La grammaire structurelle

J. Chauché

Janvier 2012

Table des matières

1 INTRODUCTION

La grammaire structurelle définit le modèle formel du système SYGMART. Ce modèle permet de décrire des algorithmes d'analyse suivant un modèle de réécriture de structures. Ce modèle est indécidable et a la puissance d'une machine de Turing. Cette équivalence est très simplement montrée par la projection de tout algorithme de Markov dans une grammaire structurelle. Bien sûr l'inverse est beaucoup plus complexe et ne peut se faire que par le passage d'une machine de Turing simulant une grammaire structurelle. L'implémentation d'une grammaire structurelle dans le système SYGMART impose bien sûr des limitations sur le nombre de règles, la taille des structures manipulables ou la complexité des étiquettes acceptables. A titre d'information, les capacités actuelles du système permettent d' écrire environ 50 000 règles d'une vingtaine de points chacune et de traiter une arborescence d'environ 200 000 points dans un temps raisonnable. Ce qui permet de définir des règles interphrastiques par exemple.

2 Éléments structurés

2.1 Étiquettes

L'information est donnée par la valeur d'une étiquette complexe. Ces étiquettes sont indépendantes des structures et constituent des objets en eux-mêmes. Une étiquette est composée d'un ensemble de variables affectées de valeurs. Ces valeurs sont dépendantes de la nature ou type des variables. Les types possibles comprennent bien sûr les types classiques entier, flottant, chaîne de caractères, vecteurs d'entiers, de flottants. Le type énumération se réalise en deux types : exclusif et non exclusif. Dans le type exclusif la variable peut prendre une seule des valeurs de références. Dans le type non exclusif la variable peut prendre comme valeur tout sous-ensemble des valeurs de références. Un type particulier pour les variables exclusives est le type potentiel. Pour ce type les valeurs de références sont non définies. Les valeurs sont accumulées au fur et à mesure de leur apparition. Cette propriété est importante pour la constitution de dictionnaire. L'inconvénient provient du fait qu'une faute dans la définition d'une valeur ne peut être détectée puisqu'elle est alors considérée comme une nouvelle valeur. Enfin il existe également un type "pointeur" appelé référence qui a comme valeur un moyen d'accéder à une autre étiquette.

Exemple : le verbe "manger" pourrait avoir comme étiquette associée

Catégorie(Verbe),Mode(Infinitif),Temps(Présent),Lemme(manger)

Les étiquettes seront nommées afin de pouvoir définir l'étiquetage d'une arborescence ou d'un élément structuré.

La définition précédente devient :

E1: Catégorie(Verbe),Mode(Infinitif),Temps(Présent),Lemme(manger)

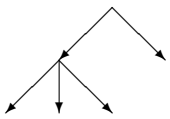
2.2 Arborescences

Une arborescence est un ensemble de points et d'arcs orientés reliant deux points, le précédent et le suivant. L'arborescence constitue un graphe connexe et possède les propriétés :

- Il existe un seul point appelé racine qui n'a aucun précédent.
- Tout point différent de la racine à un et un seul précédent.

Le point le plus important est la linéarisation : il existe une bijection entre les arborescences et l'ensemble des mots constitués de parenthèses équilibrées : le nombre de parenthèses ouvrantes est égal au nombre de parenthèses fermantes et toute parenthèse fermante apparait après la parenthèse ouvrante correspondante.

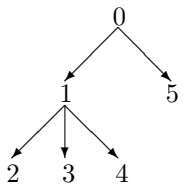
Exemple : le mot (((()())())()) correspond à l'arborescence :



2.3 Arborescences étiquetées

La décoration d'une arboresence est réalisée par une fonction d'étiquetage qui lui associe une étiquette à chacun de ces points. Aucun point n'a pas d'étiquette associée. Une étiquette peut être vide et la fonction d'étiquetage peut ne pas être injective ni surjective par rapport à l'ensemble des étiquettes. Cela signifie que plusieurs points peuvent être associés à la même étiquette, qu'il peut y avoir une étiquette pour laquelle il n'existe aucun point associé. Dans ce dernier cas l'étiquette doit être référencée par une valeur de variable d'une étiquette elle-même accessible.
Dans la représentation linéaire on fait précéder la parenthèse ouvrante correspondant à un point d'un nom d'étiquette. Ce nom peut être quelconque. Dans les sorties du système SYGMART il s'agit un simple numéro d'ordre.

Exemple : avec la structure précédente associée à la phrase "Le petit exemple expose"
nous aurons : 0(1(2()3()4())5()) / 0: Cat = Phrase; 1: Cat = GN; FS = SUJ; 2: Cat = Determ; Lemme = le; 3: Cat = Adjectif; Lemme = petit; 4: Cat = Nom; Lemme = exemple; 5: Cat = Verbe; Temps = Prsent; Personne = 3; Mode = Indicatif; Nombre = Singulier; Lemme = exposer
correspondant à l'arborescence étiquetée :



- 0: Cat = Phrase
- 1: Cat = GN; FS = SUJ
- 2: Cat = Determ; Lemme = le
- 3: Cat = Adjectif; Lemme = petit
- 4: Cat = Nom; Lemme = exemple
- 5: Cat = Verbe; Temps = Prsent; Personne = 3; Mode = Indicatif; Nombre = Singulier; Lemme = exposer

Évidemment dans une application réelle les étiquettes sont bien plus complexes.

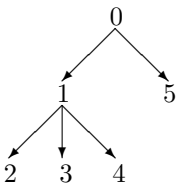
2.4 Éléments structurés

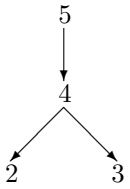
Un élément structuré est un triplet composé d'un ensemble d'arborescences, d'un ensemble d'étiquettes et d'une fonction d'étiquetage. Le nombre d'arborescences sera appelé dimension de l'élément structuré (Dans le système SYGMART ce nombre est limité à 16).

Exemple :
Pour l'exemple précédent, "Le petit exemple expose", au moins deux structures peuvent être définies, correspondant à deux interprétations, syntaxique et fonctionnelle :
nous aurons :

[1]: 0(1(2()3()4())5())
[2]: 5(4(2,3)) /
0: Cat = Phrase; 1: Cat = GN; FS = SUJ; 2: Cat = Determ; Lemme = le; 3: Cat = Adjectif; Lemme = petit; 4: Cat = Nom; Lemme = exemple; 5: Cat = Verbe; Temps = Présent; Personne = 3; Mode = Indicatif; Nombre = Singulier; Lemme = exposer

correspondant à l'arborescence étiquetée :





- 0: Cat = Phrase
- 1: Cat = GN; FS = SUJ
- 2: Cat = Determ; Lemme = le
- 3: Cat = Adjectif; Lemme = petit
- 4: Cat = Nom; Lemme = exemple
- 5: Cat = Verbe; Temps = Présent; Personne = 3; Mode = Indicatif; Nombre = Singulier; Lemme = exposer

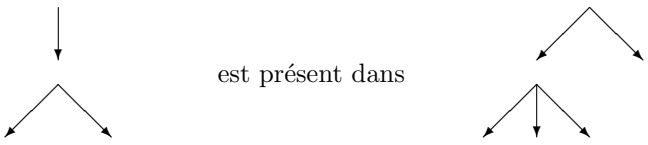
3 Schéma d'élément structuré

La grammaire structurelle est basée sur la réécriture. Aussi la reconnaissance d'élément partiel constitue son coeur opérationnel. Cette reconnaissance peut associer la consultation de dictionnaire ainsi que le calcul de prédicats.

3.1 Schéma d'arborescences

Un schéma d'arborescence est une arborescence ou une description synthétique d'un ensemble d'arborescences. Lorsqu'un schéma est une arborescence le test de sa présence dans une arborescence d'un élément structuré est simple : il doit y avoir identification des points et de la structure entre les deux éléments. Il faut remarquer que la présence d'une arborescence schéma dans une arborescence correspond à une décomposition du mot représentant l'arborescence. Tout point du schéma a une image dans l'arborescence et chaque point image a les mêmes dépendances que celles du schéma.

Exemple : le schéma (((()))) est présent dans l'arborescence (((()())())())



Le mot définissant l'arborescence est alors décomposé de la façon suivante :

- soit ((( ) ( ) ( )) ( ))
- soit ((( ) ( ) ( )) ( ))
- soit ((( ) ( ) ( )) ( ))

3.2 Schéma d'arborscences étiquetées

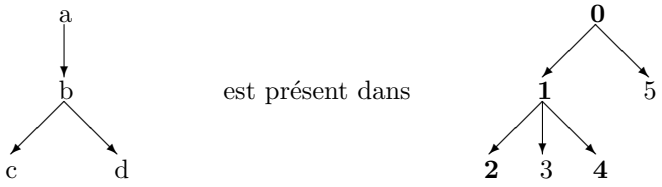
Lorsque l'arborescence est étiquetée la reconnaissance d'un schéma doit être en accord avec des prédicats associés aux points du schéma que doivent verifier les points images. On distingue deux types de prédicat : les prédicats propres et les prédicats inter sommets. Les prédicats propres sont très importants car ils limitent l'espace de recherche des différentes possibilités. Les prédicats inter-sommets permettent de sélectionner une solution dans l'espace des possibilités obtenues après filtrage conformément aux conditions propres. Pour définir un schéma étiqueté il faut bien sûr nommer les points. Les prédicats suivent alors la description structurelle et sont identifiés par le nom du point auquel il se rapporte.

Exemple : le schéma

a(b(c()d())) / a: Cat = Phrase; b: Cat = GN et FS = SUJ; c: Cat = Determ; d: Cat = Nom

est présent dans l'arborescence

0(1(2()3()4())5()) / 0: Cat = Phrase; 1: Cat = GN; FS = SUJ; 2: Cat = Determ; Lemme = le; 3: Cat = Adjectif; Lemme = petit; 4: Cat = Nom; Lemme = exemple; 5: Cat = Verbe; Temps = Présent; Personne = 3; Mode = Indicatif; Nombre = Singulier; Lemme = exposer



- a: Cat = Phrase
- b: Cat = GN et FS = SUJ
- c: Cat = Determ
- d: Cat = Nom

- 0: Cat = Phrase
- 1: Cat = GN; FS = SUJ
- 2: Cat = Determ; Lemme = le
- 3: Cat = Adjectif; Lemme = petit
- 4: Cat = Nom; Lemme = exemple
- 5: Cat = Verbe; Temps = Présent; Personne = 3; Mode = Indicatif; Nombre = Singulier; Lemme = exposer

Dans ce cas la décomposition est maintenant unique :

0(1(2 3 **4)** 5 **)**

3.3 Schéma d'élément structuré

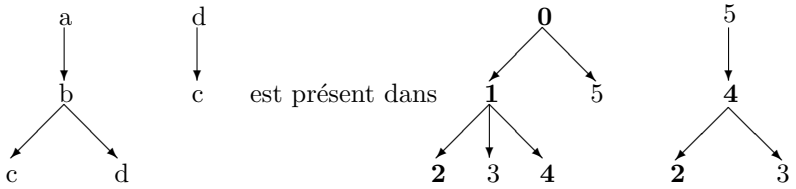
Un élément structuré comporte un ensemble d'arborescences. Le schéma comportera également plusieurs arborescences. Pour qu'un schéma soit reconnu il faut donc qu'il y ait une reconnaissance pour chaque schéma présent. La notion de dimension est alors importante. Le schéma décrit pour une ou plusieurs dimensions le schéma d'arborescence à reconnaître dans cette dimension. Une contrainte supplémentaire concerne la fonction d'étiquetage : si deux points de deux schémas d'arborescences portent la même identification alors ces deux points (de dimensions différentes nécessairement) doivent faire référence à la même étiquette. Un schéma sera reconnu si tous les schémas d'arborescences définis dans une dimension de l'élément structuré testé sont présents. Ainsi un schéma peut être reconnu s'il comporte plus ou moins de dimensions que l'élément structuré testé.

Exemple : le schéma

[1]: a(b(c()d())) / a: Cat = Phrase; b: Cat = GN et FS = SUJ; c: Cat = Determ; d: Cat = Nom
[2]: d(c)
On peut remarquer qu'ici il n'est pas nécessaire de définir des prédicats pour les points de la deuxième dimension puisque, portant le même nom que des points de la première dimension ils doivent faire obligatoirement référence à la même étiquette.

est présent dans l'élément structuré :

[1]: 0(1(2()3()4())5())
[2]: 5(4(2,3)) /
0: Cat = Phrase; 1: Cat = GN; FS = SUJ; 2: Cat = Determ; Lemme = le; 3: Cat = Adjectif; Lemme = petit; 4: Cat = Nom; Lemme = exemple; 5: Cat = Verbe; Temps = Présent; Personne = 3; Mode = Indicatif; Nombre = Singulier; Lemme = exposer



- a: Cat = Phrase
- b: Cat = GN et FS = SUJ
- c: Cat = Determ
- d: Cat = Nom

- 0: Cat = Phrase
- 1: Cat = GN; FS = SUJ
- 2: Cat = Determ; Lemme = le
- 3: Cat = Adjectif; Lemme = petit
- 4: Cat = Nom; Lemme = exemple
- 5: Cat = Verbe; Temps = Présent; Personne = 3; Mode = Indicatif; Nombre = Singulier; Lemme = exposer

Dans ce cas la décomposition est maintenant unique :

[1] **0(1(2** 3 **4)** 5 **)**
[2] 5(**4(2** 3 **)**)

3.4 Relaxation des contraintes dans un schéma de reconnaissance

L'extension de l'expressivité d'un schéma s'effectue par la définition de contraintes. Pour la description des contraintes on appelle point image le point de l'élément structuré testé qui correspond à un point du schéma :

- Le schéma peut être constitué d'une forêt : suite ordonnée d'arborescences. La reconnaissance d'une forêt correspond à la reconnaissance sous un même point et dans le même ordre des arborescences de la forêt.
- Deux points d'un schéma peuvent avoir l'obligation d'avoir leur image contiguë (en général il peut exister un nombre de points quelconque entre deux points images).
- Entre deux points ordonnés du schéma il est possible de définir un prédicat qui devra être vérifié par tous les points existants entre les deux points images.
- Un point du schéma peut être défini comme facultatif, c'est à dire ne pas avoir éventuellement d'image.
- Un ensemble de points dépendant d'un même point peut être considéré comme non ordonné.
- La dépendance entre un point et ses descendants peut être généralisée, c'est á dire qu'un descendant peut ëtre à une profondeur quelconque.

Dans la réalisation par le système SYGMART ces contraintes ont quelques incompatibilités entre elles. Par exemple le fait de supprimer les contraintes d'ordre, interdit la définition de contraintes prédicatives entre deux sommets désordonnés.

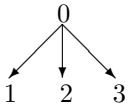
4 Transformation d'élément structuré

La notion de schéma correspond à une extension sur les éléments structurés de la notion d'infixe défini sur les mots. Les systèmes de réécritures définis sur la transformation de mots correspond à la substitution d'infixes. Par exemple, dans les algorithmes de Markov on substitue toujours l'infixe le plus à gauche du mot traité. Pour une définition de la transformation il faut donc définir la méthode de substitution ainsi que les règles de choix de l'occurence à traiter.

4.1 Transformation d'arborscences

Un schéma de transformation est défini par une arborescence. Lorsque un schéma est reconnu l'ensemble des points ayant un point image en racine sont supprimés et remplacés par l'arborescence de transformation. L'ensemble des points supprimés qui ne sont pas des points images sont alors distribués sur les points de l'arborescence de transformation. Pour définir cette distribution le schéma de reconnaissance et le schéma de transformation définissent implicitement des ensembles potentiel de forêts. On appelle "eliste" un triplet ordonné composé de une à trois références à des points d'un schéma. Le premier ne peut être vide et constitue la racine de la "eliste". Les deux autres peuvent être vides mais dans le cas contraire les points référencés doivent être dépendants dans le schéma de la racine de la "eliste", et être dans le même ordre que dans le schéma. Ces deux éléments seront placés entre deux chevrons derrière la racine.
Exemple :
Soit le schéma :

0(1,2,3)



L'ensemble des elistes induites par ce schéma est alors :

0<,>	0<,1>	0<,2>	0<,3>
0<1,>	0<1,2>	0<1,3>	0<2,>
0<2,3>	0<3,>	1<,>	2<,>
3<,>			

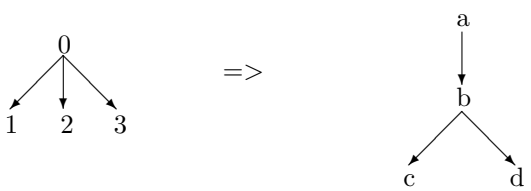
Une "eliste" stricte impose que les deuxième et troisième éléments soient contigus dans le schéma. C'est à dire que si le deuxi'eme est absent il n'existe pas dans le schéma de point à gauche du troisième et si le troisième est absent il n'existe pas de point à droite du deuxième.
Exemple :
Avec le schéma précédent l'ensemble des "elistes" strictes est le suivant :

0<,1>	0<1,2>	0<2,3>	0<3,>
1<,>	2<,>	3<,>	

Toute transformation comprend une fonction de l'ensemble des "elistes" du schéma de reconnaissance sur l'ensemble des "elistes" strictes du schéma de transformation et une relation d'ordre sur l'ensemble des "elistes" ayant la même image.

Soit la transformation :

0(1,2,3) => a(b(c,d))

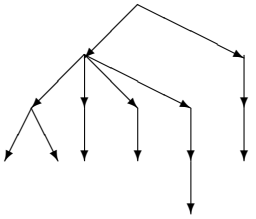


Les fonction et relations d'ordre :

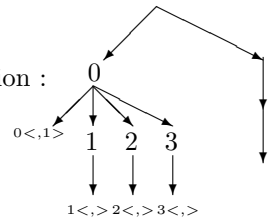
- fonction :

  - 0<,> − > a<b,>
  - 1<,> − > b<d,>
  - 2<,> − > c<,>
  - 3<,> − > c<,>
- relation d'ordre :
  - 2<,> > 3<,>

Alors la transformation appliquéee à l'arborescence :

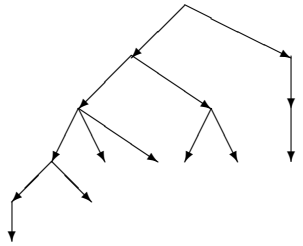


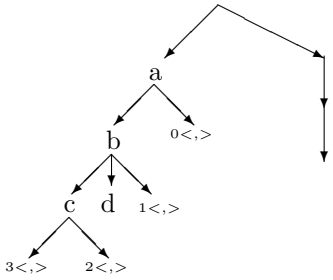
soit la décomposition :



(((( ) ( )) (( )) (( )) ((( )))) (( )))

produit le résultat :





(((( ) (( ))) ( ) ( )) (( ) ( ))) (( )))

4.2 Transformation d'arborescences étiquetées

La transformation d'une arborescence étiquetée comporte, en plus de la transformation, un traitement de l'association d' étiquetage. Les points de l'arborescence résultante qui ne proviennent pas directement d'un point du schéma de transformation conservent leur étiquette associée. Les autres sont associés à une étiquette vide lorsqu'il n'y a pas de précision particulière. Sinon pour un point provenant du schéma de transformation on peut lui associer :

- une étiquette qui était associée à un point du schéma de reconnaissance.
- une étiquette constante issue de la lecture d'un dictionnaire.
- une étiquette définie par la valeur d'une variable référence d'une étiquette associée à un point du schéma de reconnaissance.

Une fois cette association réalisée l'étiquette ainsi identifiée peut être modifiée. L'ordre dans lequel s'effectue cette association et donc la modification éventuelle d'une étiquette suit l'ordre canonique de la lecture du schéma de transformation (lecture de l'arborescence en profondeur d'abord, ou lecture de l'image chaîne de caractères de gauche à droite). Cette modification est définie par une suite d'affectations des différentes variables composant l'étiquette.

4.3 Transformation d'élément structuré

La transformation d'élément structuré effectue l'ensemble des transformations des différentes dimensions. L'ensemble des dimensions est ordonné et donc les transformations s'effectuent dans l'ordre de ces dimensions. Ce point est important lorsqu'il y a modification de la même étiquette dans plusieurs schémas de transformations pour des dimensions différentes. Plusieurs cas sont à considérer :

- l'élément structuré a le même nombre de dimensions que n'en comporte la règle de transformation : toutes les dimensions de l'élément structuré sont transformées.
- l'élément structuré a plus de dimensions que n'en comporte la règle de transformation : dans ce cas seules sont concernées les dimensions présentes dans la transformation.
- l'élément structuré a moins de dimensions que n'en comporte la règle de transformation : seules sont concernées les dimensions de l'élément structuré qui appartiennent également à la transformation.

4.4 Choix de l'occurence à transformer

Pour un schéma de reconnaissance dans une dimension il existe souvent plusieurs occurences possibles. L'application d'une règle s'effectue simultanément sur toutes les occurences possibles en respectant les contraintes suivantes :

- On applique toujours la règle sur l'occurence la plus haute dans l'arborescence puis la plus à droite dans le cas où il y aurait plusieurs occurences au même niveau.
- On applique la règle éventuellement sur une autre occurence à la condition que cette occurence n'ait aucun point commun avec les occurences déjà identifiées et en respectant la règle précédente en faisant abstraction des points des occurences identifiées.

5 Grammaire structurelle

La grammaire structurelle a pour but de définir un algorithme de transformation d'élément structuré. Cet algorithme peut correspondre à différentes tâches comme l'analyse syntaxique, l'analyse en dépendance, la traduction automatique, la fouille de texte ou le taggage. Une expérience pour la réalisation d'ihm avec la maîtrise de fichier html/xml a montré que l'utilisation de la grammaire structurelle n'était pas limitée au traitement de texte en langage naturel. Une des différences fondamentales de la grammaire structurelle avec les grammaires syntagmatiques réside dans le fait que contrairement aux autres formalismes la grammaire structurelle n'a pas d'algorithme propre et constitue plutôt un paradigme de programmation. La complexité de l'analyse est donc de la responsabilité du programmeur et n'a pas de contrainte particulière, elle peut aller de la linéarité (il est nécessaire au moins de lire l'élément) jusqu'à l'indécidabilité. Par exemple l'analyseur SYGFRAN écrit avec SYGMART a une complexité entre $n\log_2(n)$ et n^2 si n est le nombre de mots du texte analysé.

5.1 Grammaire élémentaire

Une grammaire élémentaire est composée d'une suite ordonnée de règles de transformation. L'ordre des règles définit la priorité d'application des règles. Pour une application de la grammaire toutes les règles applicables respectant la priorité des règles et le non chevauchement des schémas de reconnaissance sont appliquées. L'application d'une grammaire peut être définie par une suite transitive de transformations : Après l'application de la grammaire, la grammaire est de nouveau appliquée sur le résultat obtenu par cette première application et ainsi de suite. Une application constitue alors un pas de la grammaire. Une grammaire élémentaire a un mode d'application. On distingue trois modes d'applications.

- Controlée : le nombre de pas est borné par un entier. La grammaire produira le résultat soit parce que après la dernière application aucune règle n'est plus applicable, soit parce que le nombre de pas a atteint l'entier définissant la borne.
- Exhaustive : Après chaque application de la grammaire, les règles ayant participé à la transformation sont retirées de la grammaire pour les applications futures. Le nombre de pas est donc borné par le nombre de règles.
- Indécidable : La grammaire est appliquée transitivement jusqu'à ce qu'aucune règle ne soit applicable. Seul ce mode permet de simuler un algorithme de Markov quelconque. La grammaire correspondante peut bien sûr entrer dans une boucle infinie.

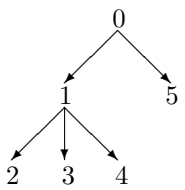
5.2 Composition de grammaires

La grammaire structurelle est composée d'un réseau conditionnel de grammaires élémentaires. On distingue donc des points d'entrée, des points de sortie et des points d'échec dans le réseau. Les arcs entre les grammaires du réseau sont conditionnels. Les conditions de cheminement sont définies par la présence ou nom d'un schéma d'élément structuré.L'application d'une grammaire consiste en un cheminement, avec éventuellement retour arrière, d'un point d'entrée à un point de sortie. Les arcs sortants d'une grammaire sont donc ordonnés et testés dans cet ordre. Le retour arrière annule donc le passage par un arc pour tester le suivant en ayant l'état de l'élément structuré restitué.

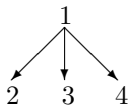
5.3 Récurrences

Les règles et grammaires élémentaires peuvent être récursives. Toute récurrence est définie à partir d'une dimension de l'élément structuré. On a donc une récurrence sur une arborescence. Dans l'appel récursif l'élément structuré traité sera donc celui existant avant l'appel où la dimension support de la récurrence est modifiée. Evidemment au cours de cet appel récursif un autre appel ayant comme support la même ou une autre dimension peut être effectué.

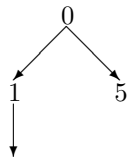
- La récurrence de grammaire.
  La recurrence de grammaire comporte deux appels, le premier vertical, le second horizontal.
  - La grammaire qui comporte une récurrence verticale sera appliquée sur l'élément structuré resultant de l'application de la grammaire appelée en recurrence. Cet appel s'effectuera sur l'élément structuré tel que l'arborescence de la dimension de recurrence ait pour racine le fils gauche de sa racine. Ainsi si la grammaire G appelle la grammaire G1 en récurrence verticale et que l'arborescence de recurrence est la suivante :



Alors G1 sera appliquée sur l'arborescence (les autres dimensions étant inchangées) :



Et la grammaire G sera appliquée sur le résultat :



résutat de l'application de G1

  - La grammaire qui comporte une récurrence horizontale est d'abord appliquée puis la grammaire appelé en recurrence horizontale sera appliquée avec comme racine le frère droit de la racine de l'arborescence sur laquelle cette grammaire a été appliquée. Cet appel n'a de sens que dans un suite d'appel récursif car au départ de la récurrence la racine n'a évidemment pas de frère.

- La récurrence de règle.
  Une règle qui comporte un appel récursif sera considérée comme appliquée lorsque le traitement de la grammaire appelée en récurrence aura été appliquée sur le résultat de l'application de cette règle.

6 Exemples

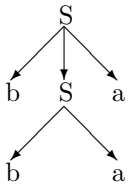
Pour définir les règles structurelles on utilise le symbole '*' dans la définition d'un schéma pour exprimé la contrainte de continuité. Ansi avec le schéma a(b,*,c) cela signifie que les point b et c doivent être contigu lors de la reconnaissance, en d'autre termes cela impose que l'élément de la décomposition a<b,c> soit vide.

6.1 Langage hors contexte

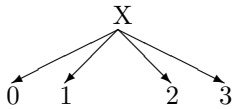
Soit la grammaire syntagmatique engendrant le langage $a^n b^n$:

S -> aSb
S -> ab

La structure syntaxique de a^2b^2 est alors :

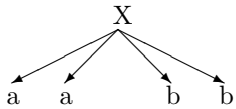


L'analyse du mot aabb par une grammaire structurelle nécessite de projeter ce mot dans une arborescence. La projection la plus simple est de placer une racine X au dessus du mot, de placer chaque caractere dans une étiquette et de construire un point descendant de X pour chaque étiquette ainsi construite. L'entrée de la grammaire sera donc :



- X: Val = X
- 0: Val = a
- 1: Val = a
- 2: Val = b
- 3: Val = b

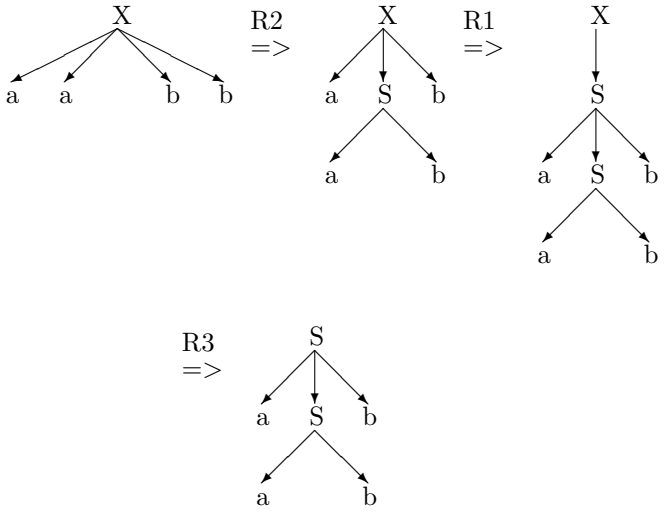
pour la simplicité nous représenterons l'élément par :



La grammaire d'analyse est alors la suivante :

R1: X(a,*,S,*,b) / X: Val = X; a: Val = a; S: Val = S; b: Val = b => X(S1(a,S,b)) / S1: Val = S.
R2: X(a,*,b) / X: Val = X; a: Val = a; b: Val = b => X(S(a,b)) / S: Val = S.
R3: X(*,S,*) / X: Val = X; S: Val = S => S.
On ne précise que l'étiquette qui doit être créée ou modifiée.

L'application donne :



Cet exemple permet de montrer quelques propriétés de la grammaire structurelle :

- Quelque soit le mot d'entrée la grammaire produira un résultat. Si le mot n'est pas du langage des constructions partielles seront réalisées. Cet aspect est très important pour l'analyse de la langue naturelle : si la phrase n'est pas reconnue, les constructions des groupes simples (groupes nominaux, groupes nominaux prpositionnels, groupes adverbiaux, ...) seront en grande partie construits, ce qui permettra une exploitation (dégradée) du résultat.
- Sur les mots qui n'appartiennent pas au langage le point X sera toujours présent. Cette propriété permet de définir un prédicat d'appartenance au langage.
- Pour les mots du langage le résultat sera leurs structures syntaxiques.
- A chaque pas de l'analyse une seule règle est appliquée. Il faut donc effectuer exactement n/2 applications plus une. Comme une application est linéaire cet algorithme a une complexité en n^2.

6.2 Langage sous contexte

Soit la grammaire syntagmatique engendrant le langage $a^n b^n c^n$:

S -> aSBC
S -> aBC
CB -> BC
aB -> ab
bB -> bb
bC -> bc
cC -> cc

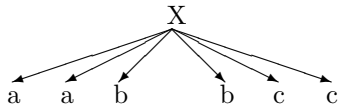
Bien sûr la strucure syntaxique ne peut être projective (la lecture du mot des feuilles correspondant au mot analysé). Avec la grammaire structurelle il est possible de reconnaître ce langage et de produire une structure sémantiquement acceptable : Sous un point les différents éléments auront le même ordre

d'apparition : Sous un pont S il y aura les a, b et c créés à l'origine par la même règle.
La grammaire d'analyse est alors la suivante (Les conditions ne sont pas notées par simplification d'ériture) :

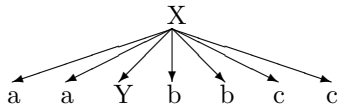
R1: X(a,*,b) => X(a,Y,b)
R2: X(b,*,c) => X(b,Z,c)
R3: X(a,*,Y,*,b,Z,*,c) => X(S(a,b,c),Y,Z)
R4: X(a,*,S,*,Y,*,b,Z,*,c) => X(S(a,S,b,c),YZ)
R5: X(*,S,*,Y,*,Z,*) => S

Une contrainte supplémentaire doit être définie : entre le point b et le point Z il ne peut exister de point Y ou Z.

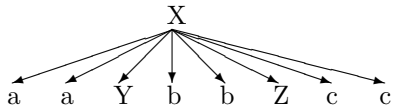
Cet algorithme reconnait alors le langage $a^n b^n c^n$ avec une complexité en n^2également.



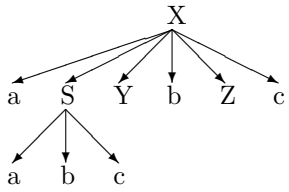
R1:



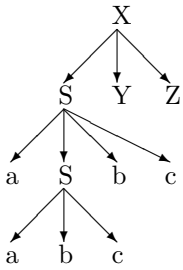
R2:



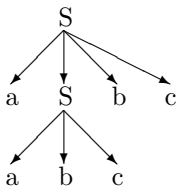
R3:



R4:



R5:



6.3 Algorithmique de l'analyseur SYGFRAN

L'analyseur SYGFRAN a une complexité proche de $n\log_2(n)$. Pour illustrer cette propriété il suffit de montrer la règle d'analyse d'une expression arithmétique d'une somme : $x + y + z + ...$

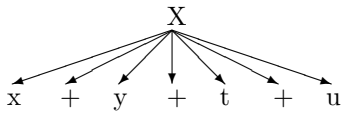
Le vocabulaire des étiquettes comprendra en plus le symbole E_+.

Une propriété des règles et de leurs schémas de reconnaissance consiste en la possibilité d'avoir une racine virtuelle : si dans le schéma X(A,B) X est virtuel il peut être partagé avec plusieurs autres schémas et donc plusieurs autres transformations. Dans ce cas le schéma sera noté simplement A,B et dans les elistes ce point apparaîtra avec un symbole particulier : @ en SYGMART.

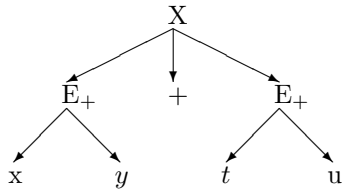
La eliste entre A et B sera donc notée : @<A,B>.

La règle d'analyse est alors la suivante :

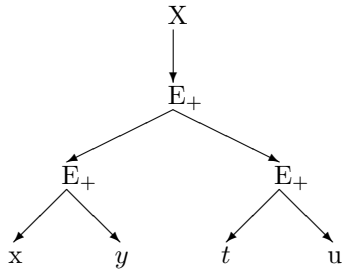
R: X,∗,+,∗,Y => E_+(X,Y) si X et Y sont étiquetés par des variables ou par E_+



R: Deux occurences du schéma ne se chevauchent pas et sont donc appliquées simultanément. La priorité de choix étant l'occurence la plus à droite nous avons d'abord le choix "t + u" puis le choix "x + y" :



R:



L'analyse d'une expression a donc une complexité en $n\log_2(n)$. La pluspart des règles de SYGFRAN fonctionnent de cette façon. Certains traitements locaux ainsi que des conditions inter-sommets alourdissent un peu le traitement.

6.4 Exemple de transformation SYGFRAN

RCOORDATTRIBENUM: 0,1(*,%2,*),*,3,*,4,*,5,*,%6,*,7,*,8,*,%9,*,10 /

0: (KPH = PH)&((TYP $>= ATTRIB)|(TYP $>= TRANSIND)|(TYP $>= TRANSIND-PASS));
3: (FLX = ':'); 4: (CAT = PONCT)&(SOUSP $>= ENUMERATION);
5: (K = GN);
6: (CAT = PONCT)&((SOUSP $>= CONJONCTION)|(SOUSP $>= DEFINITION));
7: (CAT = PONCT)&(SOUSP $>= ENUMERATION); 8: (K = GN);
9: (CAT = PONCT)&((SOUSP $>= CONJONCTION)|(SOUSP $>= DEFINITION));
10: (CAT = PONCT)&((SOUSP $>= ENUMERATION)|(SOUSP $>= TERMINAISON))

=> 0,1(%2),X(3,4,5,%6,7,8),%9,10 /

X:!5(COORDIN = C; NUM = PLU); 5:5(COORDIN = P);
8:8(COORDIN = P; FS(FREG1) = ATTR).

0(∗0<,>∗),∗<0,1>∗,1(∗1<,2>∗,2(∗2<,>∗),∗1<2,>∗),
X(3(∗3<,>∗),4(∗4<,>∗),5(∗5<,>∗),6(∗6<,>∗),7(∗7<,>∗),
8(∗8<,>∗)),9(∗9<,>∗),∗<9,10>∗,10(∗10<,>∗)

Le dernier élément est produit par le compilateur et permet de visualiser l'interprétation de la transformation produite par le système (notamment la fonction de projection).

=-=-=-=-=-=-=-=-=-=-=-=-=