \overset{\star}{\underset{G_3}{\Rightarrow}} w_i$
  Donc par hypothèse de récurrence :
  $\Sigma \underset{G_2}{\Rightarrow} \Sigma'_1\Sigma'_2...\Sigma'_m \overset{\star}{\underset{G_2}{\Rightarrow}} w_1w_2...w_m$ et $L(G_3) \subseteq L(G_2)$.
  donc $L(G_2) = L(G_3)$.

1.3.2 Algorithme de Cocke : reconnaissance des langage de type 2.

Pour tout langage hors contexte il existe un algorithme déterministe qui décide si un mot donné appartient à ce langage. Cet algorithme permet de fournir toutes les structures syntaxiques de ce mot.

Soit un langage hors contexte L et soit la grammaire de forme normale de Chomsky qui l'engendre.

Toutes les productions de cette grammaire ont la forme suivante :

$\Sigma \rightarrow \sigma \quad \Sigma \in V_N, \sigma \in V_t$: type 1.
$\Sigma \rightarrow \Sigma'\Sigma" \quad \Sigma, \Sigma', \Sigma" \in V_N$: type 2.

Si w est de longueur n l'algorithme consistera à construire une matrice carrée $n \times n$ dont chaque élément est une partie de V_N.

Propriété recherchée :

Un élément $M(i,j)$ de la matrice contiendra un symbole non terminal Σ si et seulement si :

$$\Sigma \underset{G}{\overset{\star}{\Rightarrow}} a_j a_{j+1} \ldots . a_{j+i}$$

Donc :

Un élément $M(0,j)$ contiendra un symbole Σ si et seulement si

$\Sigma \to a_j \in P$.

Un éléménť $M(i,j)$ contiendra le symbole Σ si et seulement si il existe un entier k tel que :

$$\Sigma \underset{G}{\Rightarrow} \Sigma_1 \Sigma_2 \underset{G}{\overset{\star}{\Rightarrow}} a_j a_{j+1} .. a_{j+k} a_{j+k+1} \ldots a_{j+i}$$

soit :

$$\Sigma_1 \underset{G}{\overset{\star}{\Rightarrow}} a_j a_{j+1} \ldots a_{j+k}$$

$$\Sigma_2 \underset{G}{\overset{\star}{\Rightarrow}} a_{j+k+1} \ldots a_{j+i}$$

Donc Un élément $M(i,j)$ contiendra un symbole Σ si et seulement si il existe un entier k tel que

$$\begin{array}{lrl} \Sigma_1 \in M(k,j) & & \\ \Sigma_2 \in M(l, j+k+1) & \text{tel que} & l+j+k+1 = j+i \\ & \text{soit} & l = i-k-1 \\ & \text{et} & \Sigma \to \Sigma_1 \Sigma_2 \in P. \end{array}$$

Le contenu d'un élément $M(i,j)$ de la matrice ne dépend donc que des éléments $M(k,j)$ tels que $k < i$ et $i+j < n$. Il est donc possible de construire cette matrice en calculant les éléments ligne à ligne. Une fois la matrice construite la propriété défine plus haut nous donne le test d'appartenance :

$w \in L(G)$ si et seulement si $S \underset{G}{\overset{\star}{\Rightarrow}} w$

$w \in L(G)$ si et seulement si $S \underset{G}{\overset{\star}{\Rightarrow}} a_0 \ldots . a_{n-1}$

$w \in L(G)$ si et seulement si $S \in M(n-1, 0)$

La matrice contient de plus toutes les structures syntaxiques possibles pour le mot w.
D'où l'algorithme :

$for(j = 0; j < n; ++j)$
$\{$
    $M(0,j) \ \ |= A \quad si\ A \to aj \in P.$
$\}$
$for(i = 1; i < n; ++i)$
$\{$

```
        for(j =0; j < n; ++j)
        {
            for(k =0; k < i; ++k)
            {
                M(i, j)  |= A   si B ∈ M(k, j)
                                   C ∈ M(i − k − 1, j + k + 1)
                                   A → BC ∈ P.
            }
        }
    }
```

Exemple :
Le langage $a^*b^nc^n \cup a^nb^nc^*$ est engendré par la grammaire :

$$
\begin{array}{ll}
S \rightarrow ZT & S \rightarrow XY \\
Z \rightarrow AZ & X \rightarrow AU \\
Z \rightarrow a & U \rightarrow XB \\
T \rightarrow BV & X \rightarrow AB \\
V \rightarrow TC & Y \rightarrow CY \\
T \rightarrow BC & Y \rightarrow c \\
A \rightarrow a & C \rightarrow c \\
B \rightarrow b &
\end{array}
$$

Matrice de reconnaissance de $a^3b^3c^3$:

S								
S	S							
S		S						
X			T					
	U			V				
	X			T				
Z		U			V	Y		
Z	Z	X			T	Y	Y	
A Z	A Z	A Z	B	B	B	C Y	C Y	C Y
a	a	a	b	b	b	c	c	c

Cet algorithme a une complexité de n^3. Dans des cas particuliers il est possible de construire des algorithmes plus performants notamment dans le cas des langages déterministes. Le support formel est alors un automate à pile déterministe qui peut être généré automatiquement à partir de la grammaire. Citons les générateurs YACC et BISON qui permettent de construire de tels analyseurs. L' utilisation de ces procédés, du fait du déterminisme imposé, convient difficilement au traitement des langues naturelles.

1.3.3 Analyse des langages de type 1.

Pour la reconnaissance des langages de type 1 on peut utiliser les algorithmes classiques en intelligence artificielle qui sont basés sur les moteurs d'inférences.

La présentation des moteurs d'inférences se fera ici que dans le cadre des grammaires sous contexte. Cette présentation permet néanmoins d'avoir une bonne appréhension de ces techniques.

Une grammaire sous contexte à la propriètè suivante :

> Si $w \to w' \in P$ alors $|w| \leq |w'|$

Une règle de grammaire sous contexte peut se voir comme une règle générale de déduction :

> Si la présence de w est attestée on peut déduire un nouveau mot (fait) en remplaçant w par w'.

La problématique de la reconnaissance d'un mot d'un langage donné est alors la suivante :

> étant donné un mot w ce mot est-il déductible du fait S avec l'ensemble des règles de la grammaire ?

Un moteur d'inférence est un algorithme qui permettra d'engendrer toutes les déductions possibles à partir des faits, c'est à dire ici des mots donnés. Chaque moteur se distingue par la stratégie qu'il emploie pour générer ces faits nouveaux.

Le cycle de base d'un moteur d'inférence comprend deux grandes parties : l'évaluation et l'exécution. L'évaluation permet de sélectionner les règles applicables et l'évaluation modifie l'état du système, c'est à dire la base de faits :
Cycle de base :

> Base de fait BF_1
> Base de règles BR_1
> $\Downarrow$
> évaluation $\Rightarrow$ Ensemble BR_2 sélectionné
> $\Downarrow$
> exécution $\Rightarrow$ modification de $BF_1 \Rightarrow BF_2$

Dans un moteur d'analyse d'un langage sous-contexte nous aurons :

> BF : ensembles des mots déductibles par la grammaire. cet ensemble peut être initialement réduit à S. Il peut également comporter un ensemble de mots fréquents par exemple.
>
> BR : ensemble des règles de grammaire.

Dans la recherche d'un mot il suffit d'appliquer le cycle de base jusqu'a ce que tous les mots ajoutés à la base de fait aient une longueur supérieure au mot recherché.
Exemple :

$$
\begin{aligned}
BF &= \{ && S, aaSBBcc, aabb, AAAAABBBBBCCCCC\} \\
\\
BR &= \{ && S \to ABC, S \to ASBC, CB \to BC, \\
& && A \to a, aB \to ab, bB \to bb, \\
& && bC \to bc, cC \to cc\}
\end{aligned}
$$

Recherche du mot $aaabbbccc$:

cycle 1 :

règles applicables : $S \to ABC$ $S \to ASBC$ $CB \to BC$ $A \to a$

mots nouveaux : ABC $ASBC$ $AAABCBBCC$ $aASBCBC$ $AAASBCBBCC$ $aAAAABBBBBCCCCC$

BF_1 : { S , ABC, $ASBC$, $AASBCBC$, $AAABCBCBC$, $AAASBCBCBC$, $aabb$, $aASBCBC$, $AAAAABBBBBCCCCC$, $aASBCBC$, $aBBBBBBBBBCCCCC$ }

cycle 2 :

règles applicables : $S \to ABC$, $S \to ASBC$, $CB \to BC$, $A \to a$

mots nouveaux : $AABCBC$ $AAAABCBCBCBC$ $AASCBBC$ $AAAASBCBCBCBC$ $AAABBCCBC$ $AAABCBBCC$ $AAASBBCBC$ $aASBBCC$

On parle de chaînage avant lorsque la recherche s'effectue à partir d'une position initiale jusqu'à une position finale. L'exemple précédent correspond donc à un chaînage avant.
On parle de chaînage arrière lorsque la recherche s'effectue à partir de position finale jusqu'a une position initiale connue.
Exemple de chaînage arrière dans la recherche de "aaabbbccc" avec les mêmes éléments :

> La base $BUTSF$ contient l'ensemble des buts. Elle contient les éléments contenus dans la base de fait précédente.
>
> La base BF se réduit alors au mot recherché "$aaabbbccc$".
>
> Les règles doivent s'appliquer de manière inverse.

cycle 1 :

règles applicables : $A \to a$ $B \to b$ $C \to c$

mots nouveaux : $Aaabbbccc$ $aaaBbbccc$ $aaabbbccC$

$BF1$: { $aaabbbccc$ $Aaabbbccc$ $aaaBbbccc$ $aaabbbccC$ }

cycle 2 :

règles applicables : $A \to a$ $B \to b$ $C \to c$

mots nouveau : $AAabbbccc$ $AaaBbbccc$ $AaabbbccC$ $aaaBBbccc$ $aaaBbbccC$ $aaabbbcCC$

$BF2$: { $aaabbbccc$ $Aaabbbccc$ $aaaBbbccc$ $aaabbbccC$ $AAabbbccc$ $AaaBbbccc$ $AaabbbccC$ $aaaBBbccc$ $aaaBbbccC$ $aaabbbcCC$ }

On parle de chaînage mixte lorsque les deux approches sont effectuées simultanément :

Dans le cas précédent on effectue un chainage avant sur la base de buts, un chaînage arrière sur la base de faits et le processus s'arrête lorsque l'intersection de la base de buts et la base de faits est non nulle :

cycle 1 :

règles applicables à la base de buts : $S \rightarrow ABC$ $S \rightarrow ASBC$
$CB \rightarrow BC$ $A \rightarrow a$

règles applicables à la base de faits : $A \rightarrow a$ $B \rightarrow b$ $C \rightarrow c$

mots nouveaux de la base de buts : ABC, $ASBC$, $AAABCBBCC$, $aAAAABBBBBCCCCC$, $aASBCBC$, $AAASBCBBCC$,

mots nouveaux de la base de faits : $Aaabbbccc$, $aaaBbbccc$, $aaabbbccC$

$BButs_1$: { S , ABC, $ASBC$, $AASBCBC$, $AAABCBCBC$, $aASBCBC$, $aabb$, $aASBCBC$, $AAAAABBBBBCCCCC$, $AAASBCBCBC$, $aBBBBBBBBBCCCCC$ }

BF_1 : { $aaabbbccc$, $Aaabbbccc$, $aaaBbbccc$, $aaabbbccC$ }

cycle 2 :

règles applicables à la base de buts : $S \rightarrow ABC$, $S \rightarrow ASBC$, $CB \rightarrow BC$, $A \rightarrow a$

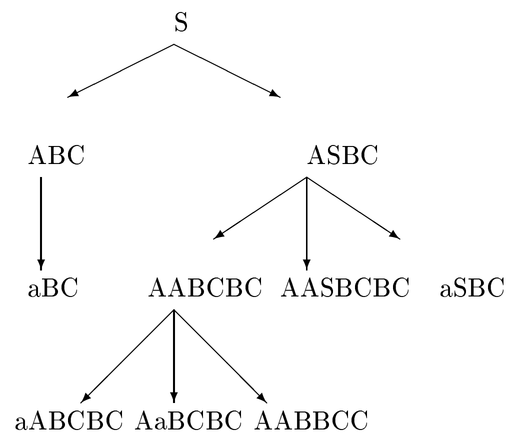
mots nouveauxde la base de buts : $AABCBC$, $AASCBBC$, $AAAABCBCBCBC$, $AAAASBCBCBCBC$, $AAABBCCBC$, $AAABCBBCC$, $AAASBBCBC$, $aASBBCC$

mots nouveaux de la base de faits : $AAabbbccc$, $AaaBbbccc$, $AaabbbccC$, $aaaBBbccc$, $aaaBbbccC$, $aaabbbcCC$,

$BButs_2$: { S , ABC, $ASBC$, $AASBCBC$, $AAABCBCBC$, $AAASBCBCBC$, $aASBCBC$, $AAAAABBBBBCCCCC$, $aabb$, $aASBCBC$, $aBBBBBBBBBCCCCC$ }

BF_2 : { $aaabbbccc$, $Aaabbbccc$, $aaaBbbccc$, $AAabbbccc$, $AaaBbbccc$, $aaaBBbccc$, $aaabbbccC$, $aaaBbbccC$, $AaabbbccC$, $aaabbbcCC$ }

Dans le fonctionnement de ces moteurs d'inférences toutes les règles applicables à un cycle donné le sont effectivement. Il s'agit d'une recherche en largeur de la solution. Ce terme signifie que l'arborsecence des choix associée à la recherche d'une solution est construite par niveaux successifs :



Le moteur peut fonctionner en profondeur. Dans ce cas à chaque cycle une seule règle est sélectionnée. Il est alors nécessaire d'avoir un dispositif de retour arrière pour pouvoir parcourir l'arborescence des choix. Cette arborescence est parcourue de façon canonique avec comme cycle récursif de base :

- l'ensemble des descendants d'un point est parcouru en énumérant un à un chaque descendant et ses dépendants.

Sur l'exemple de la reconnaissance du mot "aaabbbccc" on remarque aisément que ces techniques sont difficilement applicables à la reconnaissance des langues naturelles. Le processus serait dailleurs alourdi par le fait que l'application d'une règle devrait s'accompagner d'une construction syntaxique associée. Néanmoins tous les analyseurs de langues naturelles utilisent ces techniques dans un cadre plus restreint afin d'éviter une explosion combinatoire. La problématique de l'analyse des langues naturelles devient : comment augmenter l'aspect descriptif des langages hors contextes sans accéder complètement au cadre des langages sous contexte.

Chapitre 2

Formalismes syntaxiques.

2.1 Grammaire structurelle

La grammaire structurelle est un outil de manipulations de structures. Cet outil permet de définir un cadre de description opératoire au traitement automatique des langues naturelles. La grammaire est basée sur la définition de transformation de structures. Ces transformations définissent des algorithmes de Markov dans l'espace des éléments manipulables qui sont appelés éléments structurés.

2.1.1 Eléments structurés.

Un élément structuré est construit à partir de trois éléments :
- Un ensemble d'étiquettes
- Un ensemble fini d'arborescences ordonnées.
- Une fonction d'étiquetage.

2.1.1.1 Etiquette.

Une étiquette correspond à une structure de traits de PATR-II : Une étiquette est définie par un ensemble de variables affectées de valeurs. La coindexation doit ici être explicitement définie par des variables "référence".

Exemple :

cat(SN,SV,P).

nombre(singulier,pluriel).

personne(1,2,3).

Etiquette : { cat(SN),nombre(singulier),personne(2) }.

Chaque variable a une valeur vide (non affectée) définie par défaut. L'ensemble des variables pouvant être présentes dans une étiquette définit l'univers des éléments structurés.

2.1.1.2 Arborescence.

Une arborescence est une structure neutre. Un élément structuré contient un ensemble fini d'arborescences. Une arborescence peut être définie par un graphe orienté ayant les propriétés suivantes :

- Il existe un seul point nommé racine qui n'a pas d'antécédant.
- Tout point différent de la racine a un et un seul antécédant.
- L'ensemble des descendants d'un point est totalement ordonné.

2.1.1.3 Fonction d'étiquetage.

La fonction d'étiquetage est une application qui associe à chaque point des différentes arborescences une étiquette. Cette application n'a pas de contraintes particulières, notamment elle n'est pas nécessairement injective. La fonction Et définit l'ensemble des points d'un ensemble d'arborescences :

Si A est une arborescence ayant un seul point x alors $Et(A) = \{x\}$

Si A est une arborescence de racine x ayant comme descendants les points x_1,...,x_n et si A_i est la sous-arborescence maximale de racine x_i alors $Et(A) = \{x\}|Et(A_1)|...|Et(A_n)$

Si A est un ensemble contenant une seule arborescence A_r alors $Et(A_r) = Et(A)$.

Si A et B sont deux ensembles d'arborescences alors $Et(A|B) = Et(A)|Et(B)$.

Si A est un ensemble d'arborescence la fonction d'étiquetage est une application de $Et(A)$ dans E.

2.1.1.4 Element structuré.

Un élément structuré est un triplet $< \mathcal{A}, \mathcal{E}, f >$ où :

- $\mathcal{A}$ est un ensemble fini d'arborescences ordonnées.
- $\mathcal{E}$ est un ensemble d'étiquettes.
- f est une fonction d'étiquetage : $Et(\mathcal{A}) \rightarrow \mathcal{E}$ qui associe à chaque point de $Et(\mathcal{A})$ une étiquette de $\mathcal{E}$.

Exemple :

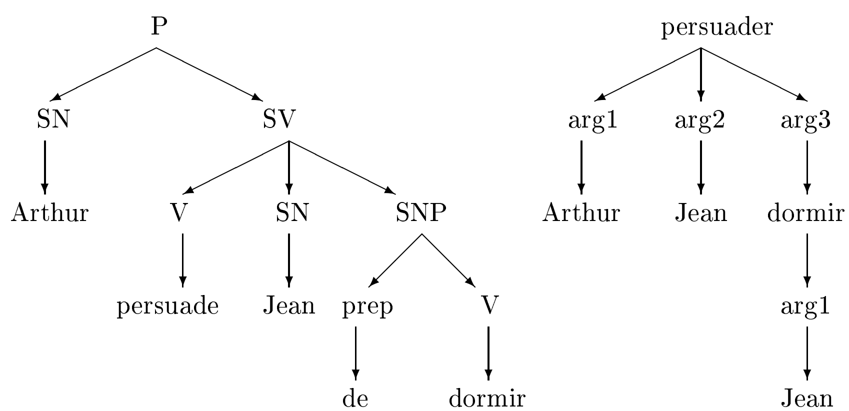
$$\mathcal{A} = \left\{ \begin{tikzcd} & & 1 \arrow[dl] \arrow[dr] & \\ & 2 \arrow[dl] \arrow[dr] & & 3 \\ 4 & & 5 & \end{tikzcd} \right\}$$

$$
\begin{array}{rrl}
\mathcal{E} = \{ & E_1 & : \text{cat(SV),Atête}(E_{I_1})\text{,Asouscat}(E_{I_2}). \\
& E_{I_1}: & \text{forme(finie).} \\
& E_{I_2}: & \text{premier(tête(cat(SN),accord(nombre(sing),} \\
& & \text{personne(3ème))), reste(fin).} \\
& E_2: & \text{cat(SV) ;Atête}(E_{I_1})\text{ ;Asoucat}(E_{I_3}). \\
& E_{I_3}: & \text{Apremier}(E_3)\text{ ; tête(forme(infinitif)) ;} \\
& & \text{souscat}(E_{I_5}). \\
& E_4: & \text{cat(SV) ;Atête}(E_{I_1})\text{ ;Asoucat}(E_{I_4}). \\
& E_{I_4}: & \text{Apremier}(E_5)\text{ ;Areste}(E_{I_3}). \\
& E_5: & \text{cat(SN) ; tête(accord(nombre(sing),} \\
& & \text{personne(3ème))).} \\
& E_{I_5}: & \text{premier}(E_5)\text{ ;reste(fin). } \}
\end{array}
$$

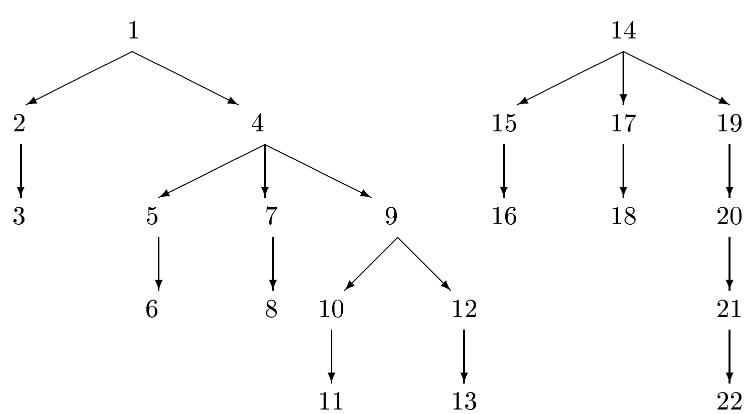
$$f : 1 \to E_1,\ 2 \to E_2,\ 3 \to E_3,\ 4 \to E_4,\ 5 \to E_5.$$

La structure est alors identique à celle obtenue par la grammaire d'unification avec la phrase "persuade Arthur de dormir".

Avec une deuxième structure on peut aisément définir la structure syntaxico-sémantique :



La structure syntaxique comprend donc deux arborescences :



$\mathcal{E}$:	E_1 : cat(P).	E_{12} : cat(V).	f :	$1 \rightarrow E_1$	$12 \rightarrow E_{12}$
	E_2 : cat(SN).	E_{13} : UL(dormir).		$2 \rightarrow E_2$	$13 \rightarrow E_{13}$
	E_3 : UL(Arthur).	E_{14} : typ(arg1).		$3 \rightarrow E_3$	$14 \rightarrow E_6$
	E_4 : cat(SV).	E_{15} : typ(arg2).		$4 \rightarrow E_4$	$15 \rightarrow E_{14}$
	E_5 : cat(V).	E_{16} : typ(arg3).		$5 \rightarrow E_5$	$16 \rightarrow E_3$
	E_6 : UL(persuader).	E_{17} : typ(arg1).		$6 \rightarrow E_6$	$17 \rightarrow E_{15}$
	E_7 : cat(SN).			$7 \rightarrow E_7$	$18 \rightarrow E_8$
	E_8 : UL(Jean).			$8 \rightarrow E_8$	$19 \rightarrow E_{16}$
	E_9 : cat(SVP).			$9 \rightarrow E_9$	$20 \rightarrow E_{13}$
	E_{10} : cat(PREP).			$10 \rightarrow E_{10}$	$21 \rightarrow E_{17}$
	E_{11} : UL(de).			$11 \rightarrow E_{11}$	$22 \rightarrow E_8$

Les deux structures sont assoiciées par leurs étiquettes. Les étiquettes associées aux points 8, 18 et 22 par exemple sont identiques.

On appelle dimension le numéro d'ordre d'une arborescence. Nous aurons ainsi une arborescence dans la dimension 1 ou dimension syntaxique, une deuxième arborescence dans la dimension 2 ou dimension logique, etc....

2.1.2 Règles de transformation.

Les règles de transformations permettent de modifier un élément structuré. Elles permettront de définir un algorithme de Markov. Ces modifications de structure s'opèrent sur des éléments structurés de manière similaire aux transformations de chaînes. Les coupes de la grammaire transformationnelle deviennent alors des cas particuliers de ces transformations.

2.1.2.1 Sous-arborescence.

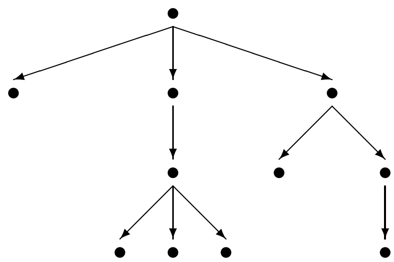
Une arborescence A est une sous-arborescence d'une arborescence B si il est possible d'identifier un ensemble de points de B dont les relations dans B sont compatibles avec celles de A.

Définition (ordonnée) :

- Tout point A est une sous-arborescence d'une arborescence non vide B.
- Une arborescence de racine X ayant comme descendants de X les arborescences X_1,...,X_n tel que $\forall i$ X_i précède X_{i+1} est une sous-arborescence de racine Y d'une arborescence B si il existe un point Y de B ayant comme descendants les arborescences Y_1,...,Y_m tel que $\forall j$ Y_j précède Y_{j+1} et tel que :
  - $\forall i$ X_i est une sous arborescence de racine Y_{j_i}.
  - $\forall k, l : k < l \Rightarrow j_k < j_l$.

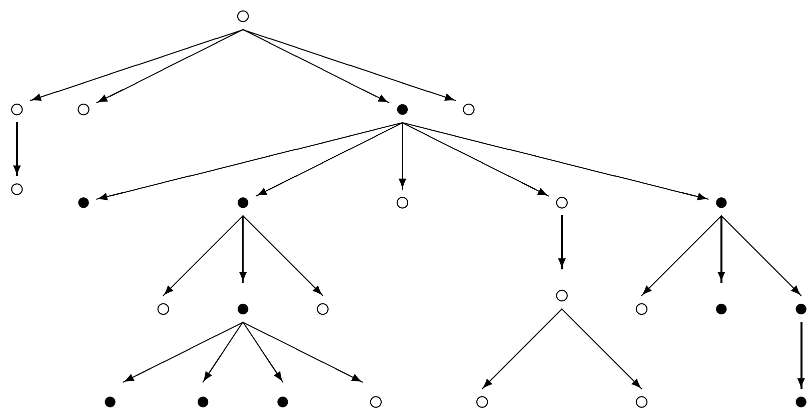
Exemple :

$$0(1, 2(3(4(5, 6, 7))), 8(9, 10(11)))$$



est une sous arborescence de

$0(1(2), 3, 4(5, 6(7, 8(9, 10, 11, 12), 13), 14, 15(16(17, 18)), 19(20, 21, 22(23))), 24)$



2.1.2.2 Schéma d'arborescences.

Un schéma d'arborescence permet de définir un ensemble de sous-arborescences. Les extensions de la définition portent sur les points suivants :

- Ordre : Les descendants d'un point ne sont plus forcément ordonnés. La condition sur les éléments Y_{i_j} devient :

  $\forall k, l : k \neq l \Rightarrow Y_{j_k} \neq Y_{j_l}$.

- Dépendance généralisée : La relation de dépendance est fermée transitivement. La condition sur X_i devient :

  $\forall i$ X_i est une sous arborescence de Y_{j_i}.

- Partie facultative : Une partie des X_i peut être absente. Si X_i est défini comme facultatif, la condition devient :

  $\forall i : X_i$ facultatif $\Rightarrow$
  - Soit X_i est une sous-arborecence de Y_{j_i}.
  - soit X_i n'est pas une sous-arborescence de Y_{j_i} et X_i n'appartient pas à la sous arborescence reconnue.

- Continuité : Deux points X_i et X_{i+1} peuvent être obligatoirement contigüs dans leurs réalisations dans B :

  Si X_i et X_{i+1} sont déclarés contigüs et si X_i est une sous-arborescence de Y_{j_i}, X_{i+1} est une sous-arborescence de Y_{j_k} alors

$j_k = j_i + 1$.

Cette contrainte de continuité impose l'ordre des éléments X_i et X_{i+1}.

Notation d'un schéma :

-	(et)	définissent la dépendance.
-	?	définit la dépendance généralisée.
-	%	définit le point comme facultatif.
-	,	définit l'ordre entre deux points.
-	-	sépare deux points non ordonnés.
-	;	sépare deux groupes de points de façon ordonnée.
-	$*$	impose la continuité.

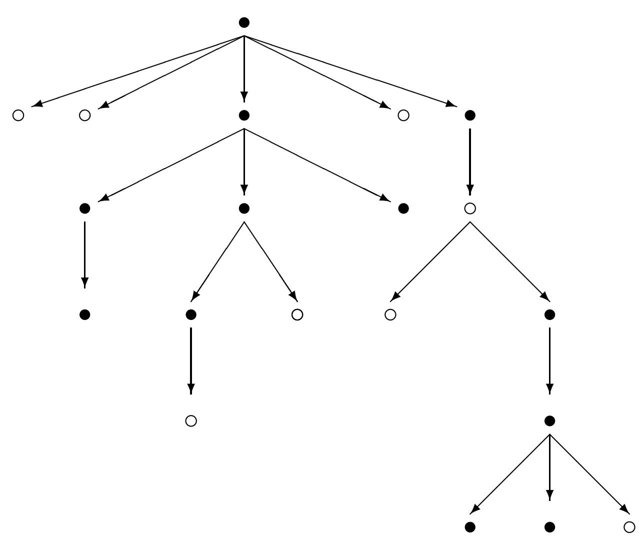
Exemple de schéma :

Le schéma

1(2 ?(3(4(5,6)))-7(8(9) ;10-11(12)))

est présent dans l'arborescence

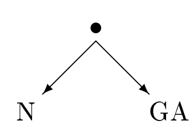
1(2,3,4(5(6),7(8(9),10)),11),12,13(14(15,16(17(18,19,20)))))



Un schéma possède trois éléments :

– Une partie structurelle définissant la géométrie des parties à transformer.
– Un ensemble de conditions portant sur les étiquettes associées à un point. Ces conditions sont appelées conditions propres.
– Une condition portant sur les étiquettes associées à un ensemble de points. Cette condition est appelée condition inter-sommet.

Exemple :



0(1,2) /	1 : cat = N ;	2 : cat = GA /	(GNR(1) = GNR(2)) & (NBR(1) = NBR(2)).
	identification du Nom condition propre au point 1	identification du Groupe Adjectival	condition d'accord portant sur les étiquettes 1 et 2

Schéma structurel :

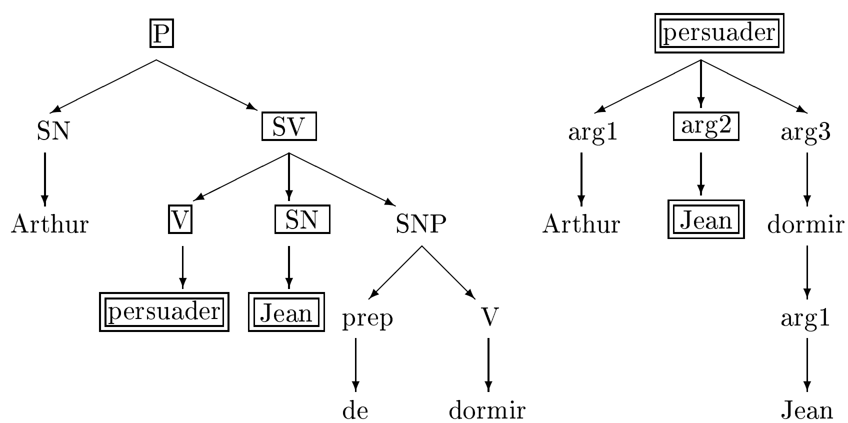
Un schéma structurel est un ensemble fini de schémas portant chacun sur une arborescence de l'élément structuré. Ces schémas précisent la dimension de l'arborescence dans lequel ils doivent être recherchés.

Lorsqu'un point d'un schéma est identifié par le même nom qu'un point d'un autre schéma l'étiquette associée à ces deux points doit être la même.

Exemple :

Dans la phrase "Arthur persuade Jean de dormir".

[1] : P(SV(V(persuader),SN(Jean)))	identifie 'persuader' et 'Jean'
[2] : persuader(arg2(Jean))	par identification de nom.

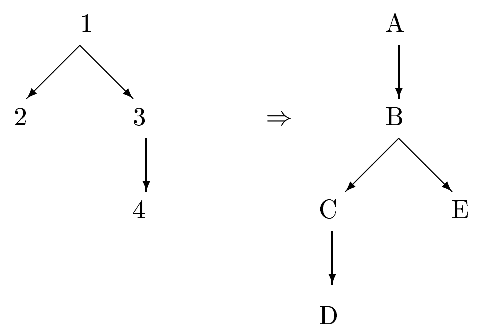


2.1.2.3 Transformation.

Une transformation d'élément structuré est définie à partir des transformations d'arborescences. Une transformation d'arborescence est définie par deux arborescences, une fonction d'application et une relation d'ordre. La première arborescence correspond à un schéma d'arborescence et détermine la condition d'applicabilité de la règle. La deuxième arborescence définit la transformation du schéma reconnu. Cette transformation pour être complètement définie fait appel aux deux derniers éléments de la transformation : la fonction d'application et la relation d'ordre. Ces deux derniers éléments précisent où seront placés les éléments de la structure transformée qui n'appartiennent pas au schéma reconnu.

Exemple :

1(2,3(4)) $\Rightarrow$ A(B(C(D),E)).



$1 <, 2 > \rightarrow A <, B >$

$1 < 2, 3 > \rightarrow B < C, E >$

$1 < 3, > \rightarrow A < B, >$

$2 <, > \rightarrow D <, >$

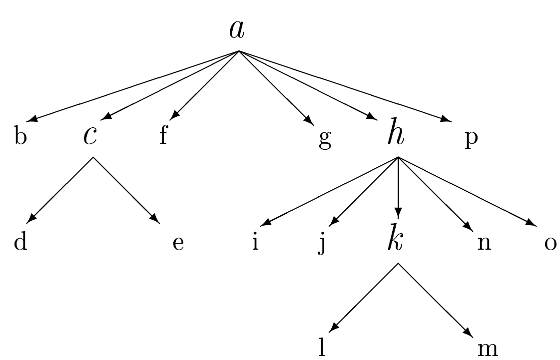
$3 <, 4 > \rightarrow B < E, >$

$3 < 4, > \rightarrow B < E, >$

$4 <, > \rightarrow E <, >$

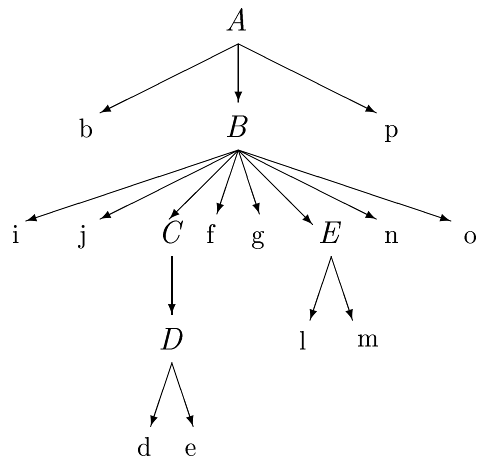
L'arborecence

a(b,c(d,e),f,g,f(i,j,k(l,m),n,o),p)



est transformée en supposant que le schéma soit reconnu sur a(c,h(k)) et donne l'arborescence :

A(b,B(i,j,C(D(d,l)),f,g,E(l,m),n,o),p).



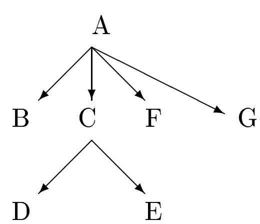
Les noms qui sont donnés ici sont bien sûr arbitraires. Dans le résultat les noms donnés aux points permettent d'identifier le schéma de transformation.

2.1.2.4 Définition d'une liste :

Une liste dans une arborescence A est définie par un triplet $X < Y, Z >$ où :

- X appartient à A.
- Y est absent ou Y appartient à A et Y est descendant direct de X.
- Z est absent ou Z appartient à A, Z est descendant direct de X et placé à droite de Y si il existe.

Exemple :



Les listes possibles sont :

$A <, >$
$A <, B >$
$A < B, >$
$A < B, F >$
etc...

Les éléments suivants ne sont pas des listes :

$A < D, G >$
$C < E, D >$

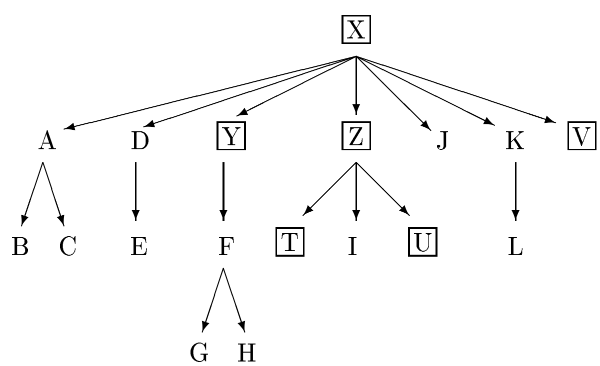
Les listes sont définies en même temps que le schéma d'arborescence. Il n'est donc pas nécessaire de les écrire explicitement dans un schéma. Lorsqu'un schéma d'arborescence est identifié dans une arborescence, les listes correspondent à des parties de cette arborescence qui ne sont pas dans le schéma. Il est donc nécessaire de définir à quelle partie correspondent ces listes. Cette définition correspond à l'actualisation d'une liste dans une arborescence donnée ou un schéma à été identifié.

2.1.2.5 Actualisation d'une liste dans une arborescence par rapport à une sous-arborescence :

Soit A une sous-arborescence d'une arborescence B et soit une liste $X < Y, Z >$ de A. L'actualisation de la liste $X < Y, Z >$ dans B est l'ensemble des arborescences B_1,...,B_n tel que :

- L'arborescence ayant pour racine X et pour descendants immédiats les arborescences B_1,...,B_n est une sous arborescence de B de racine X.
- Aucun point de A n'appartient à une arborecence B_i.
- Y est à gauche de la racine de B_1 dans B.
- Z est à droite de la racine de B_n dans B.
- L'ensemble B_1,...,B_n est l'ensemble maximum vérifiant les conditions précédentes.

Exemple :



$A :$	$X(Y, Z(T, U), V)$
$B :$	$X(A(B, C), D(E), Y(F(G, H)), Z(T, I, U), J, K(L), V)$
$X <, > :$	$A(B, C)\ D(E)\ J\ K(L)$
$Z < T, U > :$	I
$X < Z, V > :$	$J\ K(L)$
$X < Y, Z > :$	vide

2.1.2.6 Transformation d'arborescence :

Une transformation est définie par un quadruplet (A, B, f, O) où :

- A et B sont deux arborescences.
- f est une fonctions de l'ensemble des listes de A dans l'ensemble des listes de B
- O est une relation d'ordre partiel sur l'ensemble des listes de A tel que :

  Si x et y sont deux listes de A tel que $f(x) = f(y)$ alors $O(x, y)$ est défini.

Application d'une transformation :

Une transformation modifie une arborescence X et donne une nouvelle arborescence X'. Pour définir le transformé il est nécessaire de définir $S(X, A)$: On appelle $S(X, A)$ l'arborescence obtenue en supprimant dans X le point A et tous ces descendants.

Une arborescence X' est le transformé d'une arborescence X par la transformation (A, B, f, O) si et seulement si :

- A est une sous arborescence de X de racine x. (A à une image dans X).
- B est une sous arborescence de X' de racine x'. (B à une image dans X').
- $S(A, x) = S(B, x')$ et x et x' ont les mêmes voisins. (la partie sommet est indentique).
- Pour toute liste $T < U, V >$ de A :

  Si $f(T < U, V >) = T' < U', V' >$ alors l'actualisation de $T < U, V >$ dans X appartient à l'actualisation de T$' < U', V' >$ dans X'.

- Pour tout couple de listes $T < U, V >$ et $T_1 < U_1, V_1 >$ de A :

  Si $f(T < U, V >) = f(T_1 < U_1, V_1 >)$ et $T < U, V > \; < T_1 < U_1, V_1 >$ (Ordre O) alors l'actualisation de $T < U, V >$ dans X qui se trouve être une partie de l'actualisation de $f(T < U, V >) = T' < U', V' >$ dans X' se trouve à gauche de l'actualisation de $T_1 < U_1, V_1 >$ dans X qui est une autre partie de l'actualisation de $T' < U', V' >$ dans X'. Autrement dit l'actualisation dans X' respecte l'ordre défini par O.

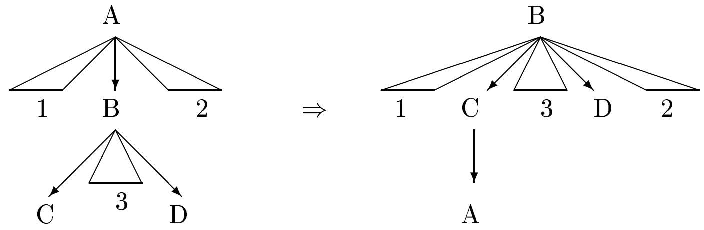
- Pour toute liste $T' < U', V' >$ de B l'actualisation de $T' < U', V' >$ dans X' est la réunion de l'actualisation des listes $T_1 < U_1, V_1 >$,...,$T_n < U_n, V_n >$ de A dans X tel que $\forall i f(T_i < U_i, V_i >) = T' < U', V' >$.

Dans la grammaire structurelle :

le schéma de reconnaissance définit l'arborescence A ;
le schéma de transformation définit les autres éléments : l'arborescence B, la fonction de transfert, l'ordre sur les listes.

Exemple :

$$\mathrm{A(B(C,D))} \Rightarrow \mathrm{B(*A<,B>*,C(A),*B<C,D>*,D,*A<B,>*)}$$



$1 : A <, B >$

$2 : A < B, >$

$3 : B < C, D >$

Les listes non définies sont perdues (par exemple $B <, C >$)

A(E,F,B(G(L),C(H),I,D,J),K) $\Rightarrow$ B(E,F,C(A),I,D,K).

points perdus : G,H,L,J

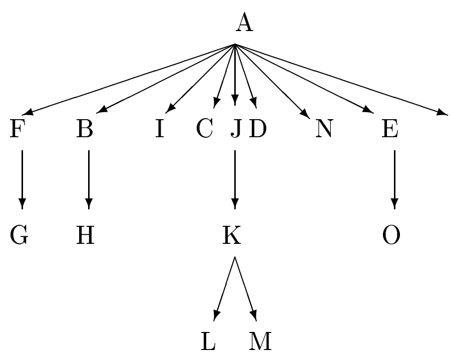
2.1.2.7 Extensions :

- Une liste peut apparaître plusieurs fois et dans ce cas les points concernés sont dupliques.
- Lorsque deux listes se chevauchent les points communs de l'intersection sont dupliqués.

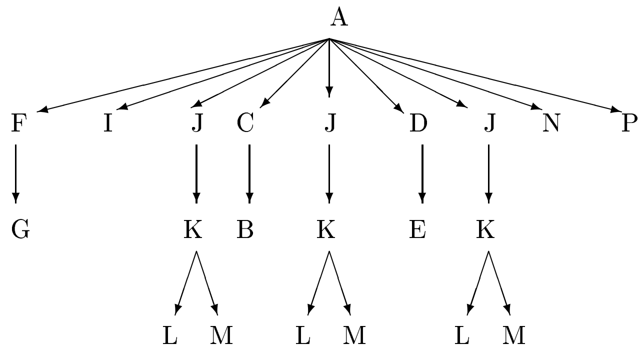
Exemple :

$\mathrm{A(B,C,D,E)} \Rightarrow \mathrm{A}(*\mathrm{A}<,\mathrm{D}>*,\mathrm{C(B)},*\mathrm{A}<\mathrm{C},\mathrm{D}>*,\mathrm{D(E)},*\mathrm{A}<\mathrm{C},>*)$

La liste $*\mathrm{A}<\mathrm{C},\mathrm{D}>*$ est dupliquée trois fois car elle est définie explicitement et elle apartient aux listes $*\mathrm{A}<,\mathrm{D}>*$ et $*\mathrm{A}<\mathrm{C},>*$.



A(F(G),B(H),I,C,J(K(L,M)),D,N,E(O),P) $\Rightarrow$



A(F(G)I,I(K(L,M)),C(B),J(K(L,M)),D(E),J(K(L,M)),N,P)

2.1.2.8 Transformations d'éléments structurés :

Une transformation d'éléments structurés correspond à une transformation simultanée sur les arborescences qui composent le schéma. Une transformation d'élément structuré est donc définie par un ensemble fini de transformations qui s'appliquent chacune dans une dimension.

> [1] : schéma [2] : schéma ... [n] : schéma $\Rightarrow$ [1] : résultat [2] : résultat .... [n] : résultat.

Le résultat d'une transformation comprend une définition de modification d'étiquettes. De façon symétrique à la définition d'un schéma une tranformation possède trois éléments :

- Une partie structurelle définissant la géométrie des parties transformées.
- Un ensemble d'affectations portant chacune sur une étiquette associée à un point du schéma de transformation.
- Une suite d'affectations portant sur les étiquettes associées aux points du schéma de transformation. Dans cette affectation les valeurs des étiquettes associées aux points du schéma de transformation tiennent compte des modifications précédentes.

Pour le traitement des étiquettes dans les schémas de reconnaissance ou de transformation les valeurs de variables sont définies de plusieurs façons :

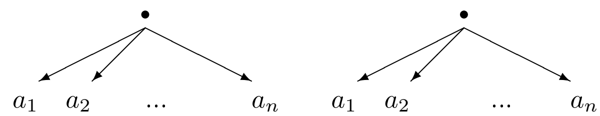
- Une valeur peut être issue d'une étiquette associée à un point d'un schéma.
- Une valeur peut être définie comme le résultat d'une fonction appliquée à la valeur d'une étiquette associée à un point d'un schéma.
- Une valeur peut être définie à partir d'une étiquette nommée. Cette étiquette sera accessible dans toute l'application de la grammaire. Ces étiquettes nommées correspondent à des registres d'étiquettes.

Exemple :

Soit n éléments ordonnés. Il s'agit de construire une arborescence binaire tel que le ième élément ait pour descendants les éléments de rang 2i et 2i+1. La forme initiale des éléments est simplement leurs listes ordonnées mise dans une arborescence plate :

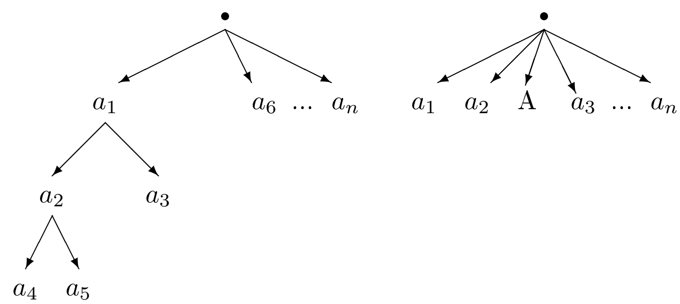
> I(a_1,a_2,....,a_n).

Cette arborescence est projetée dans deux dimensions. Nous auront donc au départ l'élément structuré suivant :



[1] : I(a_1,a_2,....,a_n). [2] : I(a_1,a_2,....,a_n).

Le point A n'appartient pas à la liste a_1,...,a_n. L'étape intermédiaire de construction de cette arborescence binaire dans la dimension 1 sera définie par l'élément structuré suivant :



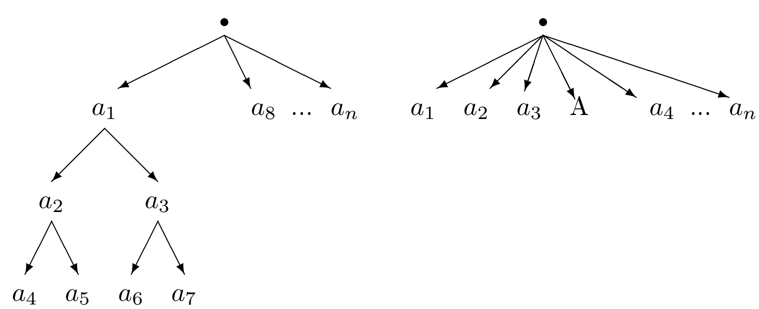
[1] : I(a_1(a_2(a_4,a_5),a_3),a_6,a_7,...,a_n) [2] : I(a_1,a_2,A,a_3,a_4,....,a_n)

La présence du point A signifie qu'il faut placer les futurs descendants sous le point a_3. La règle de transformation est alors la suivante :

[1] : 0($*$,1 ?(2),$*$,3,$*$,4) $\Rightarrow$ 0(1(2(3,4))).

[2] : 0(A,2) $\Rightarrow$ 0(2,A).

Après l'application de cette transformation la structure précédente devient :



Le point 2 ayant le même nom dans les deux schéma doit avoir la même étiquette associée dans les deux dimensions : Il permet donc de passer de l'adressage linéaire 0(A,2) à l'adressage arborescent : 1 ?(2).

2.1.3 Grammaire élémentaire : Algorithme de Markov.

Un ensemble de règles d'une grammaire structurelle est regroupé pour former un algorithme de Markov sur les éléments structurés. Cette grammaire élémentaire permettra donc une définition opérationnelle d'une analyse de langue naturelle.

2.1.3.1 Définition des algorithmes de Markov.

La définition des algorithmes de Markov date de 1954 ; Une présentation détaillée des algorithmes de Markov peut être trouvée dans " Introduction do mathematical logic. E Mendelson.".

> Soit V un vocabulaire fini et V^* le monoïde libre engendré par V et l'opération de concaténation.
>
> Un algorithme de Markov définit une fonction effectivement calculable d'un sous ensemble de V^* et à valeur dans V^*
>
> Soit w un mot de V^* et $\mathcal{U}$ un algorithme. $\mathcal{U}$ est dit applicable à w si w est dans le domaine de $\mathcal{U}$. Si w est applicable à $\mathcal{U}$, $\mathcal{U}(w)$ est la valeur de cet algortihme.
>
> Un algorithme sur un alphabet V est un algorithme sur tout alphabet V' tel que $V \subseteq V'$. On suppose que les signes '.' et '$\rightarrow$' n'appartiennent à aucun vocabulaire.
>
> Si w et w' sont deux mots de V^* alors :
>
> $w \rightarrow w'$ est une production simple.
>
> $w \rightarrow .w'$ est une production terminale.
>
> Un algorithme de Markov est une liste finie ordonnée de productions.

$$
\begin{array}{ccc}
w_1 & \rightarrow & (.)w'_1 \\
w_2 & \rightarrow & (.)w'_2 \\
 & ..... & \\
 & ..... & \\
w_n & \rightarrow & (.)w'_n
\end{array}
$$

> Un mot x est un sous mot d'un mot w si et seulement si :
>
> $\exists y, z \in V^*$ tel que $w = yxz$

Fonctionnement de l'algorithme :

1. $\mathcal{U} : w \sqsupseteq$ si aucune partie gauche w_i des productions de l'algorithme n'est un sous-mot de w.
2. Soit m le plus petit entier tel que w_m soit un sous-mot de w et soit w' le mot obtenu en remplaçant l'occurrence la plus à gauche de w_m dans w.

   $\mathcal{U} : w \vdash w'$ si $w_m \rightarrow w'_m$ est une production simple.

   $\mathcal{U} : w \vdash .w'$ Si $w_m \rightarrow .w'_m$ est une production terminale.

3. $w \models (.)w'$ si il existe une séquence finie w_1,...,w_n tel que :
   - $w = w_1$
   - $w' = w_n$
   - $\forall i, 1 \leq i \leq n-2$: U : $w_i \vdash w_{i+1}$
   - Soit $\mathcal{U} : w_{n-1} \vdash w_n$ On écrira alors $\mathcal{U} : w \models w'$
     Soit $\mathcal{U} : w_{n-1} \vdash .w_n$ On écrira $\mathcal{U} : w \models .w'$
4. $\mathcal{U}(w) = w'$ si et seulement si :
   - Soit $\mathcal{U} : w \models .w'$.
   - soit $\mathcal{U} : w \models w'$ et $\mathcal{U} : w' \sqsupseteq$.

Un algorithme de Markov est donc un système de réécriture sur des mots. L'arrêt de l'algorithme correspond à l'application d'une règle terminale ou à l'obtention d'un mot auquel aucune règle ne peut être applicable.

Exemple :

Soit V un alphabet fini et $\alpha, \beta \notin V$.

L'algorithme suivant est défini sur $V' = V \cup \{\alpha, \beta\}$.

1. $\alpha\alpha \to \beta$
2. $\beta\sigma \to \sigma\beta \quad \forall \sigma \in V$
3. $\beta\alpha \to \beta$
4. $\beta \to .\varepsilon \quad \varepsilon$ est le mot vide
5. $\alpha\sigma\sigma' \to \sigma'\alpha\sigma \quad \forall \sigma, \sigma' \in V$
6. $\varepsilon \to \alpha$

Alors $\forall w \in V^* \quad \mathcal{U}(w) = \widetilde{w}$.

$$123 \vdash \alpha 123 \vdash 2\alpha 13 \vdash 23\alpha 1 \vdash \alpha 23\alpha 1 \vdash 3\alpha 2\alpha 1 \vdash \alpha 3\alpha 2\alpha 1 \vdash \alpha\alpha 3\alpha 2\alpha 1$$
$$\vdash \beta 3\alpha 2\alpha 1 \vdash 3\beta\alpha 2\alpha 1 \vdash 3\beta 2\alpha 1 \vdash 32\beta\alpha 1 \vdash 32\beta 1 \vdash 321\beta \vdash .321$$

2.1.3.2 Equivalences :

Soit $\mathcal{U}$ et $\mathcal{U}'$ deux algorithmes et w un mot de V^* :

$\mathcal{U}(w) \approx \mathcal{U}'(w)$ si et seulement si :
- Soit $\mathcal{U}(w) = \mathcal{U}'(w)$
- Soit $\mathcal{U}$ et $\mathcal{U}'$ sont tous les deux non applicables à w.

Deux algorithmes $\mathcal{U}$ et $\mathcal{U}'$ sont dits pleinement équivalents si et seulement si

$$\forall w \in V^* : \mathcal{U}(w) \approx \mathcal{U}'(w).$$

$\mathcal{U}$ et $\mathcal{U}'$ sont dits équivalent relativement à V si et seulement si :

$\forall w \in V^*$: Si $\mathcal{U}(w)$ ou $\mathcal{U}(w')$ existe alors $\mathcal{U}(w) \approx \mathcal{U}'(w)$.

Un algorithme est dit fermé si et seulement si il possède une prodution terminale de la forme :

$$\epsilon \to .Q.$$

Un algorithme peut toujours avoir un équivalent fermé : Il suffit d'ajouter en fin de liste la production :

$$\epsilon \to .\epsilon$$

Un algorithme fermé se termine toujours par application d'une règle terminale.

2.1.3.3 Composition des algorithmes :

Soit $\mathcal{U}$ et $\mathcal{U}'$ deux algorithmes sur un vocabulaire V. Pour chaque symbole σ de V on définit un symbole Σ_σ appelé correspondant de σ. Soit alors l'alphabet V' :

$$V' = V \cup \{\Sigma_\sigma | \sigma \in V\}$$

Soit $\alpha, \beta \notin V'$.

Soit $\mathcal{S}_\mathcal{U}$ le schéma de $\mathcal{U}$ ou l'on a remplacé le '.' par 'α'. $\mathcal{S}_{\mathcal{U}'}$ le schéma de $\mathcal{U}'$ où l'on a remplacé chaque symbole par son correspondant, le symbole '.' par 'β', les productions de la forme $\epsilon \rightarrow w$ par $\alpha \rightarrow \alpha w$, et les productions de la forme $\epsilon \rightarrow .w$ par $\alpha \rightarrow \alpha\beta w$.

Alors le schéma suivant est dit être le composé des algorithmes $\mathcal{U}$ et $\mathcal{U}'$:

$$\begin{array}{rl}
\sigma\alpha \rightarrow \alpha\sigma & \forall\sigma \in V \\
\alpha\sigma \rightarrow \alpha\Sigma_\sigma & \forall\sigma \in V \\
\Sigma_\sigma\sigma' \rightarrow \Sigma_\sigma\Sigma_{\sigma'} & \forall\sigma, \sigma' \in V \\
\Sigma_\sigma\beta \rightarrow \beta\Sigma_\sigma & \forall\sigma \in V \\
\beta\Sigma_\sigma \rightarrow \beta\sigma & \forall\sigma \in V \\
\sigma\Sigma_{\sigma'} \rightarrow \sigma\sigma' & \forall\sigma, \sigma' \in V \\
\alpha\beta \rightarrow .\varepsilon & \\
\mathcal{S}_{\mathcal{U}'} & \\
\mathcal{S}_\mathcal{U} &
\end{array}$$

Si l'on désigne par $\mathcal{U}$" cet algorithme alors :

$$\mathcal{U}"(w) = \mathcal{U}'(\mathcal{U}(w)) \quad \forall w \in V^*$$

On note alors : $\mathcal{U}" = \mathcal{U}' \circ \mathcal{U}$.

La composition fonctionne ainsi : Le seul algorithme applicable au départ est l'algorithme $\mathcal{U}$ puisqu'il est le seul à fonctionner sur V. Il se termine en produisant le symbole 'α'. A partir de cet instant les trois premières règles sont applicables. Elles permettent de placer 'α' à gauche du mot et de transformer tous les symboles de V en leurs correspondants. Une fois ces règles non applicables l'algorithme $\mathcal{S}_{\mathcal{U}'}$ est applicable et se termine en produisant un symbole 'β'. Les règles 4 à 7 deviennent alors applicables. Elles permettent de retransformer les symboles correspondants en leurs symboles initiaux et de déplacer le symbole 'β' sur la gauche. Quand ces règles ne sont plus applicables le mot est de la forme $\alpha\beta w$" et la règle terminale devient applicable définissant le résultat w" qui est bien sûr $\mathcal{U}'(\mathcal{U}(w))$ puisque l'on a appliqué $\mathcal{U}'$ sur le résultat de l'application de $\mathcal{U}$.

Les propriétés des algorithmes de Markov sont les suivantes :

- Chaque fonction récursive partielle est partiellement Markov-calculable.
- Chaque fonction récursive est Markov-calculable.

2.1.4 Grammaire structurelle élémentaire.

Une grammaire structurelle élémentaire est un ensemble ordonné de règles de transformations.

Une règle de transformation s'applique si le schéma de reconnaissance de cette règle est présent dans l'élément structuré. Dans ce cas la règle s'appliquera sur l'occurrence la plus à droite de l'élément structuré.

Une règle de transformation s'appliquera sur un schéma de reconnaissance enraciné sur le point x de l'élément structuré si aucune règle de rang inférieure ne possède un schéma enraciné sur un descendant de x et si il n'existe aucun ascendant y de x tel que le schéma de reconnaissance puisse être enraciné en ce point y.

Une règle est donc appliquée sur la partie la plus haute et la plus à droite de l'élément structuré.

Un pas d'application de la grammaire structurelle correspondra à une application simultanée de toutes les règles applicables suivant les critères précédents.

Il existe deux types de règle dans une grammaires structurelle élémentaire, les règles ordinaires et les règles terminales.

La grammaire structurelle s'arrête lorsqu'aucune règle n'est applicable ou, lorsque un pas d'application fait intervenir une règle terminale.

Une grammaire structurelle élémentaire correspond donc à une extension des algorithmes de Markov sur les éléments structurés. Cette extension porte également sur la définition d'un pas de transformation puisque toutes les transformations possibles sont effectuées dans ce pas. Ces transformations doivent respectées l'ordre des règles et le non chevauchement de structures.

2.1.4.1 Propriété

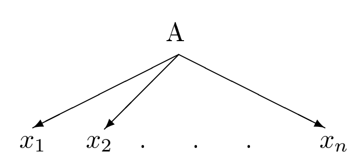
Les propriétés de la grammaire structurelle sont directement issues des propriétés des algortithme de Markov :

- Pour chaque fonction récursive partielle il existe une grammaire structurelle équivalente.
- Pour chaque fonction récursive il existe une grammaire structurelle équivalente.

Preuve :

Le codage d'un mot se fera à partir d'une arborescence à un seul niveau.

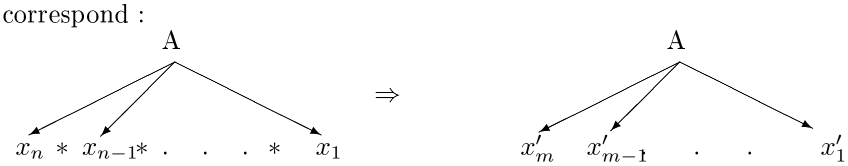
$$w = x_1 x_2 ....x_n$$



Soit $\mathcal{U}$ un algorithme de Markov. A chaque règle $w \rightarrow w'$ de cet algorithme on fait correspondre une règle structurelle :

$$\text{à :} \qquad x_1x_2....x_n \;\; \rightarrow \;\; x'_1x'_2....x'_m$$

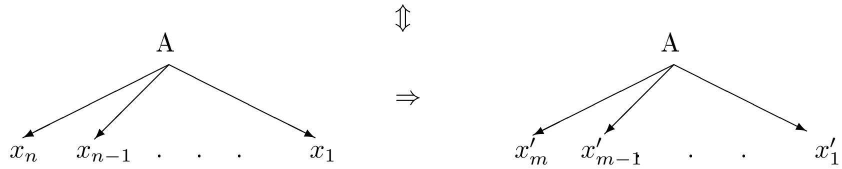
correspond :



Les règles de la grammaire structurelle seront définies dans le même ordre que les règles de l'algorithme et auront ainsi la même priorité. Les mots sont inversés (mots miroir) car les règles de la grammaire structurelle s'appliquent sur les occurences les plus à droite.
Une règle de la grammaire structurelle sera terminale si et seulement si la règle correspondante de l'algorithme est terminale.
La propriété suivante découle driectement de ces définitions :

$$x_1x_2....x_n \;\; \vdash \;\; x'_1x'_2....x'_m$$

$$\Updownarrow$$



Comme conséquence de cette propriété nous avons :
Pour chaque algorithme de Markov il existe une grammaire structurelle élémentaire équivalente.

2.1.4.2 Modes d'applications d'une grammaire élémentaire.

Un algorithme de Markov correspond à une application particulière des grammaires élémentaires. Il existe d'autre modes d'applications qui déterminent un fonctionnement plus restrictif.

Le premier contrôle possible correspond à la définition d'une borne sur le nombre d'itérations possibles de la grammaire. Dans le cas général ce contrôle est défini comme itératif (I) et la grammaire s'arrête lorsqu' aucune règle n'est plus applicable. Il est possible de définir un nombre maximum n d'itérations potentiellement réalisables par la grammaire. Dans ce cas la grammaire s'arrête soit parcequ'aucune règle se trouve applicable, soit parceque ce nombre a été atteint.

Le deuxième contrôle concerne la liste des règles de la grammaire potentiellement applicables. A chaque itération les règles utilisées sont retirées de cette liste. La grammaire s'arrête donc avec un nombre de pas d'itérations qui est au maximum le nombre de règles de cette grammaire. Ce mode de fonctionnement (E) est dit exhaustif (E) du fait qu'il correspond à la recherche de l'application de toutes les règles d'une grammaire.

Le dernier type de contrôle concerne la définition des récurrences. Il existe deux récurrences principales : la récurrence de règle et la récurrence de grammaire. La récurrence de grammaire correspond à un parcours d'une arborescence d'une dimension d'un élément structuré. Avec la grammaire élémentaire on définit deux autres grammaires qui seront appliquées respectivement sur le fils gauche et sur le frère droit de l'arborescence d'entrée :

> Si G_1 est la grammaire qui est appliquée sur la structure $X(X_1, X_2, ..., X_n)$; et si G_1 à comme grammaires de récurrence G_2 et G_3 alors :
> - G_2 sera appliquée sur la structure de racine X_1.
> - G_1 sera appliquée sur l'arborescence de racine X, les arborescences de racines X_1, X_2, ..., X_n étant éventuellement modifiées.
> - G_3 sera appliquée sur la structure de racine le frère droit immédiat de X.
>
> Bien sur dans le cas où $G_1 = G_2 = G_3$ cela correspond à un parcours canonique de l'arborescence d'entrée.

Notation :

$$\begin{array}{l} G_1(G_2, G_3) : \\ \qquad\qquad R_1 \\ \qquad\qquad R_2 \\ \qquad\qquad .... \\ \qquad\qquad R_n \end{array}$$

Le deuxième type de récurrence est défini dans les règles : Une règle récursive sera considérée comme appliquée lorsque sur son résultat la grammaire définie en récurrence sera appliquée. Cette grammaire de récurrence sera appliqée sur une sous-arborescence ayant comme racine un point du schéma. La récurrence de règle correspond donc à une définition de transformation complexe.

Notation :

> $R_1(G; X)$ (le point X est un point du schéma de transformation).

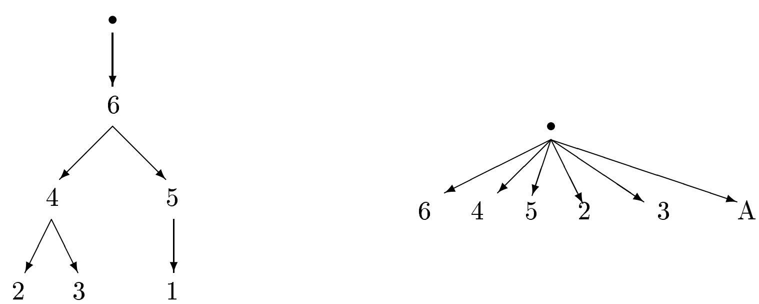
2.1.4.3 Grammaire structurelle.

Une grammaire structurelle est un réseau de grammaires élémentaires. Chaque point de ce réseau corespond à une grammaire élémentaire. Le réseau de grammaire correspond donc à une composition des algorithmes associés aux grammaires élémentaires. Ce réseau est un réseau conditionnel. Une application correspond à un chemin dans ce réseau depuis une grammaire initiale jusqu'à une grammaire finale. On a donc ici un automate d'etats finis de grammaires élémentaires. Cet automate est non déterministe et il fonctionne avec un algorithme de retour arrière. Les arcs dépendants d'une grammaire élémentaire sont ordonnés et conditionnels, les conditions sont définies par la présence ou l'absence de schéma d'éléments structurés. Les chemins possibles sont énumérés dans l'ordre de leurs définitions. La récurrence de règle peut également concerner un cheminement dans le réseau. Elle est désignée dans ce cas comme récurrence généralisée.

Exemples de grammaire :

1. Tri en tas (heap sort) :

   Pour trier une liste linéaire suivant la méthode du tri en tas il faut considérer un élément structuré à deux dimensions :
   – La première dimension contiendra l'arborescence binaire.
   – La deuxième dimension contiendra la liste des éléments.
   Une arborescence est en tas si un point est supérieur à ces deux descendants : exemple de tas :



arborescence en tas

L'algorithme repose sur une procédure de remise en tas. Cet algorithme est défini par une grammaire élémentaire récursive contenant une seule règle :

tas.
rtasm(tas ;[1] : B [2] : y) :

[1] : ∗(0(1-%2)) // (chaine(0) < chaine(1)) &
(chaine(1) > chaine(2))
[2] : x(0,1)

⇒ [1] : A(∗0<,> ∗,B(∗1<,> ∗),%2) / A :1 ; B : 0
[2] : y(∗x<,0> ∗,A,∗x<0,1> ∗,B,∗x<1,> ∗) / y : x ; A :1 ; B : 0.

Cette règle est supposée s'appliquer sur un tas partiel. Par récurrence elle s'appliquera sur un tas dépendant. La racine de l'arborescence de la dimension 1 est comparée à ces deux descendants (1 ou 2 car un point est facultatif). Ce point est échangé avec le plus grand de ces deux descendants qui devient le point B dans le résultat. Ce point sera donc la racine de l'élément transformé par récurrence à l'appel de cette même grammaire. Dans la dimension 2 les échanges seront définis par identité d'étiquette :

1 ère application sur 1(5,3)

Appel récursif avec comme structure (les points B et y étant les racines de l'appel) :

2ème application sur 1(2) :

Appel récursif avec comme structure :

Fin de l'appel récursif, la règle n'est plus applicable résultat :

Cette grammaire doit être précédée d'une grammaire de construction du tas. Elle est ensuite appelée récursivement par une règle de tri qui suivant cet algorithme échange le dernier et le premier élément du tas :

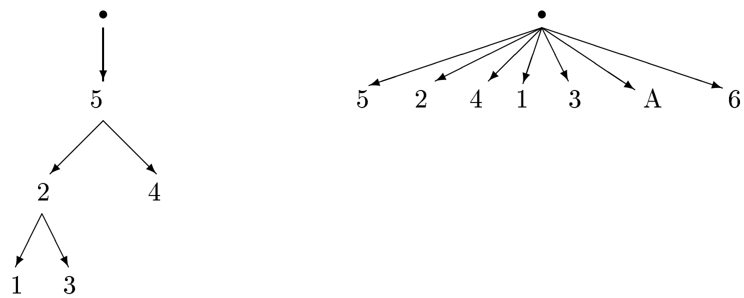
tri(I).

RD(tas ;0) :[1] : ∗(0 ?(1(∗)))
[2] : ∗(x(∗,0,1,∗,B) / B : chaine=″

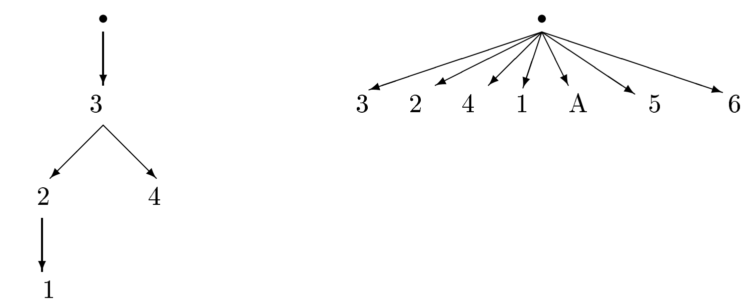
⇒ [1] : 0 / 0 :1
[2] : y(1,∗x<0,1> ∗,B,0,∗x<B,> ∗) / y :x.

RF : [2] ; 0(1) / 1 : chaine = ″ ⇒ [2] : 0.

La dernière règle ne s'appliquera qu'à la fin du tri et permettra de supprimer le point auxiliaire B qui a servi d'index linéaire.



Cette structure est transformée par la règle RD qui positionne le maximum courant à l'endroit repéré par A. La structure devient :

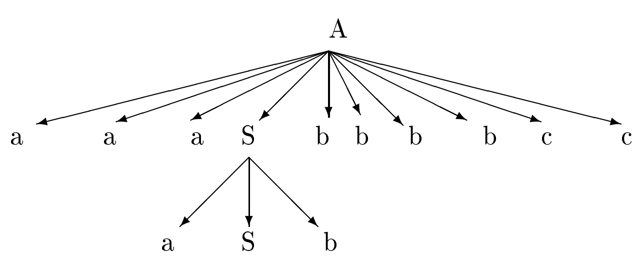


L'application itérative de cette règle produit une liste triée dans la dimension 2.

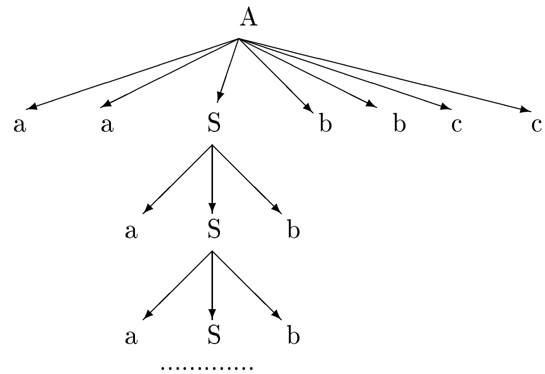
2. Grammaire d'analyse du langage $a^n b^n c^* \cup a^* b^n c^n$

   La grammaire d'analyse de ce langage construit dans une dimension le premier cas et dans l'autre dimension le deuxième cas. Pour une dimension la grammaire comprend deux règles : une règle pour la structure parenthèsée et une règle pour la suite initiale ou finale.

RPAR : [1] : X(a,*,S,*,b) $\Rightarrow$ X(S(a,S,b))

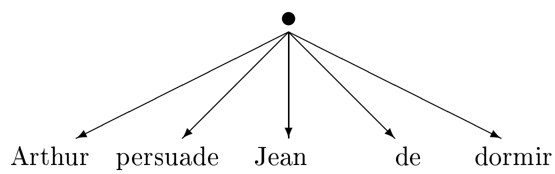


$\Rightarrow$



Le langage $a^n b^n c^n$ peut être également reconnu par cette grammaire. Dans ce cas le critère d'arrêt correspond à une structure correcte dans les deux dimensions et correspond donc à l'intersection des langages hors contextes $a^n b^n c^*$ et $a^* b^n c^n$.

3. Analyse de la phrase "Jean persuade Arthur de dormir". On suppose que l'analyse s'effectue à partir d'une arborescence simple qui comprend sur les feuilles les renseignements possibles associés à chacun des mots :

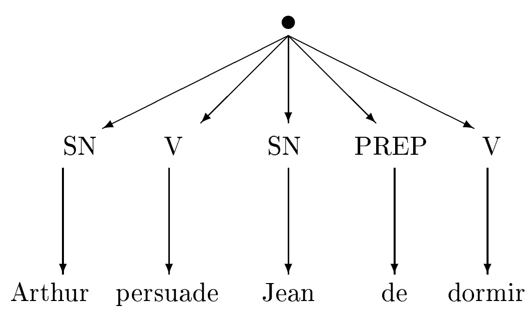


La première grammaire construit les sygntagmes élémentaires :

G1(U) :

RV : 0 / 0 : cat = verbe $\Rightarrow$ X(0) / X :(cat = V).
RSN : 0 / 0 : cat = nom $\Rightarrow$ X(0) / X :(cat = SN).
RPREP : 0 / 0 : cat = prep $\Rightarrow$ X(0) / X :(cat = PREP).

Cette grammaire produit la structure suivante :



La deuxième grammaire construit la structure syntaxique :

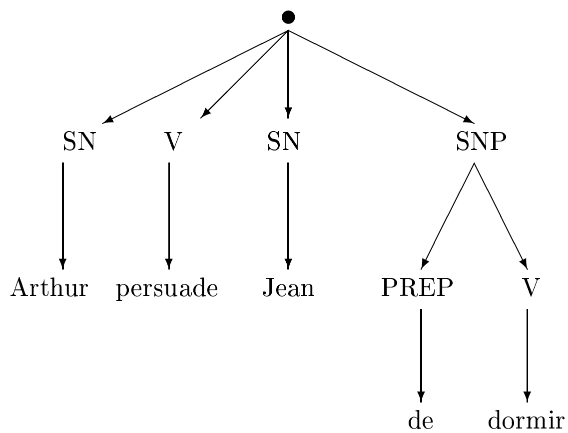
G2(E) :

RSNP : 0,*,1 / 0 : cat = PREP ; 1 : cat = V $\Rightarrow$ X(0,1) / X : (cat = SNP).
RSV : 0,*,1,*,2 / 0 : cat = V ; 1 : cat = SN ; 2 : cat = SNP $\Rightarrow$

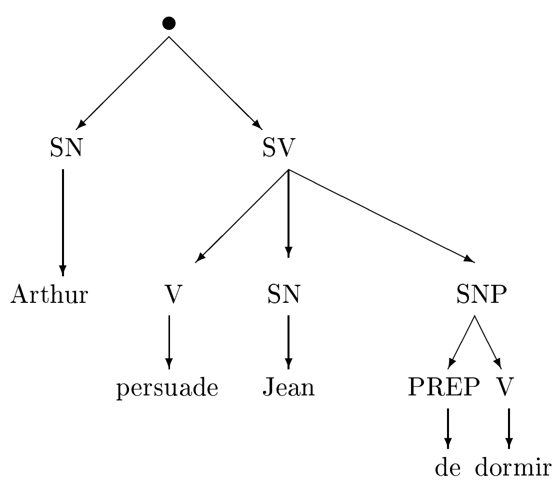
X(0,1,2) / X : (cat = SV).
RP : 0,*,1 / 0 : cat = SN ; 1 : cat = SV ⇒ X(0,1) / X :(cat = P).

Cette grammaire est exhaustive. Les règles s'appliquent suivant leurs priorités :

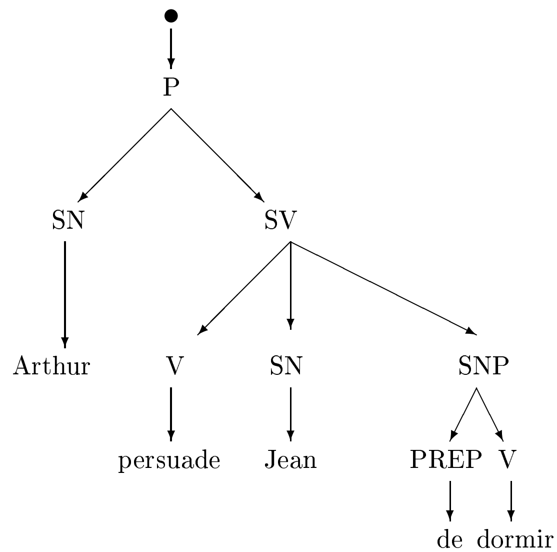
RSNP :



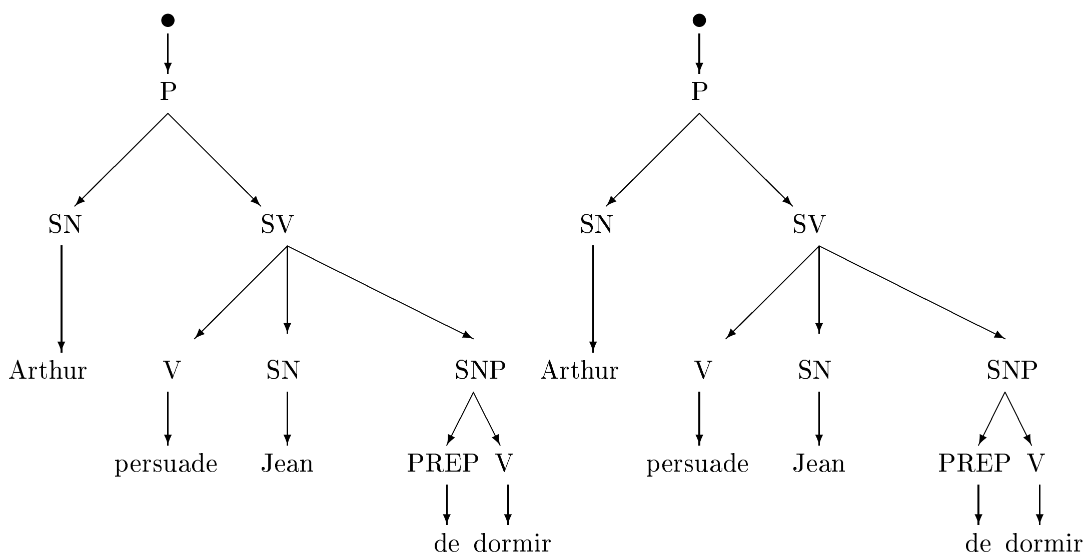
RSV :



RP :



Le résultat est dupliqué dans deux dimensions. La deuxième dimension contiendra la forme logique. La structure de départ de la troisième grammaire est donc la suivante :



Cette grammaire de construction de la forme prédicative est la suivante :

G3(I) :

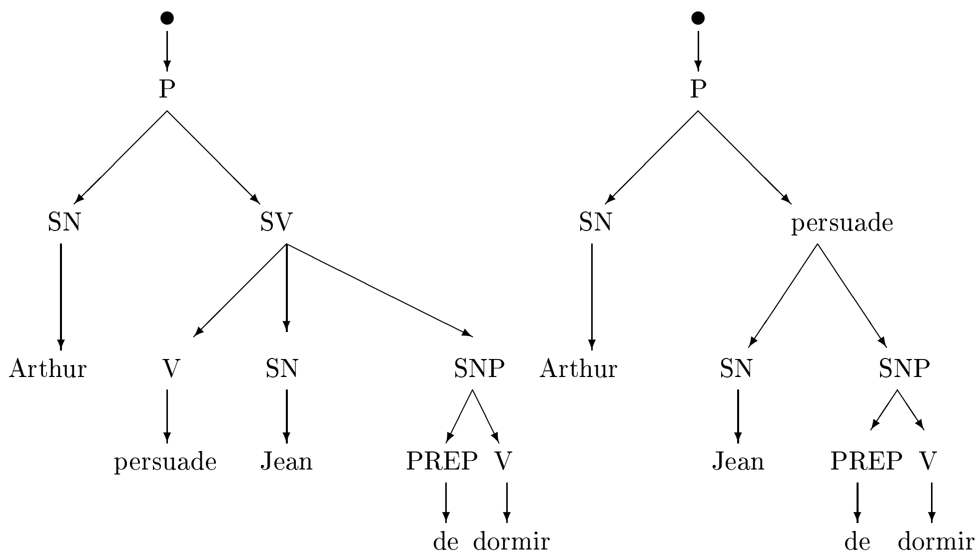
RPRED : [2] : 0(1(2)) / 0 : cat = SV ; 1 : cat = V ; 2 : cat = verbe ⇒
[2] : 2(*0*).

ROBJ : [2] : 0(1(2)) / 0 : cat = verbe ; 1 : cat = SN ; 2 : cat = nom ⇒
[2] : 0(1(2)) / 1 : cat = arg2.

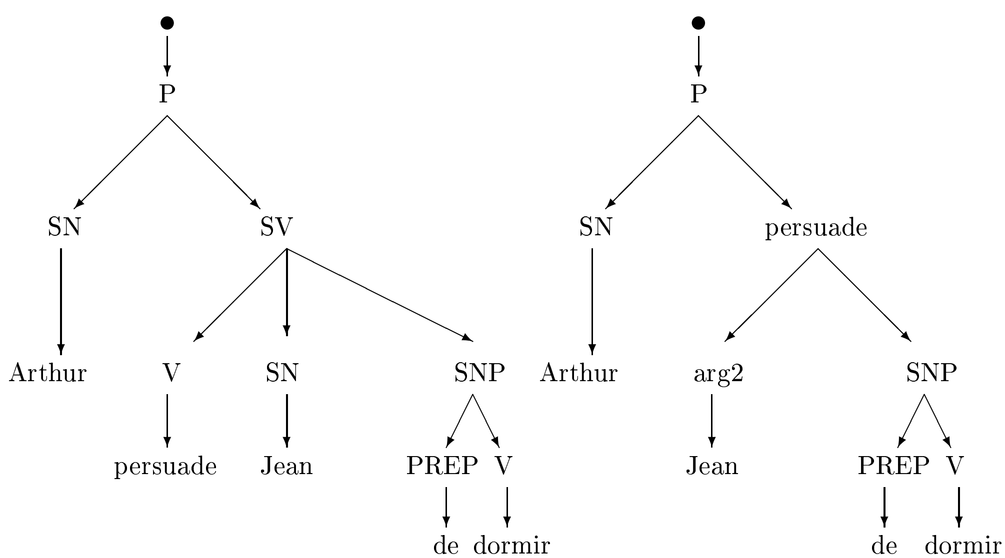
ROBJI : [2] : 0(1(2,3)) /0 : cat = verbe ; 1 : cat = SNP ; 2 : cat = PREP ;

3 : cat = v ⇒
[2] : 0(1(3)) / 1 : cat = arg3.
RSUJ : [2] : 0(1(2),*,3) / 0 : cat =P ; 1 : cat = SN ; 2 : cat = nom 3 : cat = predicat ⇒
[2] : X(1(2),∗3 <, > ∗) / X :3 ; 1 :1(cat = arg1).
RACTUAL : [2] : 0(1(2),3(4(5))) / 0 : cat = predicat ; 1 : cat = arg2 ; 2 : cat = nom ;
3 : cat = arg3 ; 4 : cat = V ; 5 : cat = verbe ⇒
[2] : 0(1(2),3(5(X(Y)))) / X : (cat = arg1) ; Y : 2.

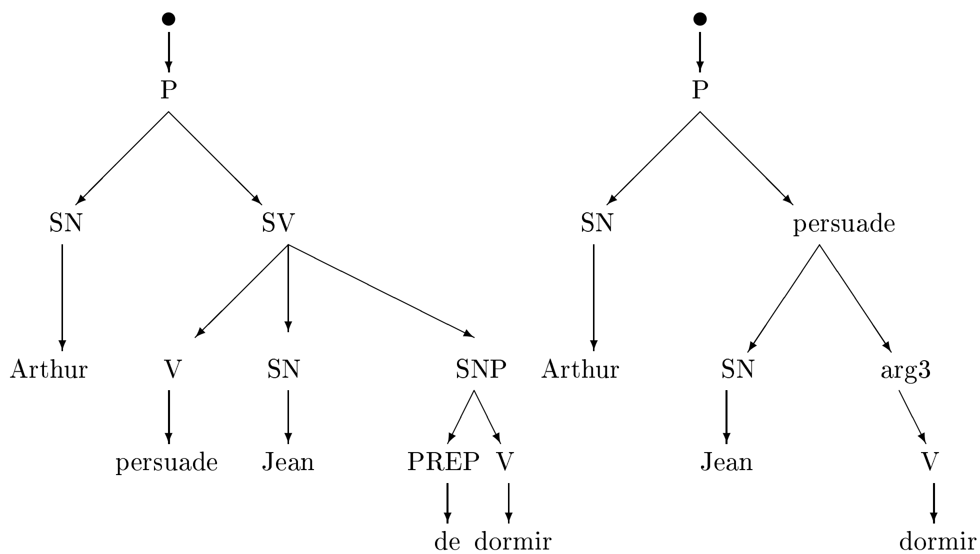
RPRED :



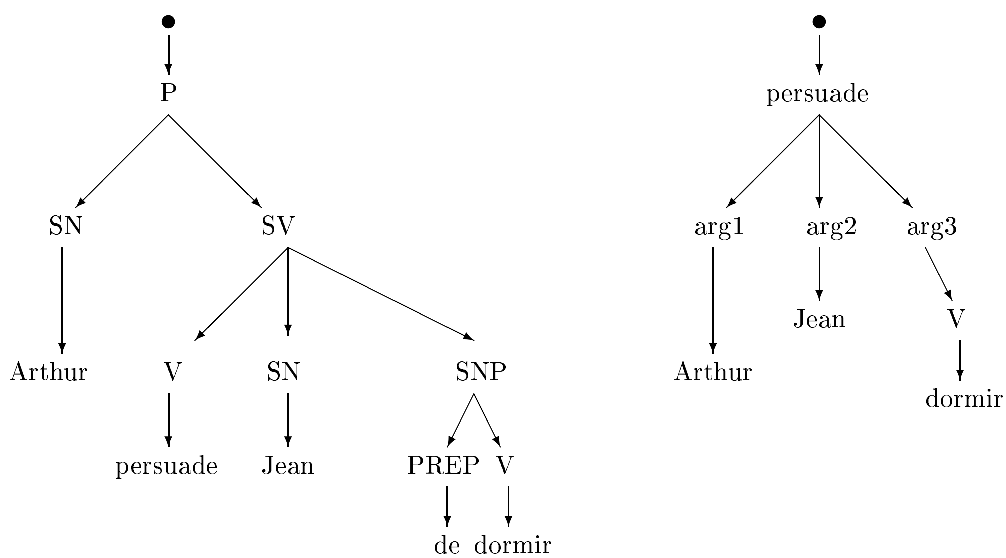
ROBJ :



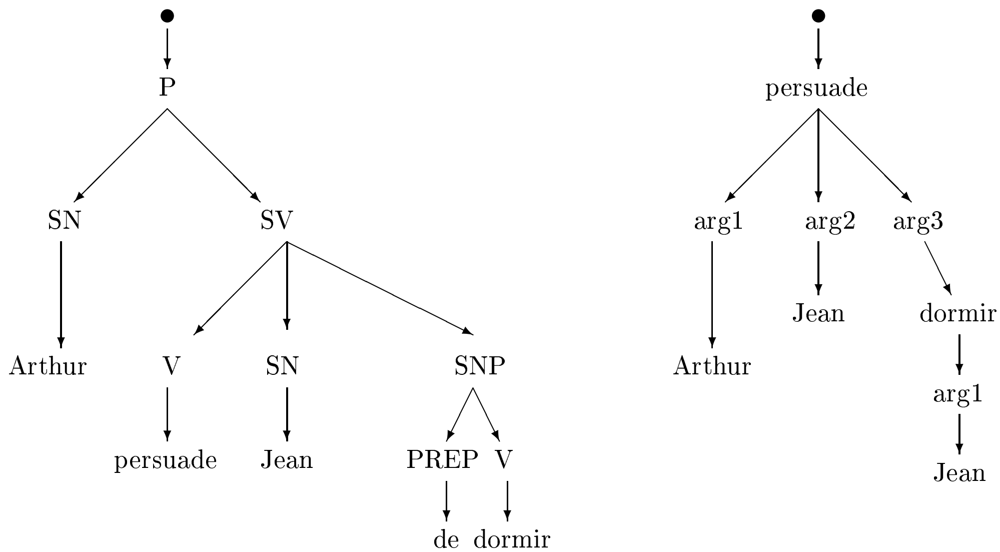
ROBJI :



RSUJ :



RACTUAL :



2.2 Réseaux de transitions augmentés.

Les réseaux de transitions augmentés définissent un processus d'analyse. La description de la langue s'effectue donc ici par la définition de l'algorithme accepteur. Ces réseaux ont pour cadre théorique les automates d'états finis récursifs.

2.2.1 Automates d'états finis récursifs.

Un automate d'états finis récursif est défini en même temps qu'un ensemble d'automates $\mathcal{A}$ dont il dépend.

Soit $\mathcal{A}_1, ..., \mathcal{A}_n$ un ensemble fini d'automates récursifs sur l'ensemble $\mathcal{A}$. Un automate d'états finis récursif $\mathcal{A}_i$ de $\mathcal{A}$ est défini par un sextuplet :

$$\mathcal{A}_i = (V_t \ ,\mathcal{A},\mathcal{Q}_i,\mathcal{Q}_{0_i} \ ,\mathcal{F}_i \ ,\mu_i \ )$$

où :
- V_t est un ensemble fini, le vocabulaire terminal.
- $\mathcal{Q}_i$ est un ensemble fini, l'ensemble des états.
- $\mathcal{Q}_{0_i} \subseteq \mathcal{Q}_i$ est l'ensemble des états initiaux.
- $\mathcal{F}_i \subseteq \mathcal{Q}_i$ est l'ensemble des états finals.
- μ_i est la fonction de transition :

$$\mathcal{Q}_i \times V_t \cup \mathcal{A} \to \mathcal{Q}_i$$

La fonction de transition μ_i induit une relation sur $\mathcal{Q}_i \times V_t^*$ de la façon suivante :

$$(q, w_1 w_2) \underset{\mathcal{A}_i}{\vdash} (q', w_2) \text{ si et seulement si}$$

- Soit $w_1 \in V_t$ et $q' \in \mu_i(q, w_1)$
- Soit $\exists \mathcal{A}_j \in \mathcal{A}$ tel que $q' \in \mu_i(q, \mathcal{A}_j)$ et $w_1 \in L(\mathcal{A}_j)$

Soit $\overset{\star}{\underset{\mathcal{A}_i}{\vdash}}$ l'extension réflexive et transitive de la relation $\underset{\mathcal{A}_i}{\vdash}$:

- $(q,w) \overset{\star}{\underset{\mathcal{A}_i}{\vdash}} (q,w) \quad \forall w \in V_t^*$
- $\forall w, w' \in V_t^*$:

  $(q,w) \overset{\star}{\underset{\mathcal{A}_i}{\vdash}} (q',w') \Leftrightarrow$
  - Soit $q = q'$ *et* $w = w'$
  - Soit $\exists q" \in \mathcal{Q}_i, \quad w_1, w_2 \in V_t^*$ tel que :

    $w = w_1 w_2, \quad (q, w_1 w_2) \underset{\mathcal{A}_i}{\vdash} (q", w_2) \quad$ et $\quad (q", w_2)$

    $\overset{\star}{\underset{\mathcal{A}_i}{\vdash}} (q', w')$

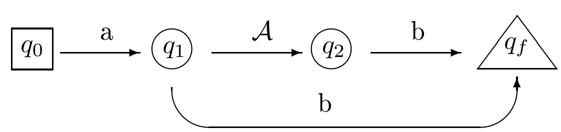
Le langage accepté par un réseau de transition récursif est alors défini par :

$$\mathrm{L}(\mathcal{A}_i) = \{\ w \mid (q_0, w) \overset{\star}{\underset{\mathcal{A}_i}{\vdash}} (q', \epsilon)\ , \quad q_0 \in \mathcal{Q}_0 \quad \text{et} \quad q' \in \mathcal{F}_i\ \}$$

Exemple :

Automate récursif reconnaissant $a^n b^n$:

$\mathcal{A}$:



2.2.2 Grammaire de transition récursive.

Les automates d'états finis récursifs engendrent la notion de grammaires de transitions récursives. Ces grammaires sont définies de la même manière que les automates.

Soit $\mathcal{G}_1, ..., \mathcal{G}_n$ un ensemble $\mathcal{G}$ de grammaires de transitions récursives. Une grammaire de transition récursive $\mathcal{G}_i$ de $\mathcal{G}$ est définie par le quintuplet :

$\mathcal{G}_i = (\mathcal{G}, V_N, V_t, S, \mathcal{P})$ où

$\mathcal{P} : V_N \rightarrow (V_t \cup \mathcal{G}) \times (V_N \cup \{\varepsilon\})$

toutes les règles de $\mathcal{P}$ sont donc de la forme :

- Soit $\Sigma \rightarrow \sigma \Sigma'$
- Soit $\Sigma \rightarrow \mathcal{G}_j \Sigma'$

$\sigma \in V_t, \quad \mathcal{G}_j \in \mathcal{G}, \quad \Sigma' \in V_N \cup \{\varepsilon\}$

La dérivation dans une grammaire de transition récursive est alors définie de la façon suivante :

$\Sigma \underset{\mathcal{G}_i}{\Rightarrow} w$ si et seulement si

- Soit $\Sigma \rightarrow \sigma\Sigma' \in P$ et $w = \sigma\Sigma'$
- Soit $\Sigma \rightarrow \mathcal{G}_j\Sigma' \in \mathcal{P}$ et $\exists w' \in V_t^*$ tel que : $w = w'\Sigma'$ et $w' \in L(\mathcal{G}_j)$.

Soit $\underset{\mathcal{G}_i}{\overset{\star}{\Rightarrow}}$ l'extension réflexive et transitive de la relation $\underset{\mathcal{G}_i}{\Rightarrow}$:

- $w \underset{\mathcal{G}_i}{\overset{\star}{\Rightarrow}} w \quad \forall w \in (V_t \cup V_N)^*$
- $w \underset{\mathcal{G}_i}{\overset{\star}{\Rightarrow}} w'$ si $\exists w" \in (V_t \cup V_N)^*$ tel que : $w \underset{\mathcal{G}_i}{\Rightarrow} w"$ et $w" \underset{\mathcal{G}_i}{\overset{\star}{\Rightarrow}} w'$

Le langage engendré par $\mathcal{G}_i$ est alors défini par :

$$\mathrm{L}(\mathcal{G}_i) = \{\ w \mid w \in V_t^* \text{ et } S \underset{\mathcal{G}_i}{\overset{\star}{\Rightarrow}} w\ \}$$

2.2.2.1 Equivalence.

Une grammaire de transition récursive n'est pas plus puissante qu'une grammaire hors contexte :

Définition de la longueur de dérivation dans une grammaire de transition récursive :

> Le nombre de pas d'une dérivation dans une grammaire de transition récursive est défini par :
> - La longueur d'une dérivation dont le premier pas est défini par l'application d'une règle de la forme $\Sigma \rightarrow \sigma\Sigma'$ où $\sigma \in V_t \cup \{\varepsilon\}$ est la longueur de la dérivation sans le premier pas augmenté de 1.
> - La longueur d'une dérivation dont le premier pas est défini par l'application d'une règle $\Sigma \rightarrow \mathcal{G}_k\Sigma'$ où $\mathcal{G}_k$ appartient à $\mathcal{G}$ est la longueur de la dérivation sans le premier pas augmenté de la longueur de la dérivation définie dans $\mathcal{G}_k$.

2.2.2.2 Propriété :

Pour toute grammaire de transition récursive il existe une grammaire hors contexte qui engendre le même langage.

Soit $\mathcal{G}_i$ une grammaire de transition récursive définie dans $\mathcal{G}$. On suppose sans nuire à la généralité la propriété suivante :
- $\forall j$ tel que $\mathcal{G}_j \in \mathcal{G}$, $\quad \mathcal{G}_j = (\mathcal{G}, V_{N_j}, V_t, S_j, \mathcal{P}_j)$.
- $\forall j, k$ tel que $\mathcal{G}_j$ et $\mathcal{G}_k \in \mathcal{G}$, $j \neq k$: $V_{N_j} \cap V_{N_k} = \emptyset$.

La grammaire hors contexte définie par :

$\mathcal{G}' = (V_N, V_t, S_i, \mathcal{P})$ où :

$$V_N = \bigcup_{j:\mathcal{G}_j \in \mathcal{G}} V_{N_j}$$

$$\mathcal{P} = \bigcup_{j:\mathcal{G}_j \in \mathcal{G}} \mathcal{P}'_j \qquad \text{où } \mathcal{P}'_j \text{ est défini par :}$$

– Si $\Sigma \rightarrow \sigma\Sigma' \in \mathcal{P}_j, \sigma \in V_t$ alors $\Sigma \rightarrow \sigma\Sigma' \in \mathcal{P}'_j$.
– Si $\Sigma \rightarrow \mathcal{G}_k\Sigma' \in \mathcal{P}_j, \mathcal{G}_k \in \mathcal{G}$ alors $\Sigma \rightarrow S_k\Sigma' \in \mathcal{P}'_j$.

Montrons par induction sur la longueur de la dérivation la propriété :

$\forall j$ tel que $\mathcal{G}_j \in \mathcal{G}$, $\forall \Sigma \in V_{N_j}$, $\forall w \in (V_{N_j} \cup V_t)^*$:

$$\text{Si } \Sigma \overset{\star}{\underset{\mathcal{G}_j}{\Rightarrow}} w \quad \text{alors} \quad \Sigma \overset{\star}{\underset{\mathcal{G}'}{\Rightarrow}} w$$

La propriété est vraie pour une dérivation de longueur 1 :

Si $\Sigma \Rightarrow w$ en une dérivation dans l'une ou l'autre des grammaires alors la règle appliquée est de la forme $\Sigma \rightarrow \sigma\Sigma'$, $\sigma \in V_t$ et $\Sigma' \in V_{N_j} \cup \{\varepsilon\}$. Cette règle existe dans les deux ensembles de production et la propriété est démontrée.

Supposons la propriété vraie pour toute dérivation de longueur n.

Si $\Sigma \overset{\star}{\underset{\mathcal{G}_j}{\Rightarrow}} w$ en $n+1$ pas de dérivation. La dérivation a la forme :

$$\Sigma \underset{\mathcal{G}_j}{\Rightarrow} \sigma\Sigma' \overset{\star}{\underset{\mathcal{G}_j}{\Rightarrow}} \sigma w' \quad \text{avec } \sigma w' = w$$

Si le premier pas est produit par une règle de la forme $\Sigma \rightarrow \sigma\Sigma'$, $\sigma \in V_t$, $\Sigma' \in V_{N_j} \cup \{\varepsilon\}$ la règle appliquée appartient à $\mathcal{P}$ et par hypothèse de récurrence :

$$\Sigma \underset{\mathcal{G}'}{\Rightarrow} \sigma\Sigma' \overset{\star}{\underset{\mathcal{G}'}{\Rightarrow}} \sigma w'$$

Sinon $\sigma \in V_t^*$ et par définition de la dérivation :

$\exists \mathcal{G}_k \in \mathcal{G}$ tel que $\sigma \in L(\mathcal{G}_k)$ en moins de $n+1$ pas et tel que la règle appliquée soit de la forme $\Sigma \rightarrow \mathcal{G}_k\Sigma'$.

Donc $S_k \overset{\star}{\underset{\mathcal{G}_k}{\Rightarrow}} \sigma$ en moins de $n+1$ pas et par hypothèse de récurrence : $S_k \overset{\star}{\underset{\mathcal{G}'}{\Rightarrow}} \sigma$.

Par définition de $\mathcal{P}'_j$, $\mathcal{P}$ contient une règle de la forme : $\Sigma \rightarrow S_k\Sigma'$. De plus $\Sigma' \overset{\star}{\underset{\mathcal{G}_j}{\Rightarrow}} w'$ en moins de $n+1$ pas et par définition de la production appliquée $\Sigma' \in V_N \cup \{\varepsilon\}$ donc

si $\Sigma' = \varepsilon$ alors $\Sigma \overset{\star}{\underset{\mathcal{G}'}{\Rightarrow}} \sigma = w$

si $\sigma' \neq \varepsilon$ $\Sigma' \overset{\star}{\underset{\mathcal{G}_j}{\Rightarrow}} w'$ en moins de $n+1$ pas donc

par hypothèse de récurrence : $\Sigma' \overset{\star}{\underset{\mathcal{G}'}{\Rightarrow}} w'$

et

$$\Sigma \underset{\mathcal{G}'}{\Rightarrow} S_k\Sigma' \overset{\star}{\underset{\mathcal{G}'}{\Rightarrow}} \sigma\Sigma' \overset{\star}{\underset{\mathcal{G}'}{\Rightarrow}} \sigma w' = w$$

Un cas particulier de la propriété devient :

$S_i \underset{\mathcal{G}_i}{\overset{\star}{\Rightarrow}} w$ implique : $S_i \underset{\mathcal{G}'}{\overset{\star}{\Rightarrow}} w$

Donc $L(\mathcal{G}_i) \subseteq L(\mathcal{G}')$.

Montrons par induction sur la longueur de la dérivation la propriété :

$\forall j$ tel que $\mathcal{G}_j \in \mathcal{G}$, $\forall \Sigma \in V_{N_j}$, $\forall w \in V_t^*$ et pour tout $\Sigma' \in V_{N_j} \cup \{\varepsilon\}$:

Si $\Sigma \underset{\mathcal{G}'}{\overset{\star}{\Rightarrow}} w\Sigma'$ alors $\Sigma \underset{\mathcal{G}'_j}{\overset{\star}{\Rightarrow}} w\Sigma'$

La propriété est vraie pour une dérivation de longueur 1 :

Si $\Sigma \underset{\mathcal{G}'}{\overset{\star}{\Rightarrow}} w\Sigma'$ en une dérivation de longueur 1 alors la règle appliquée est de la forme $\Sigma \rightarrow w\Sigma'$ et par construction de $\mathcal{P}'_j$ elle appartient à $\mathcal{P}_j$ et donc $\Sigma \underset{\mathcal{G}_j}{\Rightarrow} w\Sigma'$.

Supposons la propriété vraie pour toute dérivation de longueur n.

Si $\Sigma \underset{\mathcal{G}'}{\overset{\star}{\Rightarrow}} w\Sigma'$ en $n+1$ pas de dérivation, la dérivation est de la forme :

$\Sigma \underset{\mathcal{G}'}{\Rightarrow} w'\Sigma" \underset{\mathcal{G}'}{\overset{\star}{\Rightarrow}} w'w"\Sigma'$ et $w'w" = w$.

Si la règle appliquée est de la forme

$\Sigma \rightarrow \sigma\Sigma"$, $\sigma \in V_t$, elle appartient à $\mathcal{P}_j$. Alors $\sigma = w'$ et par hypothèse d'induction $\Sigma" \underset{\mathcal{G}_j}{\overset{\star}{\Rightarrow}} w"\Sigma'$ donc $\Sigma \underset{\mathcal{G}_j}{\overset{\star}{\Rightarrow}} w'\Sigma"$ $\underset{\mathcal{G}_j}{\overset{\star}{\Rightarrow}} w'w"\Sigma'$

Sinon la règle appliquée est de la forme

$\Sigma \rightarrow S_k\Sigma"$ et $\Sigma \underset{\mathcal{G}'}{\Rightarrow} S_k\Sigma" \underset{\mathcal{G}'}{\overset{\star}{\Rightarrow}} w'\Sigma" \underset{\mathcal{G}'}{\overset{\star}{\Rightarrow}} w'w"\Sigma'$

Donc La règle $\Sigma \rightarrow \mathcal{G}_k\Sigma" \in \mathcal{P}_j$ et comme par hypothèse d'induction

$S_k \underset{\mathcal{G}'}{\overset{\star}{\Rightarrow}} w'$ en moins de $n+1$ pas

$S_k \underset{\mathcal{G}_k}{\overset{\star}{\Rightarrow}} w'$ et $w' \in L(\mathcal{G}_k)$.

on a donc : $\Sigma \underset{\mathcal{G}_j}{\Rightarrow} w'\Sigma"$

par hypothèse d'induction on a également :

$\Sigma" \underset{\mathcal{G}_j}{\overset{\star}{\Rightarrow}} w"\Sigma'$

Donc : $\Sigma \underset{\mathcal{G}_j}{\Rightarrow} w'\Sigma" \underset{\mathcal{G}_j}{\overset{\star}{\Rightarrow}} w'w"\Sigma'$

Un cas particulier de cette propriété devient :

$S_i \underset{\mathcal{G}'}{\overset{\star}{\Rightarrow}} w$ implique $S_i \underset{\mathcal{G}_i}{\overset{\star}{\Rightarrow}} w$

Donc $L(\mathcal{G}') \subseteq L(\mathcal{G}_i)$ et $L(\mathcal{G}_i) = L(\mathcal{G}')$.

Une grammaire de transition récursive est donc une forme particulière des grammaires hors contexte.

Cette propriété permet de situer le type de langage accepté par des automates d'etats finis récursifs.

2.2.2.3 Propriété :

Pour tout automate d'états finis récursif il existe une grammaire de transition récursive qui engendre le langage reconnu par cet automate.

Soit $\mathcal{A}_i$ un automate d'états finis récursif :

$\mathcal{A}_i = (V_t, \mathcal{A}, \mathcal{Q}, \mathcal{Q}_0, \mathcal{F}, \mu_i)$

La grammaire de transition récursive est définie par :

$\mathcal{G}_i = (\mathcal{G}, V_{N_i}, V_t, S_i, \mathcal{P}_i)$ où

$$\mathcal{G} = \bigcup_{j:\mathcal{A}_j \in \mathcal{A}} \mathcal{G}_j$$

$\forall j$:

$\mathcal{G}_j = (\mathcal{G}, V_{N_j}, V_t, S_j, \mathcal{P}_j)$ où

$V_{N_j} = \mathcal{Q}_j \cup \{S_j\}$

$\mathcal{P}_j$:

$S_j \rightarrow q_{o_j} \;\; \forall q_{o_j} \in \mathcal{Q}_{0_j}$

$q \rightarrow \sigma q'$ si $q' \in \mu_j(q, \sigma)$ $\forall q, q' \in \mathcal{Q}_j, \sigma \in V_t$.

$q \rightarrow \mathcal{G}_k q'$ si $q' \in \mu_j(q, \mathcal{A}_k)$ $\forall q, q' \in \mathcal{Q}_j, \mathcal{A}_k \in \mathcal{A}$.

$q \rightarrow \sigma$ si $\exists q' \in \mathcal{Q}_j$ tel que : $q' \in \mathcal{F}_j$ et $q' \in \mu_j(q, \sigma)$

$q \rightarrow \mathcal{G}_k$ si $\exists q'$ tel que : $q' \in \mathcal{F}j$ et $q' \in \mu_j(q, \mathcal{A}_k)$.

La longueur d'une transition d'un automate d'états finis récursif est définie par :

- Si $(q, w) \underset{\mathcal{A}_j}{\vdash} (q', \varepsilon)$ parceque $q' \in \mu_j(q, w)$ alors la transition est dite de longueur 1.
- Si $(q, w) \underset{\mathcal{A}_j}{\vdash} (q', \varepsilon)$ parceque $\exists \mathcal{A}_k \in \mathcal{A}$ tel que $w \in L(\mathcal{A}_k)$ et $q' \in \mu_j(q, \mathcal{A}_k)$ alors la longueur de la transition est égale à la longueur de la transition dans $\mathcal{A}_j$ nécessaire pour reconnaître w augmentée de 1.

– Si $(q, w_1w_2) \underset{\mathcal{A}_j}{\vdash} (q", w_2) \overset{\star}{\underset{\mathcal{A}_j}{\vdash}} (q', \varepsilon)$ et $w_2 \neq \varepsilon$ alors la longueur de la dérivation est égale la somme des longueurs des dérivations $(q, w_1) \underset{\mathcal{A}_j}{\vdash} (q", \varepsilon)$ et $(q", w_2) \overset{\star}{\underset{\mathcal{A}_j}{\vdash}} (q', \varepsilon)$.

Montrons par induction sur la longueur de la dérivation la propriété suivante :

$\forall j : \mathcal{G}_j \in \mathcal{G}, \forall q \in \mathcal{Q}_j$:

$$\text{Si } q \overset{\star}{\underset{\mathcal{G}_j}{\Rightarrow}} w \quad \text{alors} \quad (q, w) \overset{\star}{\underset{\mathcal{A}_j}{\vdash}} (q', \varepsilon) \text{ et } q' \in \mathcal{F}_j$$

La propriété est vraie pour toute dérivation de longueur 1 :

Si $q \underset{\mathcal{G}_j}{\Rightarrow} w$ alors la règle appliquée est de la forme $q \rightarrow w$ et $\exists q_f \in \mathcal{F}_j$ tel que $q_f \in \mu_j(q, w)$

donc $(q, w) \underset{\mathcal{A}_j}{\vdash} (q_f, \varepsilon)$

Supposons la propriété vraie pour toute dérivation de longueur n.

Soit une dérivation de longueur $n+1$ tel que $q \overset{\star}{\underset{\mathcal{G}_j}{\Rightarrow}} w$ et $w \in V_t^*$

Cette dérivation est nécessairement de la forme :

$$q \underset{\mathcal{G}_j}{\Rightarrow} \sigma q" \overset{\star}{\underset{\mathcal{G}_j}{\Rightarrow}} w'w" \text{ avec } q" \overset{\star}{\underset{\mathcal{G}_j}{\Rightarrow}} w" \text{ et } w = w'w"$$

et par définition de $\mathcal{G}_j$:
- soit $\sigma \in V_t$ et $\sigma = w'$,
- soit $\sigma = \mathcal{G}_k$ et $w' \in L(\mathcal{G}k)$.

1. Si $\sigma \in V_t$ nécessairement $q" \in \mu_j(q, \sigma)$ et :

   $(q, \sigma w") \underset{\mathcal{A}_j}{\vdash} (q", w")$

   par hypothèse d'induction :

   $q" \overset{\star}{\underset{\mathcal{G}_j}{\Rightarrow}} w"$ implique $(q", w") \overset{\star}{\underset{\mathcal{A}_j}{\vdash}} (q', \varepsilon)$ et $q' \in \mathcal{F}_j$

   donc $(q, \sigma w") \underset{\mathcal{A}_j}{\vdash} (q", w") \overset{\star}{\underset{\mathcal{A}_j}{\vdash}} (q', \varepsilon)$ et $q' \in \mathcal{F}_j$ soit

   $(q, \sigma w") \overset{\star}{\underset{\mathcal{A}_j}{\vdash}} (q', \varepsilon)$ et $q' \in \mathcal{F}_j$.

2. Si $\sigma = \mathcal{G}_k$ alors $w' \in L(\mathcal{G}_k)$ et $S_k \overset{\star}{\underset{\mathcal{G}_k}{\Rightarrow}} w'$ dans une dérivation de longueur inférieure à $n+1$.

   Donc $S_k \underset{\mathcal{G}_k}{\Rightarrow} q_{0_k} \overset{\star}{\underset{\mathcal{G}_k}{\Rightarrow}} w'$ car les seules règles ayant pour partie gauche $S_k \in \mathcal{G}_k$ sont de la forme $S_k \rightarrow q_{0_k}$, $q_{0_k} \in \mathcal{Q}_{0_k}$

par hypothèse d'induction :

$q_{0_k} \underset{\mathcal{G}_k}{\overset{\star}{\Rightarrow}} w'$ implique $(q_{0_k}, w') \underset{\mathcal{A}_k}{\overset{\star}{\vdash}} (q', \varepsilon)$ et $q' \in \mathcal{F}_k$ et donc $w' \in L(\mathcal{A}_k)$.

Puisque la règle appliquée contient $\mathcal{G}_k q"$, $q" \in \mu_j(q, \mathcal{A}_k)$ et donc

$(q, w'w") \underset{\mathcal{A}_j}{\vdash} (q", w")$

Comme $q" \underset{\mathcal{G}_j}{\overset{\star}{\Rightarrow}} w"$ en moins de $n+1$ pas, par hypothèse d'induction on a également :

$(q", w") \underset{\mathcal{A}_j}{\overset{\star}{\vdash}} (q', \varepsilon)$ et $q' \in \mathcal{F}_j$

donc $(q, w'w") \underset{\mathcal{A}_j}{\vdash} (q", w") \underset{\mathcal{A}_j}{\overset{\star}{\vdash}} (q', \varepsilon)$ et $q' \in \mathcal{F}_j$.

soit : $(q, w) \underset{\mathcal{A}_j}{\overset{\star}{\vdash}} (q', \varepsilon)$ et $q' \in \mathcal{F}_j$.

Comme cas particulier de cette propriété nous avons :

$w \in L(\mathcal{G}_i)$ implique : $S_i \underset{\mathcal{G}_i}{\overset{\star}{\Rightarrow}} w$ c'est à dire : $\exists q_{0_i} \in \mathcal{Q}_{0_i}$ tel que :

$S_i \underset{\mathcal{G}_i}{\Rightarrow} q_{0_i} \underset{\mathcal{G}_i}{\overset{\star}{\Rightarrow}} w$ donc $(q_{0_i}, w) \underset{\mathcal{A}_i}{\overset{\star}{\vdash}} (q_f, \varepsilon)$ et $q_f \in \mathcal{F}_i$.

donc $w \in L(\mathcal{A}_i)$.

Donc $L(\mathcal{G}_i) \subseteq L(\mathcal{A}_i)$.

Montrons par induction sur la longueur de la transition la propriété suivante :

Si $(q, w) \underset{\mathcal{A}_j}{\overset{\star}{\vdash}} (q', \varepsilon)$ et $q' \in \mathcal{F}_j$ alors $q \underset{\mathcal{G}_j}{\overset{\star}{\Rightarrow}} w$

La propriété est vraie pour des transition de longueur 1.

La seule possibilité dans ce cas est $w \in V_t$ et donc une application d'une transition tel que $q' \in \mu_j(q, w)$ et $q' \in \mathcal{F}_j$

donc il existe une règle dans $\mathcal{G}_j$ de la forme $q \rightarrow w$ et donc

$q \underset{\mathcal{G}_j}{\Rightarrow} w$

Supposons la propriété vraie pour toutes les transitions de longueur n.

Soit une transition de longueur $n+1$ telle que

$(q, w) \underset{\mathcal{A}_j}{\overset{\star}{\vdash}} (q', \varepsilon)$ et $q' \in \mathcal{F}_j$

Nécessairement $w = w'w"$ et l'on a :

$$(q, w'w") \underset{\mathcal{A}_j}{\vdash} (q", w") \underset{\mathcal{A}_j}{\overset{\star}{\vdash}} (q', \varepsilon)$$

Par hypothèse d'induction $q" \underset{\mathcal{G}_j}{\Rightarrow} w"$.

1. Si $w' \in V_t$ et $q" \in \mu_j(q, w')$ alors il existe une règle de $\mathcal{G}_j$ de la forme $q \rightarrow w'q"$.

   donc : $q \underset{\mathcal{G}_j}{\Rightarrow} w'q" \underset{\mathcal{G}_j}{\overset{\star}{\Rightarrow}} w'w" = w$

2. Sinon $w' \in V_t^*$ et il existe un automate $\mathcal{A}_k$ tel que $q" \in \mu_j(q, \mathcal{A}_k)$ et $w' \in L(\mathcal{A}_k)$. Il existe donc dans $\mathcal{G}_j$ une règle de la forme $q \rightarrow \mathcal{G}_k q"$. $w' \in L(\mathcal{A}_k)$ dans une transition de longueur inférieure à $n+1$ on a :

   $\exists q_0 \in \mathcal{Q}_{0_k}$ tel que : $(q_0, w') \underset{\mathcal{A}_k}{\overset{\star}{\vdash}} (q_f, \varepsilon)$ en moins de $n+1$ pas.

   Par hypothèse d'induction : $q_0 \underset{\mathcal{G}_k}{\overset{\star}{\Rightarrow}} w'$

   Comme pour tout élément $q_0 \in \mathcal{Q}_{0_k}$ il existe une règle de $\mathcal{G}_k$ de la forme $S_k \rightarrow q_0$ on a

   $$S_k \underset{\mathcal{G}_k}{\Rightarrow} q_0 \underset{\mathcal{G}_k}{\overset{\star}{\Rightarrow}} w' \text{ et } w' \in L(\mathcal{G}_k).$$

   donc : $q \underset{\mathcal{G}_j}{\Rightarrow} w'q" \underset{\mathcal{G}_j}{\overset{\star}{\Rightarrow}} w'w" = w$

Comme application de cette propriété nous avons :

$w \in L(\mathcal{A}_i)$ implique :

$\exists q_0 \in \mathcal{Q}_{0_i}$ tel que $(q_0, w) \underset{\mathcal{A}_i}{\overset{\star}{\vdash}} (q_f, \varepsilon)$ $q_f \in \mathcal{Q}_{F_i}$.

donc $q_0 \underset{\mathcal{G}_i}{\overset{\star}{\Rightarrow}} w$. Et comme il existe une règle dans $\mathcal{G}_i$ de la forme $S_i \rightarrow q_0$ on a :

$$S_i \underset{\mathcal{G}_i}{\Rightarrow} q_0 \underset{\mathcal{G}_i}{\overset{\star}{\Rightarrow}} w \text{ soit } w \in L(\mathcal{G}_i).$$

Donc $L(\mathcal{A}_i) \subseteq L(\mathcal{G}_i)$ et $L(\mathcal{G}_i) = L(\mathcal{A}_i)$.

Cette propriété montre que les langages reconnus par des automates d'états finis récursifs sont hors contexte.

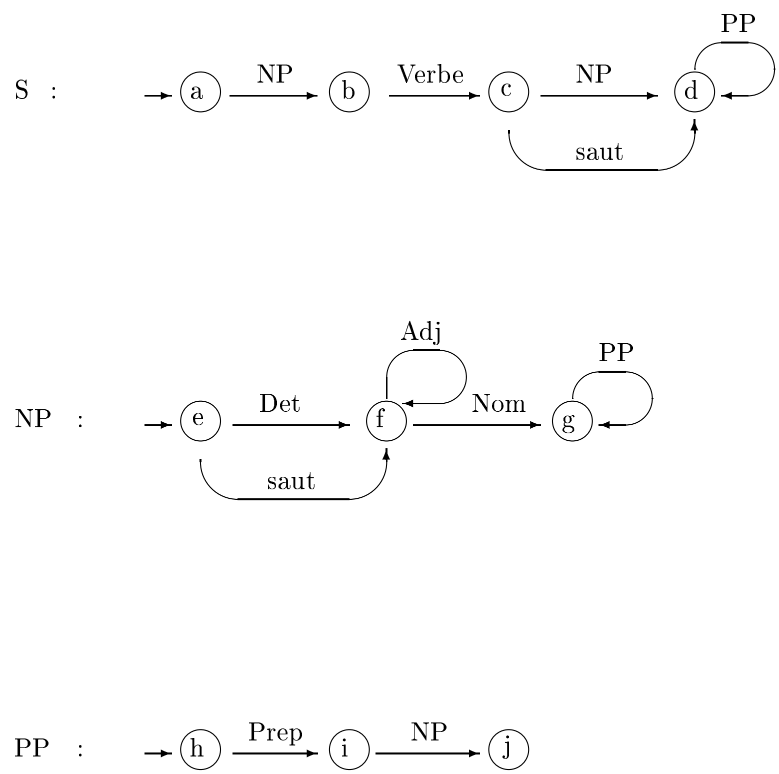
Le fait que tous les langages hors contextes puissent être reconnus par des automates d'états finis récursifs est une propriété de moindre importance. Pour montrer cette propriété on peut considérer la grammaire de forme normale de Chomsky qui engendre un langage. L'automate d'etats finis récursif est alors obtenu en considérant pour chaque non terminal à la fois un automate portant son identification et un symbole d'état des automates d'états finis récursifs. Ce processus de construction est employé pour définir la grammaire hors contexte engendrant le langage reconnu par un automate d'états finis récursif. Il sera illustré sur les ATN.

2.2.3 Réseau de transition augmenté.

Les réseaux de transition sont des automates d'états finis récursifs. La notion de saut qui n'augmente pas la puissance des automates mais simplifie l'écriture de ceux ci est ajoutée :

> Un saut entre deux états permet d'effectuer une transition sans lecture de l'entrée ni appel d'automate.
>
> Formellement cela revient à synthétiser les transitions de la façon suivante :
>
> > Si il existe un saut entre les états q et q' cela revient à remplacer ce saut par toutes les transifions de la forme :
> >
> > $q" \in \mu(q,\sigma) \forall q"$ et $\forall \sigma$ tel que $q" \in \mu(q',\sigma)$.

Exemple de réseau de transition récursif (Winograd et Sabah) :



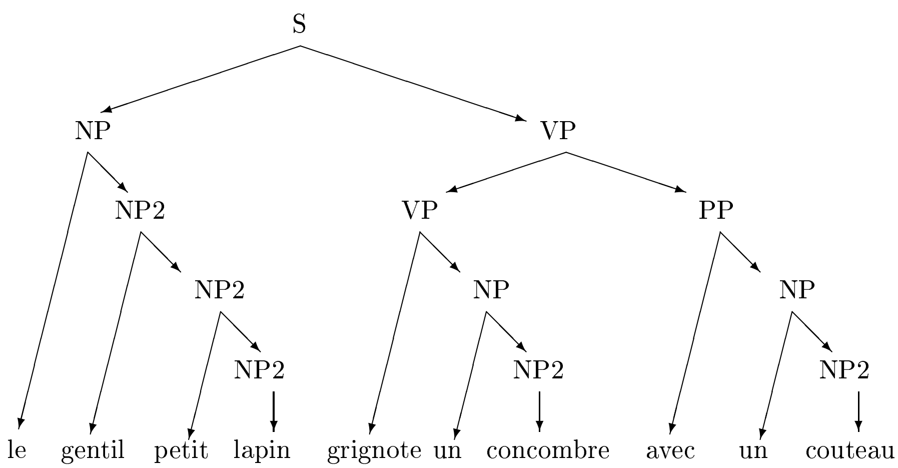
La grammaire hors contexte faiblement équivalente (dans le sens où elle accepte le même langage) est la suivante :

$$
\begin{array}{rcl}
\mathrm{S} & \rightarrow & \mathrm{NP\ VP} \\
\mathrm{NP} & \rightarrow & \mathrm{Det\ NP2} \\
\mathrm{NP} & \rightarrow & \mathrm{NP2} \\
\mathrm{NP2} & \rightarrow & \mathrm{Nom} \\
\mathrm{NP2} & \rightarrow & \mathrm{Adj\ NP2} \\
\mathrm{NP2} & \rightarrow & \mathrm{NP2\ PP} \\
\mathrm{PP} & \rightarrow & \mathrm{Prep\ NP} \\
\mathrm{VP} & \rightarrow & \mathrm{Verbe} \\
\mathrm{VP} & \rightarrow & \mathrm{Verb\ NP} \\
\mathrm{VP} & \rightarrow & \mathrm{Verb\ PP}
\end{array}
$$

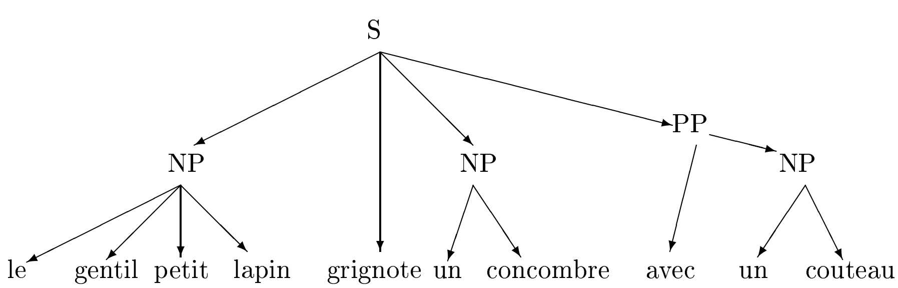
Cette grammaire donnera pour l'analyse d'une phrase comme

"Le gentil petit lapin grignote un concombre avec un couteau"

la structure syntaxique suivante :

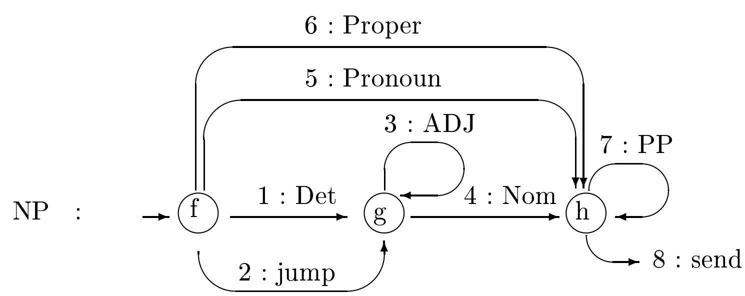


En définissant une structure issue d'un automate d'états finis récursif comme étant obtenue par le nom de l'automate ayant pour descendants l'ensemble des transitions de cet automate nous obtenons comme structure pour l'analyse de cette phrase :



Les réseaux de transitions sont dits augmentés car ils possèdent des fonctionnalités qui sont ajoutées à l'automate de reconnaissance. Ces fonctionalités ont pour but de rendre opérationnelles les définitions de la grammaire transformationnelle et de dépasser la puissance des automates d'états finis récursifs. Nous trouvons donc dans les réseaux augmentés la définition de registres et la définition d'actions associées aux transitions de l'automate récursif. Ces actions permettent d'effectuer des déplacements de fragments de phrases, de copier ou détruire des parties de la structure courante, de définir des conditions de contexte.

Exemple de reseau (Winograd) :



Feature Dimensions : Number : Singular, Plural ; default, -emty-

Initialisation, Conditions, actions :

NP-1 : $_{f}$Determiner $_{g}$
A : Set Number to the Number of *.

NP-4 :$_{g}$Noun$_{h}$
C : Number is empty or Number is the Number of *.
A : Set Number to the Number of *

NP-5 :$_{f}$Pronoun$_{h}$
A : Set Number to the Number of *.

NP-6 : $_{f}$Proper$_{h}$
A : Set Number to the Number of *.

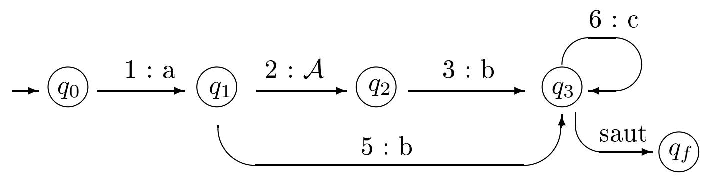
Le symbole "*" représente le noeud du graphe le plus récent. C'est à dire soit le résultat d'un appel récursif de réseau, soit la lecture d'une entrée lexicale. Les conditions (C) portent sur les valeurs de registres. Les actions (A) peuvent concerner la structure ou les registres.

Du fait que les réseaux de transitions augmentés peuvent traiter les structures de façon similaire à la grammaire transformationnelle, la puissance de ce formalisme devient celle des grammaires de type 0.

Exemple : réseau reconnaissant $a^n b^n c^n$:

L'expression la plus simple consiste à définir un registre entier N.

$\mathcal{A}$:



Registre : N initial : 0

A-1 :q_0aq_1
A : Augmenter N de 1.

A-6 : q_3cq_3
C : N doit être non nul.
A : diminuer N de 1.

A-4 : q_3Sautq_f
C : N doit être nul.

Bibliographie

[1] P.B Andrews : An introduction to mathematical logic and type theory : To Truth through proof. Academic Press

[2] R. B. Barneji : Artificial intelligence. A theorical approach. North Holland

[3] A. Barr E.A. Feignbaum : Le manuel de l'intelligence artificielle. Eyrolles

[4] H. Bestougeff G. Ligozat : Outils logiques pour le traitement du sens. Masson

[5] H Farreny M. Ghallab : Elément d'intelligence artificielle. Hermes

[6] P. Gochet P. Gribomont : Logique Méthode pour l'informatique fondamentale. Hermes

[7] E. Gregoire : Logiques non monotones et Intelligence artificielles. Hermes

[8] M. Griffiths : Intelligence artificielle : Technique algorithmique. Hermes

[9] F. Hayes-Roth D.A. Waterman D.B. Lenat : Building Expert Systems. Addison Wesley

[10] Hopcoft Hullman : Formal languages and their relations to automata. Addison-Wesley

[11] A. Kaufmann : Nouvelles logiques pour l'Intelligence artificielle. Hermes

[12] E Mendelson : Introduction to Mathematical logic. Van Nostrand Reinhold

[13] R.S Michalski J.G. Carbonell T.M Mitchell : Machine learning An artifical intelligence approach. Springer Verlag

[14] P. Miller T. Torris : Formalismes syntaxiques pour le traitement automatique du langage naturel. Hermes

[15] N.J. Nilson : Principle of artificial intelligence. Springer Verlag

[16] G Sabah : L'intelligence artificielle et le langage. Hermes

[17] J.F Sowa : Conceptual structures. Addison-Wesley

[18] J.F Sowa : Principles of Semantic Networks. Morgan Kaufmann

[19] A.Thayse & co-auteurs : Approche logique de l'intelligence artificielle. Dunod

1 De la logique classique à la programmation logique

2 De la logique modale à la logique des bases de données

3 Du traitement de la langue à la logique des systèmes experts

4 De l'apprentissage aux frontières de l'IA.

[20] A. Walker M.Cord J. Sowa W. Wilson : Knowledge Systems and Prolog. Addison Wesley

[21] T. Winograd : Language as a cognitive process. Addison-Wesley

[22] P.W Winston : Intelligence artificielle. Inter-Edition

Index